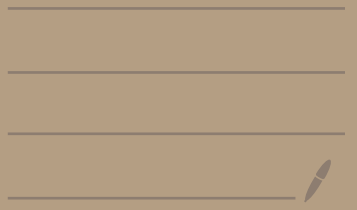
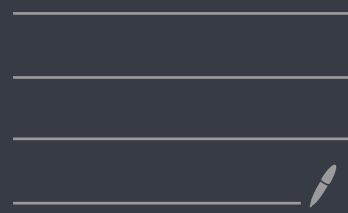
CISC 352



Unit 1



Week 1: AI origin, Agent Types and Basic Search

Rational Decisions

- goals are expressed as utility of outcomes
  - ↳ maximizing expected utility could be a danger to other things
- So constraints are very important
- Rational agents are only good at one utility

Heuristic Searching

- rule of thumb, helps make good decisions faster
- changes how fast you find an answer

{ which var to assign next?
  which to try first?

- Reflex Agents react only to current state
- Planning Agents have a goal, think ahead (search algorithms)

Search Problems

- Initial State
- Actions / Succesor functions
- Transition model
- goal test
- path cost (optional)

State Space

- set of all possible states
- can be huge
- Can have cycles
- Search trees represent paths to a node

Slide 102.

CSP's

- Constraint Satisfaction Problems
  - ↳ must assign values to variables
  - ↳ all constraints must be satisfied
  - ↳ There is no "best" solution. (just valid or invalid)
- Defined by three things:
  1) Variables X, Y, Z
  2) Domains $x \in \{1, 2, 3\}$
  3) Constraints $x \neq y$

Examples

- Class scheduling
- Sodoku

CSP's are smart pruning huge search trees

Search Problems

- AI explores all possibilities
- Each node = partial assignment
- Root = nothing assigned
- Leaf = all variables assigned

Backtracking → naive DFS

↳ one variable at a time

↳ finds issue earlier by back tracking to find problem

↳ check constraints as you go (incremental goal test)

DFS with 2 improvements → Extremely Powerfull

Back tracking Pseudocode *

Improving Backtracking:

- Ordering
- Filtering

Filtering

- forward checking: cross off values of all possible affected
- variable with domain size 1, you choose that one, when you see no possible unassigned value, then stop back track, try again.

Arc Consistency: only cutting from tail

when you check an arc and things are good, then you change smth else, you may need to check it again. Each change is independent.

ARC - Preprocessor

ARC is more checking goal / rules

(does not detect soon to be failures)

Forward Checki + ARC consistency

Lot of relationship between K → node(n)

Must know K for ARC consistency.

CSP Quiz Review

- goals are expressed in terms of utility (numbers of how good each outcome is)
- maximize expected utility, choose best average outcome.
- Rationale = perfectly optimized for goal, does not care about side effects.
- Rationale agen has perception and environment.
- Agents have sensors called percepts and actuators make actions
- Heuristic Search: iteratively improving solutions based on heurisiz function
- Bayesion Networks: probabalistic model represenring variables

---

Search

- Reflexive Agent (based on current percepts) does not care about consequences
- Planning Agents (considers future consequences, "what if". world "would look like.
- Search problem:
  - State space → set of all possible configs
  - Success function → available actions and their cost
  - Start state and goal state → start condition and tests to try goal
  - solution is a sequence of actions that transforms start state to goal state

State Abstraction

- world state vs state space
  - includes details about environments
  - abstraction with only details needed for planning
- if world states are independent, can multiply the state of each object.
- State space graph: each state occurs only once → only one
- Search tree: "what if" represents planning and their outcome → multiple

Key Distinction: single state can appear multiple times, because they represent a distinct path, search trees can be infinite.

- Search trees can have cycles (infinite loop)
- Search tree is generated from state space graph, expands paths

General Tree Search

- Expansion → expand potential options
- Fringe: collection of partial plans under consideration for expansion (chopping block)
- Exploration strategy / heuristic
- Note: failure happens when no more nodes on fringe.

Constraint Satisfaction Problems

- Find valid state not specific path
- Defined by: Variables, Domains, Constraints
- Standard Searches: concern about path to goal, CSPs: only care about satisfying all constrains and solutions are found at similiar levels.
- State is assignment of values to variables

Partial ↓ ↳ complete

Arcs are edges = constraints between variables
(tells you which nodes depend on each other)

Varieties of CSP's and Constraints

- variables can be discrete / continous
- Unary constraint: only one var involved
- Binary cons.: relates two vars
- High order: 3 or more
- Soft cons: preferences may be better (more optimal)

Backtracking Search:

- DFS: one var at a time
- order of assignment does not change final state
- Incremental constraint checking

Techniques to help Backtracking

- Forward checking: Keeps track of remaining valid domains for unassigned variables
  ↳ does not provide early detection for all failures

- Arc Consistency: An arc $X \to Y$ is consistent, if for every X, there at least one Y, which satisfies constraint If value is removed, recalibrates for all neighbors.
  ↳ can be run as a pre-processor.

