reflex agent - doesn't think about consequences; bases only on current state

planning agent: maintains model; thinks ahead

worldstate - all info about state

search " - info necessary for planning

state space graph - states = nodes

- edges = actions
- directed edges from states to successors

search tree - node encodes path from start state to given state

Tree Search: pop partial plan (n) off fringe, replace on fringe w/ children (n+1)

completeness - guaranteed to find soln

optimality - lowest cost path

branching factor - increase in # nodes on fringe each time fringe node is dequeued.

DFS - deepest fringe node LIFO

- not complete (cycles)
- not optimal (leftmost)
- $O(b^m)$, depth m
- space $O(bm)$

BFS - shallowest fringe node FIFO

- complete
- optimal ~~(cost)~~ only when all edge weights are =
- shallowest soln depth = s
  time $O(b^s)$
- space $O(b^s)$

UCS - lowest cost fringe node
  weight = path from s to v

- complete
- optimal, if edges ≥ 0
- time $O(b^{C^*/\epsilon})$

  C^* is optimal path cost, minimal cost between nodes $= \epsilon$
- space: $O(b^{C^*/\epsilon})$

Greedy

- uses heuristic
- not complete/optimal

A*

- heuristic: path cost + heuristic
- complete + optimal

has cycles

Admissibility $\Rightarrow$ optimal A* tree search

$h^*(n)$ = true optimal forward cost from node n

$\forall n,\ 0 \leq h(n) \leq h^*(n)$

"underestimates" true cost

Graph Search

- Keep track of expanded nodes
- ensure next node isn't in closed set

Consistency $\Rightarrow$ graph search optimality/completeness

- underestimate difference in path cost between two nodes
- $\forall A, C\ \ h(A) - h(C) \leq cost(A, C)$

consistency $\Rightarrow$ admissibility

Heuristic A is dominant over B if estimated goal distance for A is greater than estimated goal distance for B for every node in state space graph

Max(admissible/consistent) = admissible/consistent

Search problems: state space, successor fn, start state, goal state

consistency $\rightarrow$ f value along path never decreases

CSPs

- variables: $X_1 .. X_N$
- domain
- constraints

Unary Constraint: single var

binary - two vars

Backtracking - assign only if value violates no constraints

Filtering - prune domains of unassigned variable by removing values that lead to backtracking

Fwd Checking - prune domains of all vars that share a constraint w/ current var.

OR

Arc Consistency: catches pitfalls steps ahead of fwd checking

remove value for X_i

if $X_i \rightarrow X_j$ is violated $\forall w \in X_j$

add all $X_k \rightarrow X_i$ to queue

$x \rightarrow y$ ← head

↑ tail

AC3: $O(ed^3)$ e = #arcs

n var $O(n^2 d^3)$ d = size(largest domain)

AC: fewer backtracks
  more holistic
  more computation

pick var

Minimum Remaining Values (MRV) (1)

- select variable w/ fewest remaining valid values to assign next

Least Constraining Value → pick val (2)

- choose which value to assign next; select value that prunes fewest values from domains of remaining unassigned values

Local Search: select var that violates most constraints and reset to value that violates fewest constraints (min-conflicts heuristic)

- incomplete, suboptimal

Minimax

- opponent behaves optimally

Utility - best possible outcome

terminal utility - deterministic known value
  ↑ inherent game property

$\forall$ agent controlled states, $V(s) = \max_{s' \in succ} V(s')$

$\forall$ opp. controlled states, $V(s) = \min_{s' \in succ} V(s')$

terminal states $V(s)$ known.

- behaves similarly to DFS: $O(b^m)$

α-β pruning: $\geq O(b^{m/2})$

- m is depth of terminal nodes

Eval fn: take in state and output estimate of true minimax value of node

- used in depth-limited minimax
- removes guarantee of optimal play

$Eval(s) = w_1 f_1(s) + \ldots + w_n f_n(s)$

f: feature from input state

w_i: weight of f_i

- higher scores for better positions

Expectimax

- Chance nodes: average case

$\forall$ chance states, $V(s) = \sum_{s' \in succ} P(s')V(s')$

- not much pruning except when ∃ known, finite bounds on possible node vals.