- Backtracking is skeleton: AC-3 / FC are plug ins
  * Try to understand code

- K-Consistency: generalizes to larger groups of nodes, 1 consistency → node level, 2 consistency → arc level, K-consistency → K-1. Extendeable
  K=2 → Arc Consistency: · detects multivariable conflicts earlier.

- Branching factor in CSP's are size of domain of the var you are assigning
- Which variable you assign changes branching factor early.
  (matters when domain are large

GAC (generalized Arc Consistency)

- any assignment to node can be extended to all other var's sharing constraints. Involving hyper arcs.
- Hyper Arc: connects one variable to constraint involving multiple variables
- DFS: Expand deepest node first using Lifo stack for fringe
  ↳ not optimal, space advantage $O(bm)$ b-branching factor, m- max depth
- BFS: Expands most shallow node first, space complexity of $O(b^s)$ where s is depth
- Iterative Deepening: combines space efficiency of DFS with optimality of BFS. By increasing depth limit.
- Uniform cost search: Expand cheapest node using Priority Queue.
  Both complete and optimal for positive Arc costs.
- Heuristic deciding how to move to next node.

GAC - Pseudocode

Pruning s : [ ]
Queue: [ ]

if newVar is True:
    queue = csp. get all constraints
else:
    queue = csp. get var and constraints
for cons in queue
    for var in cons
        if val has support:
            Pass
        else: val. prune
            Prunings. append (val)

        if var. curr domain size == 0:
            return False, prunings

            For cons in CSP. get cons_with var
                if cons not in que
                que. append

return True, prunings

Value V in var X is gac-consistent w.r.t C iff there exists at least one combination of values

- DFS > BFS, when solution is very deep (low memory)
- BFS > DFS, solution is shallow, shorter path needed
- Iterative deepening runs DFS, wit depth lim

Uniform Cost Search: expands node with lowest path cost so far. Uses priority queue ordered by g(n) = cost from start to n.

- Effective depth of C^*/ϵ. Takes $O(b^{C^*/\epsilon})$. C^* is cost, ϵ is smallest step. How many cost layers before optimal solution.

- Greedy Search: expand node that seems closest to heuristic, not optimal and can be led astray. h(n) = estimated distance to goal. "what looks best"

- A* Search: combines actual cost so far and estimated cost to goal
  $f(n) = g(n) + h(n)$, g(n) is backward cost and h(n) is forward cost.
  Only stop when goal is decued from fringe.

  h - remaining cost to goal A* is optimal when heuristic is admissable

Heuristic Properties

- Heuristic is admissable if it is optimistic. Does not over estimate.
  - ↳ $h(n) \leq h^*(n)$, where $h^*(n)$ is true cheapest cost.
    - $h(n)$ is remaining cost from node n to goal
    - $h^*(n)$ is true cheapest cost to goal.

- Consistency: Heuristic is consistent if estimated cost is less than or equal to to f value never decreases along a path.

  Consistent ⇒ Admissable

- Relaxed Problems: easier versions of problems.
  - As you move closer to goal, your heuristic estimate cannot drop by more than the cost of that step. $F(n) = g(n) + h(n)$ never decreases along path.
  - Will never find cheaper path later.

Slide 208 - 215

- UCS expand equally, A* towards goal.
- all search algos are same with diff fringe strategies

Graph Search

- Tree search explores paths without remembering visited states → same tree, can appear multiple times
- Graph search remembers visited States → avoids cycles

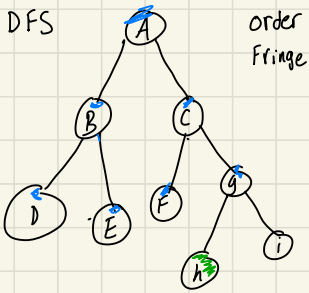
- A* Tree Search - Admissible heuristic is enough
- A* Graph search - requires consistency for optimality and efficiency.

- Tree Search Pseudocode
  - ↳ Starts from initial state
    - for every
      - Pick node from fringe
        - check if its goal
        - expand children
        - add children
  - Does not remember visited States

Graph Search

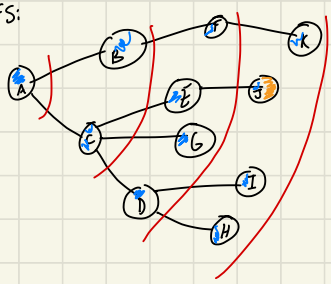
- once state expands, its neve expanded agaid. if State expands too early with sub optimal cost

1.) DFS



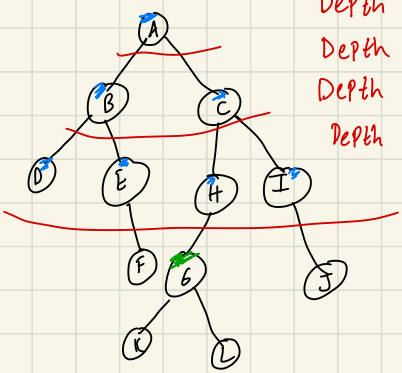
order of Nodes: {A, B, D, E, C, F, g, h}
Fringe: {B, c}, {D, E, C}, {F, G}, {h, i}

2) BFS:



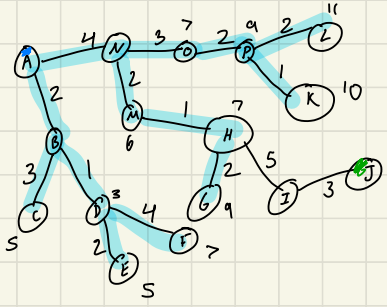
Nodes visited: {A, C, B, D, G, E, F, H, I, J}
Fringe Nodes: {C, B}, {D, G, E, B}, {H, I, G, E, B}, {J, B}

3.) Iterative Deepening



Depth Level: L1 : Nodes visited: {A} goal not found
Depth Level 2: Nodes visited: {A, B} goal not found
Depth Level 3: Nodes visited: {A, B, D, E, C, H, I} goal not found
Depth Level 4: Nodes visited: {A, B, D, E, F, C, H, G} goal found

4. Uniform Cost Search

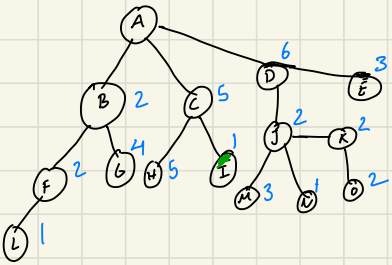


Expanding cheapest path order:
of the fringe, pick cheapest:

Nodes visited: {A, B, D, N, C, E, M, F, H, O, P, G, K, L, I, J}

- uses Cumulative Sum

5. Greedy Search



$f(n) = g(n)$, ignores Path

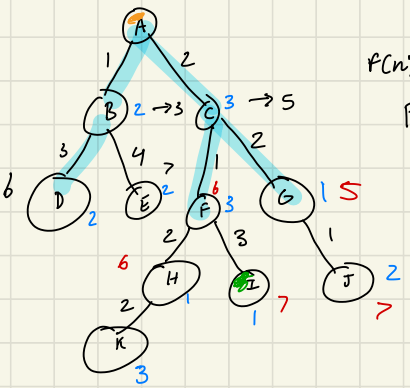
always chooses node that looks closest to a goal.

Visited nodes = {B, F, L, G, C, H, I}

Fringe after G expansion

{C, D}

6. A* Search



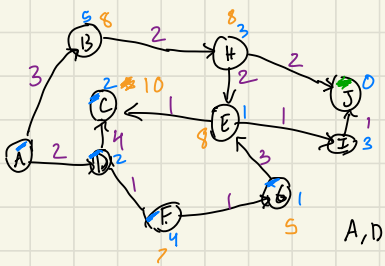
Nodes visited: {A, B, C, G, D, F, H, E, I}

$f(n) = g(n) + h(n)$

Fringe: {}

A*

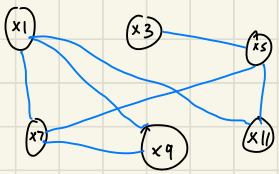
$f(n) = g(n) + h(n)$



A, D, F, G, B, E

Vars: { x1, x3, x5, x7, x9, x11 }

Cons: $x3 = x5$, $x5 \neq x7$, $x7 \neq x9$, $x9 \neq x1$, $x7 = 2 \times x1$, $x11 = x1$, $x5 \neq x11$



Arc Consistency:

1.) Empty queue, get all constraints
2.) take first constraint, pop it
3.) for every var in cons
    for every value in var.curdomain()
        if value has no support
            prune.append
            return false
        else:
            pass
    return true, prunings

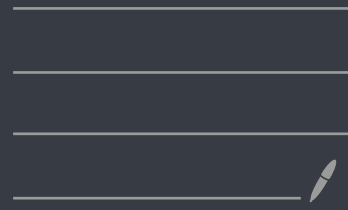
PropFC / GAC - use Heuristics MRV and DH

binary_negrid / nary_ad_grid / cagey_csp_model

- Transform Cagey → CSP
- binary_negrid
  - ↳ each row : all values diff
  - ↳ each col : all val diff
  - double for loop add constraint
- nary_ad_grid:
  - one n-ary all-diff constraint
  - get scope of vars in that row check tuples adding constraint