General Games

MDPs

- set of states S
- set of actions A
- start state
- ≥ 1 terminal state
- discount factor γ
- transition fn $T(s, a, s')$
- reward fn $R(s, a, s')$

q-states/action states = chance nodes (s, a)

- discounted utility is finite-valued iff $|\gamma| < 1$

Markov Property

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

solving MDP = find optimal policy

$\pi^*: S \rightarrow A$

V^* = max expected utility over all possible actions from s

$V^*(s)$ - optimal value of s (starting in s)

$Q^*(s,a)$ - optimal value of s, if taking action a

Bellman Eqn - condition for optimality

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

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

↓

expected utility

utility attained by acting optimally after arriving in s' from q-state (s, a)

Value Iteration - run until convergence $O(S^2 A)$

initialize $V_0(s) = 0\ \ \forall s \in S$

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

Convergence: $\forall s \in S,\ V_k(s) = V^*(s) = V_{k+1}(s)$

Policy Extraction - optimal policy if values are optimal

$\forall s \in S,\ \pi^*(s) = \arg\max_a Q^*(s,a) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

Policy Iteration

- policy converges faster than values

1) define initial policy
2) until convergence:

algebra

- policy evaluation: compute $V^\pi(s)\ \forall s$ → expected utility of starting in state s when following π

  $V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$

  or

  π_i = policy at iteration i.

  only one action at each iteration, no need for max.

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

- policy improvement

  $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$ → policy extraction

$\pi_{i+1} = \pi_i \Rightarrow$ convergence $\Rightarrow \pi_{i+1} = \pi_i = \pi^*$

RL

MDP: offline planning → knowledge of transition and reward fn

(s, a, s', r) := sample

Model Based - estimate transition/reward fns → solve MDP

" Free: estimate values/q-values directly

- get samples from episode, estimate transition + reward
- expensive to maintain counts for every (s, a, s') seen

} Model based

Model Free

Direct Eval

- passive (same policy)
- fix policy π and have agent experience episodes
- maintain total utility obtained from each state, # visits per state
- value = utility/visits
- one "bad" sample can bog down process

- calculate state values from episodes

TDL (Temporal Difference)

- passive
- learn from every experience
- uses exponential moving average

need optimal policy to converge to $V^*(s)$.

Sample = $R(s, \pi(s), s') + \gamma V^\pi(s')$. new estimate for $V^\pi(s)$

$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha \cdot sample$, α = learning rate $0 \leq \alpha \leq 1$

- typically start α at 1, slowly zero out
- older samples given exponentially less weight

Q-Learning

- finds you an optimal policy

must visit all state, action pairs.

Q-value iteration:

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

- max changes bc we transition before selecting new action in q-state.

q-value sample: $R(s,a,s') + \gamma \max_{a'} Q(s', a')$

$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot sample$ } update

→ off-policy learning := learn optimal policy by taking suboptimal or random actions

Approximate Q-Learning

- learn about few general situations and extrapolate.
- state represented as feature vector

  $V(s) = w_1 \cdot f_1(s) + \ldots + w_n \cdot f_n(s) = \vec{w} \cdot \vec{f}(s)$

  $Q(s,a) = w_1 \cdot f_1(s,a) + \ldots + w_n \cdot f_n(s,a)$

  $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot$ di

  difference = $[R(s,a,s') + \gamma \max_{a'} Q(s', a')] - Q(s,a)$ same for all weights

  $w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$ } update

- store only single weight vector, compute Q-values as needed
- more memory efficient.

ε-greedy
- explore w/ prob ε
- exploit w/ prob (1-ε)

Exploration Functions

$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot [R(s,a,s') + \max_{a'} f(s',a')]$

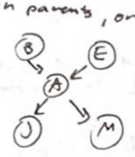
e.g. $f(s,a) = Q(s,a) + \frac{K}{N(s,a)}$

K := predetermined, N(s,a) := # time (s,a) has been visited

Bayes Nets
- DAG
- node is conditionally independent of all ancestor nodes in graph given all parents
- arrows represent conditional prob distr.

- node: $P(X \mid A_1, \dots, A_n)$; A_i is ith parent of X
- each node has a CPT (n+1 col)
  - n parents, one for self

e.g.



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

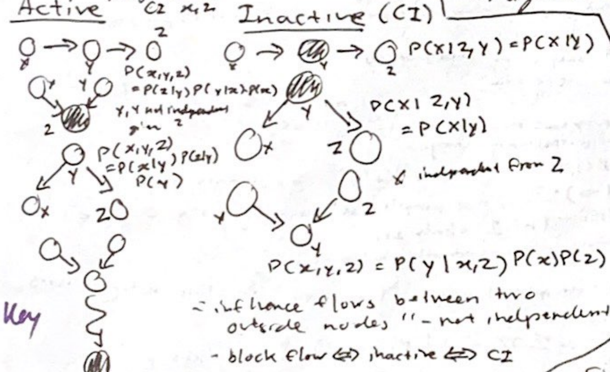
Inference - calculate joint PDF for some query var based on some set of observed var

Prior Sampling - sample many times
BN: generate too many samples

Rejection: reject samples inconsistent w/ evidence

D-Separation
- all tuples in a path are active
  ⇒ path is active, d-connects X to Y
  ⇒ X is not guaranteed conditionally independent of Y given observed nodes
- all paths X-Y are inactive
  ⇒ X and Y are CI given observed
  1 tuple is inactive

Active (not guaranteed CI X, Z)   Inactive (CI)



- influence flows between two outside nodes - not independent
- block flow ⇔ inactive ⇔ CI

∃ 1 path is active: independence not guaranteed

all paths inactive: independence guaranteed

path is active if every tuple is active

path is inactive if one tuple is inactive

middle node observed ⇒ tuple is active

---

joint distr over N boolean vars: 2^N

N-node net if nodes have ≤ K parents is $O(N \cdot 2^{K+1})$

Marginal Distribution

$P(s) = \sum_t P(t,s)$

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

$P(x,y) = P(x|y)P(y) = P(y|x)P(x)$

$P(x_1,x_2,x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)$

$P(x_1,\dots,x_n) = \prod_i P(x_i|x_1,\dots x_{i-1})$

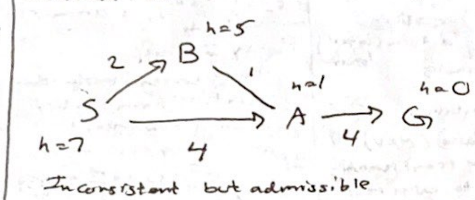
$P(x|y) = \frac{P(y|x)}{P(y)}P(x)$

x and y are independent if $P(x,y) = P(x)P(y)$, $\forall (x,y)$

X and Y are CI given Z iff $P(x|y,z) = P(x|z)$

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

CSP: n vars, d possible vals - backtracking may backtrack: $O(d^n)$



Incosistent but admissible

- search is equivalent if edge weights are changed in same way (multiplication). DFS BFS UCS
- nodes costs are added/subtracted by same number
- find by doing sum } ehh.

def $h(j) - h(i)$ is telescoping

- heuristic must return 0 in goal state
- consistent:
  doesn't decrease from state to state by more than it actually costs to get from state to state
- fully connected (complete) graph can represent joint distribution
- marginalization renders neighbors of marginalized out var dependent } marginalized
- conditioning fixes observed vars and renders ancestors dependent according to rules of d-separation } condition - ancestors dependent

- adding a constant to all h for all nodes doesn't change order of expansion

find Q value then V.

Q-learning:

move $Q(s,a)$ toward $R(s,a) + \gamma \max_{a'} Q(s',a')$

Expected utility:

$\sum_P P(U) \cdot U(x)$

1) stuck in infinite loop in finite graph w/ positive edge weights when no soln exists
2) infinite loop, finite graph, + edges, ∄ soln
3) infinite loop, infinite graph, + edges, ∃ soln
4) guaranteed to find optimal soln when edges < 1, ∃ soln
5) + edges, soln exists

$X \perp\!\!\!\perp Y \mid Z$:

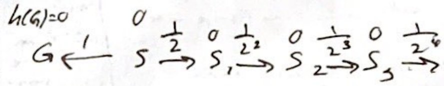
$P(X|Y,Z) = P(X|Z)$

$P(X,Y,Z) = \frac{P(X,Z)\,P(Y,Z)}{P(Z)}$

α-β pruning:
- node is pruned from under max node if it knows that min node above it has smaller value to pick than the value max node just found
- node is pruned from min node if it knows that max node above it has larger value to pick than the value min node just found

---

Infinite state space



$\sum_{j=1}^{k} \frac{1}{2^j} = 1 - \frac{1}{2^k} < 1$

$1 + \frac{1}{2} + \frac{1}{4} + \dots = 2.$

DFS Tree: 1, 2, 3   UCS Graph: 4, 5
BFS Tree: 1, 4
UCS Tree: 1, 4, 5
A* Tree: 1, 4, 5
DFS Graph: 3
BFS Graph: 4

When finding equivalent BN, if no delim var conditional independence remains, fill in all other edges.