- cagey_csp_model (cagey_grid)
  - includes col/row all-diff constraint
    - ↳ cage constraints and special cases

Unit 2



Unit 2 - Planning

- adversial search : how agents make decisions in competitive games

Minimax Algorithm

- Zero sum games: agents have opposite utilities, one players gain is others loss.
- General games: independent utilities where coopelation is possible.
- Information and Certainty: games can be deterministic or stochastic (random)
  ↳ may offer perfect information (can see all state details)
- could involve one player, two, two +

To solve deterministic game with terminal utilities

- States (S) all possible configs
- Players (P): usually two taking terms
- Actions (A): may change based on state or player
- Transition function: (S x A → S)
- Terminal test (S → {t,f}): Determines if game has ended — is game over? → once game over, numerical outcome
- Terminal utilities (S x P → R): Final score or value assigned to players at end
- Solution: find policy (S → A) mapping every state to best possible action
  ↳ best move under worst case opponent response.

Minimax Algorithm

- Core method for solving adverserial games: minimax search
  ↳ operates on state space search tree, players alternate
- one player tries to reach state with highest possible utility, while opponent (min) tries to minimize.
- minimax value is best achievable utility against rational adversary ?
- recursive structure: value of state computed by looking at terminal states. max value function chooses from its childrens min results.

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successc
    return v
```

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Dispatch?

Efficiency and Pruning

- Optimal against perfect player, but computationally expensive
- Similiar complexity to DFS, $O(b^m)$, s.c of $O(bm)$, where b = branching factor, m = max depth
- game tree pruning to cut off search space
- Do few minimax examples

Alpha-Beta Pruning

- optimization of minimax algorithm, ignores branches that can't influence final decision
- Logic: holds 2 values: α (max best guaranteed value) and β (mins best guaranteed val)
  - ↳ if value makes it worse than previously explored option, prune it.
- Does not change result at the end but can reduce time to $O(b^{m/2})$
- AI uses depth limited search. ?
- Evaluation Function: because search does not reach terminal states.
  - ↳ score goodness of current board position
- Depth tradeoff: deeper search makes quality of evaluation less critical, can't see long term benefit.

Intro to Planning

- Planning differs from search using formal indep language (S.T.R.I.P.S)
- key Dimension: categorized by determinism (deterministic vs probabilistic)
  Observability (full vs partial) and multi-agent nature (collab vs adversarial)
- Moving basic formalism to complex real world applications.

Automated Planning

- STRIPS methodology: represent problem using tuple $<F, A, I, G>$
  - F: Fluents - set of variables representing, what can be true or false in world
  - A: Actions - set of maneuvers an agent can perform
  - I: Initial State - fluents that are true at start
  - G: Goal State: - specific fluents agent intends to make true

- State represents set of fluents currently true
  - → complete state: all other fluents false
  - → Partial state: does not matter if other fluents T/F

Action Definition

- a is defined by three components
  - → Preconditions: (PRE(a)): fluents that must hold for action to be executable
  - → Delete List: (DEL(a)): fluents removed from state after action
  - → Add List: (ADD(a)): fluents being added to state

Progression: moving frome one state to next

Resulting state is calculated by taking current state, removing delete list and joining it with add list: $Progress(s, a) = (s \setminus DEL(a)) \cup ADD(a)$.

PDDL (Planning Domain Definition Language)

- Standardized language used to specify random planning problems. Splitting tasks in two file
- Domain File: general world physics: requirements, types, predicates and actions
- Problem File: Defines specific instance, including objects involved, initial state and specific goal.

- Solving via Search: Planning usually uses A* which combines g(n) - cost to get to path and heuristic: h(n)

↳ Because of massive state spaces, modern solvers rely on domain-indep heuristics rather than problem specific ones

Advanced PDDL: Conditional Effects

- Conditional affects make action more dynamic
- Concept: Single action can have diff outcomes depending on current state of the world. Conditional on current state of the world.
- Syntax: implemented using (when <condition> <effect>)
- Instead of switch on switch off. Have 1 switch, which changes based on if light is currently on/off.

Search Strategies and Domain indep heuristics

- uniform cost search and greedy search $F(n) = g(n) + h(n)$
- Key Heuristics:

h^{add}: pessimistic heuristic that sums cost of achieving all individual goals. Often overestimates true cost and is inadmissible

h^{max}: optimistic heuristic, takes max cost required to reach any goal, underestimates the true cost, making it admissable

Real world Applications

- greenhouse optimization:
- Penetration testing
- Mega Printers

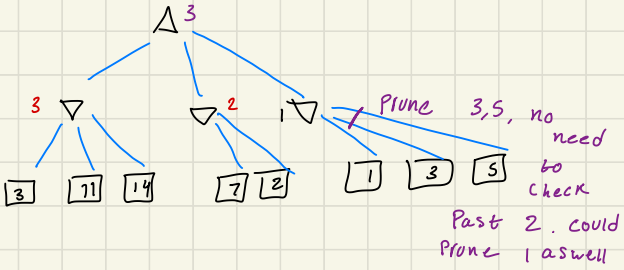
Quiz Prep

Minimax

- Successor of nodes children
- if state = terminal return utility
- if next agent is max / min decide

```
def max-value(state):
    v = -∞
    for each possible option
        v = max(v, value(child))
    return v
```

Minimax

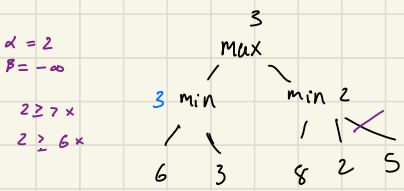


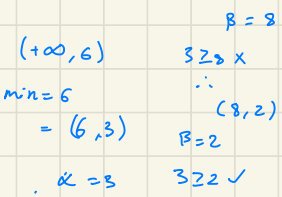
Alpha-Beta - skips branches that can't affect decision.

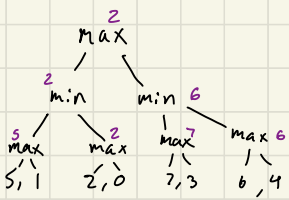
α - best value max can guarantee so far

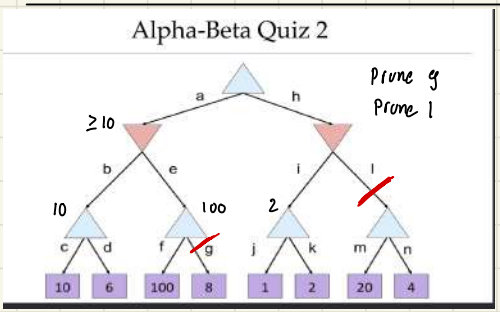
β - best value min can guarantee so far

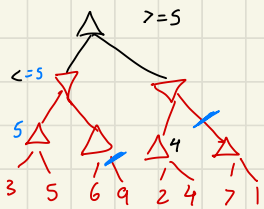
if α > β, prune











* Practise more later

Planning

- given a state, which action can be executed
- Apply actions to reach a new state
- is the plan valid to reach a goal state
- After actions a1, a2, a3, what holds true
- Conditional affects
- translate between PPDL → STRIPS

a = (Pre, add, del)

a1 = ({f1, f2}, {f3}, {f1}) . . . .

new state = (old state + add) − delete

Strips:

Pre: { have key, at_door }
add: { door_unlocked }
del { door_locked }

Quick States Exercise:

New State: (f1, , f5, f2)
(f1, f2, f5) → (f2, f3, f5)
(f2, f3, f5) → (f5, f4, f2) Final state goal achieved

PDDL:

```
(:action: opendoor
 :Precondition (and (have key) (at door))
 :effect (and door(unlocked and not(locked))
```

Domain: Physics of world
Problem: describes certain instance

Progression: applying actions and forward searching, until goal satisfied

Progress(s, a) = (s - DEL(a)) ∪ ADD(a)

Changes which actions become applicable

(:action a4
(: Pre condition
(: effects (and
(when(~ ~) and( )
conditional (not( ))

Conditional Effects

if the state looks like x, then do y

Pre conditions may happen if C1 → E1

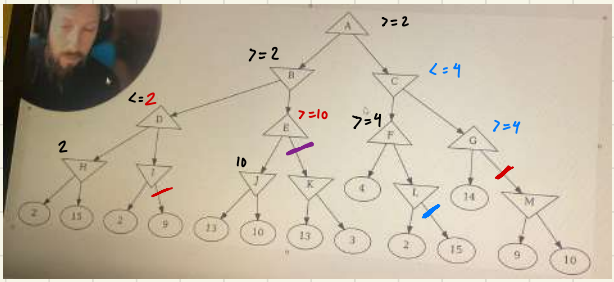
effects become: (Pre, eff)
↳ Set of conditional effect

a4 = (
{f2},
{ f3 → f4, → F3 is true, add f4
(f3 ∧ ¬f6) → f5, → if f3 and not f6 add f5
¬f6 } → regardless delete f6
)

Execut a4 if f2 holds

Q8) given english convert to STRIPS/PDDL

Q9) DO Reverse, convert PDDL → English



a1 : T
a2: F
a3: T

a1, a2

x1 is applicable

b, c

c, d yes valid