Else - fill in only those CI.

more edges = less constraint

Fwdchecking makes current var are consistent

---

valid joint distr.
- each variable appears on left of | given once
- no cyclic conditioning

nondeterministic transition ⇒ ∞ iterations

α-β

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each suc of state:
        v = max(v, value(suc, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

min: +∞, min, ≤

188

Prior Sampling - normal sampling

Rejection - early reject sample inconsistent w evidence

Likelihood Weighting - manually set all variables equal to evidence in query

- use weight for each sample: $P(e \mid \text{Sampled})$

Iterate thru vars:

- sample value if var $\neq$ evidence
- change weight of sample if var = e_i

Gibbs - first set all vars to some random val

- repeatedly pick one var at a time, clear val, resample it given all other assigned values

↳ change at most one var

Decision Networks

- chance nodes ○
- action □
- utility ◇

$EV(a \mid e) = \sum_{x_1,\dots,x_n} P(x_1,\dots,x_n \mid e)\, U(a, x_1,\dots,x_n)$

action a, evidence e, chance nodes $x_{1..n}$

$MEU(e) = \max_a EU(a \mid e)$

VPI: amount MEU is expected to increase if observe new evidence

Expected value of MEU:

$MEU(e, E') = \sum_{e'} P(e' \mid e)\, MEU(e, e')$

$MEU(e, e') = \max_a \sum_s P(s \mid e, e')\, U(s, a)$

$VPI(E' \mid e) = MEU(e, E') - MEU(e)$

↳ nonnegative: always added value - can always be zero

↳ $VPI(E_j, E_k \mid e) \neq VPI(E_j \mid e) + VPI(E_k \mid e)$

b/c observing e_j might cause us to not care about e_k

↳ order-independent. order of observation doesn't matter

$VPI(E_j, E_k \mid e) = VPI(E_j \mid e) + VPI(E_k \mid e, E_j)$

$= VPI(E_k \mid e) + VPI(E_j \mid E_k, e)$

Markov Models

- chain-like, infinite length Bayes net
- time dependent
- Markov (memoryless property)
- stationary transition model

⇒ represented w/ $Pr(W_0)$, $Pr(W_{t+1} \mid W_t)$

Mini Fwd Algo

$Pr(W_{i+1}) = \sum_{w_i} Pr(W_i, W_{i+1})$

$= \sum_{w_i} Pr(W_{i+1} \mid W_i)\, Pr(W_i)$

→ look at prob distr at time i, advance model by 1 timestep using $Pr(W_{i+1} \mid W_i)$

Stationary Distr

$Pr(W_{t+1}) = Pr(W_t) = \sum_{w_t} Pr(W_{t+1} \mid W_t)\, Pr(W_t)$

solve w/ system: $\forall i,\ Pr_\infty(W_t) = Pr_\infty(W_{t+1})$

HMM

- allows us to observe evidence at each timestep

$$\begin{array}{ccccccc} W_0 & \to & W_1 & \to & W_2 & \to & W_3 \to \\ & & \downarrow & & \downarrow & & \downarrow \\ & & F_1 & & F_2 & & F_3 \end{array}$$

W_i: state var

F_i: evidence var

$F_1 \perp\!\!\!\perp W_0 \mid W_1$

$W_i \perp\!\!\!\perp \{W_0, \dots, W_{i-2}, F_1, \dots F_{i-1}\} \mid W_{i-1}$

$F_i \perp\!\!\!\perp \{W_0, \dots, W_{i-1}, F_1, \dots, F_{i-1}\} \mid W_i$

Stationary: $Pr(W_{i+1} \mid W_i)$

$Pr(F_i \mid W_i)$ - sensor model

Belief Distr: $B(W_i) = Pr(W_i \mid f_1, \dots, f_i)$ all evidence observed

$B'(W_i) = Pr(W_i \mid f_1, \dots, f_{i-1})$ $f_1, \dots, f_{i-1}$ obs.