```
(:action register
 :parameters (s, c)
 :pre (and (open c) (not (registered s,c))
 )
 :effect
    (and (not open c) and (registered s, c)
 )
)
```

Practise Heuristic Tree + Alpha beta
+ examples w words

1.) o1, o2 are valid

2.) P, r, o2 fails i valid

3.) no valid plan

In state S, where $S = \{a, b, d\}$

we execute a4, so

$S = \{a, c, d\}$

$S_2 = \{a, c, e\}$

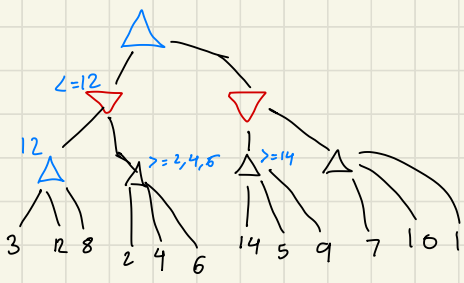
$S_3 = \{a\}$

w1 is applicable

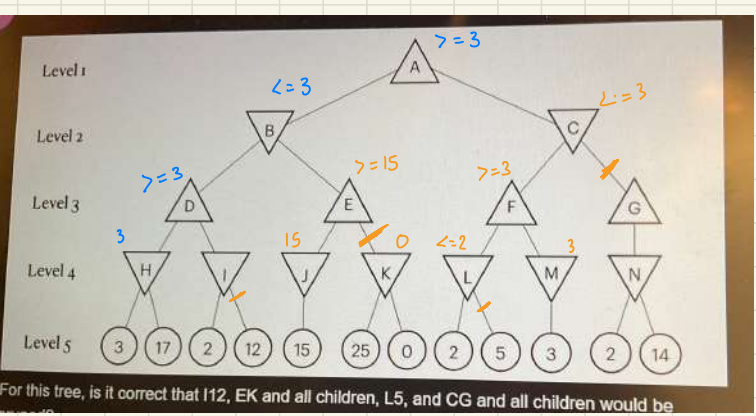
{ k, door open, }

{ door open, k, inside }

{ inside, k }

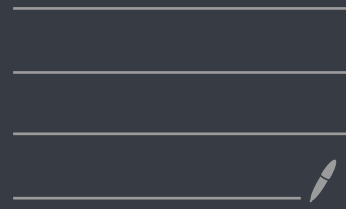


Review hmax/hadd Stuff
Alpha beta pruning



For this tree, is it correct that I12, EK and all children, L5, and CG and all children would be

Unit 3



Unit 3: Probabilistic Inference

- Managing uncertainty: agents have observed variables (things it knows) and unobserved variables (unknown environmental factors: location of ghost)
- Probabilistic reasoning is framework to guess what we don't know, with what we do know.
- Random Variables and Distributions:
  - ↳ represent aspects of world
  - ↳ table that lists every value a variable can take and assigns a probability.
- Joint Distribution: larger table, covers multiple variables, gives every possible combination of outcomes. All probabilities must sum to 1 (Probability of x and y at same time)
- Marginalization: Big table, many variables, but only care about 1, Collapse by adding probabilities for variables you want. (add rows that match what you want)

Conditional Probabilities

- Simple relation between joint and conditional probabilities.

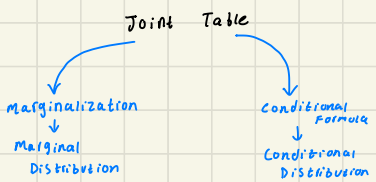
$$P(x|y) = \frac{P(x,y)}{P(y)}$$

Quiz practise:

$P(X|y) = \frac{0.2}{0.6} = 0.\overline{33}$

$P(-x|y) = \frac{0.4}{0.6} = 0.\overline{66}$

$P(-y|x) = \frac{0.3}{0.5} = 0.6$



Normalization Trick

- Normalization = rescale probabilities so they sum to 1.
- Instead of computing full formula: $P(A|B) = \frac{P(A,B)}{P(B)}$
  1) select rows with matching evidence
  2) ignore denominator
  3) normalize at the end

Reads like: whats the distribution of all possible values x can take, given evidence y.

Find $P(W|cold)$ = rows with cold, 0.2, 0.3

↳ normalize $\frac{0.2}{0.5}, \frac{0.3}{0.5}$

$0.4, 0.6$

$P(W|cold) = \frac{P(W,cold)}{P(cold)}$

↑ Avoids calculating

Distribution, Sums to 1

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

Probabilistic Inferencing

- inference is computing a conditional probability (what you want to know) based on evidence (what you already have)
- Evidence: variables you observed already
- query: variable you want to find
- goal: update beliefs based on evidence

Inference by Enumeration

- answering probability questions based on listing, summing rows from joint distribution table
- limitation: $O(d^n)$ for big tables
- sum all possibilities that produce that evidence
- $P(Q|E) = \frac{P(Q, E)}{P(E)}$ → can use normalized version or not

- $P(w = rain \mid winter) = 0.25$
  - ↳ normalize all $P(w \mid winter)$
  - $= \frac{0.25}{0.50} = 0.50$

Product Chain, Bayes Rule

- Product rule: $P(y) \cdot P(x|y) = P(x,y)$
  - ↳ useful when given conditional distribution, but want join

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

- Chain rule: break down complex situation into simpler conditionals

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

- Bayes Rule: Two ways to factor joint distribution over 2 variables: $P(x,y) = P(x|y) \cdot P(y) = P(y|x) \cdot P(x)$

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

Inference with Bayes' Rule

- Example: Diagnostic probability from causal probability:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}$$

- Example:
  - M: meningitis, S: stiff neck

$$\left.\begin{aligned} P(+m) &= 0.0001 \\ P(+s|+m) &= 0.8 \\ P(+s|-m) &= 0.01 \end{aligned}\right\} \text{Example givens}$$

$$P(+m|+s) = \frac{P(+s|+m)P(+m)}{P(+s)} = \frac{P(+s|+m)P(+m)}{P(+s|+m)P(+m) + P(+s|-m)P(-m)} = \frac{0.8 \times 0.0001}{0.8 \times 0.0001 + 0.01 \times 0.999} = 0.0079$$

→ all possible ways to get $P(+s)$

- Note: posterior probability of meningitis still very small
- Note: you should still get stiff necks checked out! Why?

Example

$P(w|dry) =$

$P(sun|dry) = P(dry|sun) \cdot P(sun)$
$= 0.9 \cdot 0.8$
$= 0.72$

$P(rain|dry) = P(dry|rain) \cdot P(rain)$
$= 0.3 \cdot 0.2$
$= 0.6$

$P(sun|dry) = 12/13$ → normalized
$P(rain|dry) = 1/13$ → normalized

Can calculate updated probabilities using causal probability

Independence

- Two variables don't affect each other
  - $P(x, y) = P(x) \cdot P(y)$
- Conditional Independence: Two events are only related because of a third event.
  - once you know third event, first two dont know anything new about each other.

$X \perp\!\!\!\perp Y \mid Z$ X is conditionally independent of Y given Z.

Bayes Net

- describes complex relationships using simple, local interactions.
- nodes are variables
- Arcs indicate a direct influence between variables
- $R \rightarrow T$, rain affects traffic

Represent Uncertainty
Instead of guessing
Compute probabilities

Conditional Probability Tables

→ if no parent → simple probability

→ if has parents → conditional prob

For independence

$R \rightarrow T \rightarrow L$

Late does not depend on rain, simplifying our calculation $P(L \mid T, R) = P(L \mid T)$

P(all variables) = Product of each node given its parents

To Build Bayes Net

1) Identify variables
2) Find causes, connect
3) Graph, Tables

Factor Zoo ?

- Probabilities treated as factors
-

P(+w) =

x and y are indep if $P(x) \cdot P(y) = P(x, y)$

x and y are conditionally indep if given z

is $x, y \mid z$ if

$P(a, b) = P(b \mid a) \cdot P(a)$

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

$P(a \mid b) = \frac{P(b \mid a) \cdot P(a)}{P(b)}$

P(+t) =

Causality vs Independence

- BN's represent causation
- Causal nets: simpler and easier for experts to build.
- True meaning: arrows represent conditional independence

Joining and Elimination

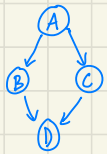
- **Joining** factors combine two smaller tables into a larger one by multiplying their entries together pointwise products
- **Eliminating** is "projection", where you shrink table by summing out specific var, you don't need

Variable Elimination

→ marginalize early

1) Pick hidden variable
2) Join small tables that mention relevant variable
3) Eliminate that var
4) repeat until only query, evidence remain

Ex



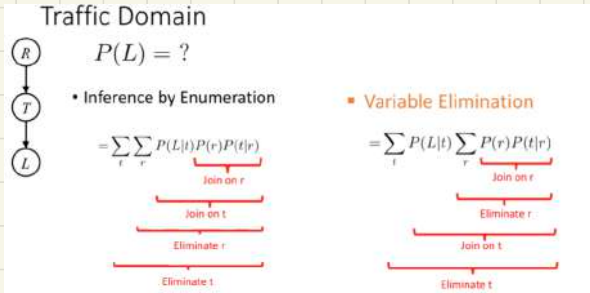
CPTs:

Factors

P(A) → f1 (A)
P(B|A) → f2 (A,B)
P(C|A) → f3 (A,C)
P(D|B,C) → f4 (B,C,D)

1) query: Find P(D)
must eliminate H.V A, B, C

2) Join factors containing A:
f5(A,B,C) = f1(A) · f2(A,B) · f3(A,C)
→ eliminate A, then repeat for B



* Ordering affects how large in between tables get
* Poly trees: Special bayes net, can always find efficient ordering

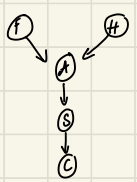
Factorization

Rain → Traffic → Accident
↓
Late

P(Rain, Accident, traffic, late) becomes...