Fwd Algo

$B'(W_{i+1}) = Pr(W_{i+1} \mid f_1, \dots, f_i) = \sum_{w_i} Pr(W_{i+1}, w_i \mid f_1, \dots, f_i)$

$= \sum_{w_i} Pr(W_{i+1} \mid w_i, f_1, \dots, f_i) \cdot Pr(w_i \mid f_1, \dots, f_i)$

$= \sum_{w_i} Pr(W_{i+1} \mid w_i)\, B(w_i)$

$B(W_{i+1}) = Pr(W_{i+1} \mid f_1, \dots, f_{i+1}) = \frac{Pr(W_{i+1}, f_{i+1} \mid f_1, \dots, f_i)}{Pr(f_{i+1} \mid f_1, \dots, f_i)}$

$\propto Pr(W_{i+1}, f_{i+1} \mid f_1, \dots, f_i)$

$= Pr(W_{i+1} \mid f_1, \dots, f_i)\, Pr(f_{i+1} \mid W_{i+1}, f_1, \dots, f_i)$

$= Pr(f_{i+1} \mid W_{i+1})\, B'(W_{i+1})$

$B(W_{i+1}) \propto Pr(f_{i+1} \mid W_{i+1}) \sum_{w_i} Pr(W_{i+1} \mid w_i)\, B(w_i)$

Time Elapse: $B'(W_{i+1})$ from $B(W_i)$

Observation: $B(W_{i+1})$ from $B'(W_{i+1})$

→ normalize

Particle Filtering

- approximate desired distr
- store n particles; in one of d states
- $n \ll d$

Time Elapse

- particle in state t_i
- Sample from $Pr(T_{i+1} \mid t_i)$

Observation

- assign weight of $Pr(f_i \mid t_i)$ for particle in state t_i w/ reading f_i
- calculate total weight for each state
- normalize distr, resample

Supervised - input/output ↳ predict outputs training weights

Unsupervised - doesn't have corresponding output

training data - used to map inputs to outputs

validation data - measure model performance by making predictions on inputs

- adjust hyperparameters

test set - final exam

Naive Bayes

- classification: group into classes (label)

features - attributes of data used to learn

- Naive: each feature F_i is CI of all other features given class label
- table: $P(Y)$ = 2 entries

$P(F_i \mid Y)$: 4 entries

total 4n+2 entries, n = # features

prediction$(f_1, \dots, f_n) = \arg\max_y P(Y=y \mid F_1 = f_1, \dots, F_n = f_n)$

$= \arg\max_y P(Y=y, F_1 = f_1, \dots, F_N = f_n)$

$= \arg\max_y P(Y=y) \prod_{i=1}^{n} P(F_i = f_i \mid Y=y)$

observations drawn from distr parameterized by θ

Maximum Likelihood Estimate learns θ

- each x_i is independent, identically distributed
- all values of θ are equally likely (uniform prior)
- likelihood $\mathcal{L}(\theta)$: fn that represents probability of having drawn sample from distr

$\mathcal{L}(\theta) = P_\theta(x_1, \dots, x_N)$ N samples

$= \prod_{i=1}^{N} P_\theta(x_i)$

MLE: $\frac{\partial}{\partial \theta} \mathcal{L}(\theta) = 0$

overfitting: building a model that doesn't generalize well to unseen data

Laplace Smoothing:

- strength k - seen k extra of each outcome

$P_{MLE}(x) = \frac{\text{count}(x)}{N}$

$P_{LAP,k}(x) = \frac{\text{count}(x) + k}{N + k|X|}$

$|X|$: number of classes e.g. words

$P_{LAP,k}(x \mid y) = \frac{\text{count}(x, y) + k}{\text{count}(y) + k|X|}$

$P_{LAP,\infty}(x) = \frac{1}{|X|}$ N

N: sample size e.g. # words in data

Linear Classifiers classify using linear comb of features (activation)

- takes in data, multiplies each feature of data point by corresponding weight

$\text{activation}_w(x) = \sum w_i f_i(x)$

$\text{classify}(x) = \begin{cases} + & \text{if act} > 0 \\ - & \text{if act} < 0 \end{cases}$

does feature vector point in same dir (<90°) as predefined weight vector?

$\text{classify}(x) = \begin{cases} + & \cos\theta > 0 \\ - & \cos\theta < 0 \end{cases}$

if act $= \|w\|\,\|f(x)\| \cos\theta = 0$, feature vec $\perp w$

- line/vector $\perp w$ separates +/- boundary "decision boundary"

Binary Perceptron

- find decision boundary that separates data
- find best possible weights, $\vec{w}$, st any training point is correctly classified

Algo:

1) initialize all weights to 0.
2) For each sample: w/ features f
   - classify sample using weight

   $y = \text{classify}(x)$

   - if $y \neq$ true label y^*

   $w \leftarrow w + y^* \cdot f(x)$
3) terminate when all samples don't require weight adjustment

update = balancing

1) misclassified + as -: $w \leftarrow w + f(x)$
2) misclassified - as +: $w \leftarrow w - f(x)$

Bias: for when boundary doesn't go thru origin

- add feature to sample feature vector that is always 1, add extra weight in weight vector

$w^T f(x) + b = 0$.

Multiclass

- one weight vector for each class

$\text{classify}(x) = \max_i w_i^T f(x)$

- class w/ max weight vector

= "stack" vectors into matrix

↳ matrix-vector multiply w/ $f(x)$

- argmax resulting vector

Weight Update:

- add feature vector to weight vector of y^*
- subtract $f(x)$ from weight vector of prediction $\hat{y}$

e.g. $w_0 = [-2\ 2\ 1]$

$w_1 = [0\ 3\ 4]$

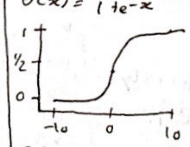
$w_2 = [1\ 4\ -2]$

$W = \begin{bmatrix} -2 & 2 & 1 \\ 0 & 3 & 4 \\ 1 & 4 & -2 \end{bmatrix}$

↳ reward correct weights, punish misleading one

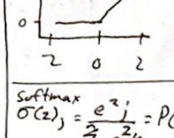
Sigmoid

$\sigma(x) = \frac{1}{1 + e^{-x}}$



ReLU

$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$



Softmax

$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=0} e^{z_k}} = P(\text{class } j \mid z)$

$= P(\text{class } j \mid f(x))$

Likelihood of weight w explaining observations

$\ell(w) = \prod_{i=1}^{m} P(y^i \mid f(x^i); w)$

- want weight that maximizes
- log-likelihood

$\ell\ell(w) = \log \ell(w)$

$= \sum_{i=1}^{m} \log P(y^i \mid f(x^i); w)$

$\nabla_w \ell\ell(w) = \left[\frac{\partial}{\partial w_1} \ell\ell(w) \dots \frac{\partial}{\partial w_n} \ell\ell(w)\right]$

Gradient Descent

- find direction of steepest descent

Initialize weights w

For i = 0,1,2,...

$w \leftarrow w - \alpha \nabla_w f(w)$

Back prop

- efficiently calculate gradients for each parameter in NN
- value after each node is partial deriv of last node's value wrt that node's value
- gradients of nodes w/ multiple children are sum of gradients of children

$$\begin{array}{l} x \longrightarrow h \dots f \\ \quad \searrow i \dots f \end{array}$$

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial h}\frac{\partial h}{\partial x} + \frac{\partial f}{\partial i}\frac{\partial i}{\partial x}$

In NN, want to find parameters w that maximize likelihood of the class probabilities

- minimize loss

MLE likelihood fn

maximize P(data | evidence)

$A_{jk} = A^T_{kj}$

Matrix Multiplication:

$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$

Matrix Vector Multiplication

Ax

$\sum_{j=1}^{m} A_{ij} x_j$

matrix multiplication

$$\begin{bmatrix} \sum_{j=1}^{m} (A_{1j} + A_{1j}^T)\, x_j \\ \vdots \\ \sum_{j=1}^{m} (A_{mj} + A_{mj}^T)\, x_j \end{bmatrix} = (A + A^T)\, x$$

Chain Rule

$P(A_4, A_3, A_2, A_1)$

$= P(A_4 \mid A_3, A_2, A_1)$

$\cdot P(A_3 \mid A_2, A_1)$

$\cdot P(A_2 \mid A_1)$

$\cdot P(A_1)$

Naïve Bayes - overconfident
→ low entropy posteriors

the more naïve, the more overconfident,
the lower entropy posterior is

posterior - conditional probability
after relevant evidence
is taken into account

$$\sum_B P(B|A) = P(+b|A) + P(-b|A) = 1$$

backward pass step for a node depends on node's inputs and gradients computed downstream of current node.

Viterbi: finding most likely path to a node

$$m_t[X_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t})$$
$$= P(e_t|x_t) \max_{x_{t-1}} P(X_t|x_{t-1}) m_{t-1}[x_{t-1}]$$

Most likely sequence of hidden states $X^*_{1:N}$
for $t = 1$ to N:
    for $x_t \in X$ (states at t):
        if $t = 1$:
            $m_t[X_t] = P(x_t) P(e_0|X_t)$
        else
            $a_t[X_t] = \operatorname{argmax}_{x_{t-1}} P(X_t|X_{t-1}) m_{t-1}[X_{t-1}]$
            $m_t[X_t] = P(e_t|X_t) P(X_t|a_t[X_t]) m_{t-1}[a_t[X_t]]$
this gives path's most likely ending point

for $t = N$ to 2:
    $X^*_{t-1} = a_t[X_t^*]$

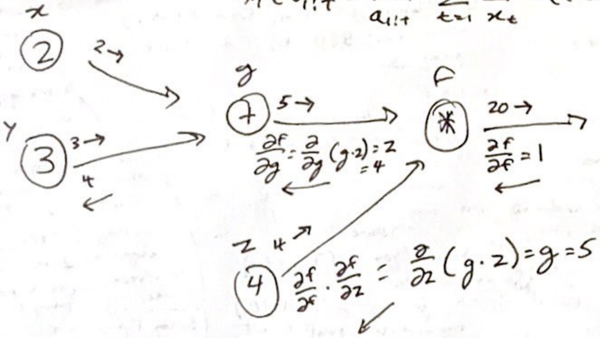
Variables that are not in union of ancestors of query and ancestors of evidence can be pruned.

parameters - # of quantities needed to specify model ~~[illegible]~~
(size of CPT)

inference by enumeration: full joint computed

output of softmax is probability distribution
elements between 0 and 1

$$MEU_{1:T} = \max_{a_{1:T}} \sum_{t=1}^{T} \sum_{x_t} B_t(x_t) U_t(x_t)$$



$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial y} = z \cdot \frac{\partial}{\partial y}(x+y) = z = 4$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} = z \cdot \frac{\partial}{\partial x}(x+y) = z = 4$$

$$g(x) = \frac{1}{1+\exp(a-x)}, \quad \frac{\partial g}{\partial x} = g(x)(1-g(x))$$

$$\frac{e^{r_1}}{e^{r_1} + e^{r_2}} = \frac{1}{1+e^{(r_2 - r_1)}}$$

$$g(z) = \frac{1}{1+\exp(-z)}$$
$$\frac{\partial g}{\partial z} = g(z)(1-g(z))$$

---

calculate joint: if binary,
size of $2^{\#\text{ vars that are not set}}$
↑ or # possible vals

$$P(Y|X_1, X_2) = \frac{P(Y, X_1, X_2)}{P(X_1, X_2)}$$

Label
Class
$\begin{bmatrix} F_1 \\ \vdots \\ F_n \end{bmatrix}$ → Naïve Bayes → Classification
features of sample

Given many samples, which you know true classifications of, how many times does Naïve Bayes accurately classify sample?

Particle Filtering / $P_1: W_1 = 0$, $P_2: W_1 = 1$ — sum weights in state

1. Compute weight of two particles after evidence $O_1 = A$
   $w(P_1) = P(O_t = A | W_t = 0) = 0.9$
   $w(P_2) = P(O_t = A | W_t = 1) = 0.5$ — state
2. Using random numbers, resample P_1 and P_2 based on weights.
   Normalize weights... P_1: $[0, 0.643)$ P_2: $[0.643, 1)$
   $P_1 = \text{sample}(\text{weights}, 0.22) = 0$
   $P_2 = \text{sample}(\text{weights}, 0.05) = 0$
3. Elapse Time: Sample P_1 and P_2 from applying time update
   $P_1 = \text{sample}(P(W_{t+1}|W_t) = 0), 0.33) = 0$
   $P_2 = \text{sample}(P(W_{t+1}|W_t = 0), 0.2) = 0$
4. Observe: Compute the weight of the two particles after evidence $O_2 = B$
   $w(P_1) = P(O_t = B | W_t = 0) = 0.1$ } from table
   $w(P_2) = P(O_t = B | W_t = 0) = 0.1$
5. Resample P_1 and P_2 based on weights
   - both $P_1, P_2 = 0$, so resampling leaves us w/ $P_1 = 0$, $P_2 = 0$
6. Estimated Distr for $P(W_2 | O_1 = A, O_2 = B)$:
   $P(W_2 = 0 | O_1 = A, O_2 = B) = 2/2$
   $P(W_2 = 1 | O_1 = A, O_2 = B) = 0/2$