= P(Rain)
× P(Accident)
× P(Traffic | Rain, Accident)
× P(Late | Traffic)

Conditional Indep
Says if I already know X, then Y does not tell me anything new.



F ⊥ H

F → A ← H, not observed blocked, indep, true

F ⊥ H | A, observed not blocked, not indep false

F ⊥ C | S
F → A → S → C, observed, blocked, indep, true

H ⊥ S | A, blocked, indep observe
H → A → S

D-Separation

- in bayes net find indep var based on graph structure
- Causal chains ($X \to Y \to Z$): if you observe Y, path is blocked and x and z become indep
- Common Cause ($X \leftarrow Y \to Z$): one cause has 2 effects. if you have Y, the effects become indep
- Common Effect/V-Structure ($X \to Z \leftarrow Y$): Two indep causes share one effect. Causes are indep unless observe effect. Observing effect "activates" path

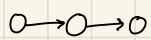
D-Seperation - Algorithm

- Are x and y independent given evidence $\{z\}$

1) Find all paths
2) check Active path (every triple)
3) if even one active path, indep is not guaranteed. if all paths are blocked, x and Y are guaranteed to be indep

- Complexity: $2^N \to O(N \times 2^{K+1})$

Active Triples



Path is active where $A \to B \to C$, where B is unobserved
Common cause $A \leftarrow B \to C$, where B is unobserved
Common effect: $A \to B \leftarrow C$, where B is observed.

Note: if atleast one path open, not indep

Ex

1) $L \perp\!\!\!\perp T' \mid T$

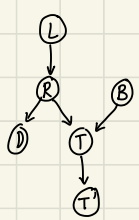
$L \to R \to T \to T'$
T is observed, causal blocked
- indep

2) $L \perp\!\!\!\perp B$

$L \to R \to T \leftarrow B$
-V affect, not observed, blocked
- indep

3) $L \perp\!\!\!\perp B \mid T$

$L \to R \to T \leftarrow B$
-V affect, observed, valid
→ not indep

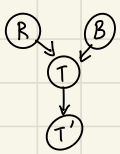


4) $L \perp\!\!\!\perp B \mid T'$
- $L \to R \to T \leftarrow B$, $T \to T'$
V affect opens path
- not indep

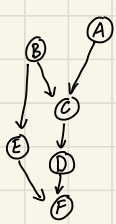
5) $L \perp\!\!\!\perp B \mid T, R$
given both, blocked, indep

Ex

$R \perp\!\!\!\perp B$
$R \not\perp\!\!\!\perp B \mid T$
$R \not\perp\!\!\!\perp B \mid T'$



1) $R \perp B$ has $R \to T \leftarrow B$, consider no T, so not observed, independent
2) $R \not\perp\!\!\!\perp B \mid T$, same thing given T, so not indep.
3) $R \not\perp\!\!\!\perp B \mid T'$
$R \to T \to T'$
$B \to T \to T'$
→ causal and we have T' ∴ blocked so not indep



$A \to C \leftarrow B$

$A \to C \leftarrow B \to E$

Naive Bayes'

$P(Y)$ = how often spam occurs
$P(word|Y)$ = how often word appears in spam

→ Predict a class Y, from features
→ All features are independent given class

$P(w_1, w_2, w_3 | spam) = P(w_1|spam) \cdot P(w_2|spam) \cdot P(w_3|spam)$

easy to compute and uses Score (class) = P (class) · product of word probabilities.

Ex: give 2 words in spam

$P(spam) = 0.33$
$P(w_1|spam) = 0.0156$
$P(w_2|spam) = 0.0153$

Spam Score = $0.33 \times 0.0156 \times 0.0153$

Ham Score : $0.66 \times P(w_1|ham) \cdot P(w_2|ham)$

Compare Probabilities, bigger Score = predicted label

$P(positive) = 0.55 \cdot 0.08 \cdot 0.05 \cdot 0.02 = \frac{11}{250\,000}$
$P(neg) = 0.45 \cdot 0.01 \cdot 0.03 \cdot 0.07 = 9.45 \times 10^{-6}$

Laplace Smoothing

- when estimating probabilities if word is not in relevant group. Probability is 0, but instead of x 0, add K fake observations to all category.
- Count (x)/N
  ↳ (Count(x) + K) / (N + K x number_of_categories)
  → add K to each count, increase total accordingly.

Ex Data: {r, r, b, g, b, b, g, r, b, g, r}
count : r=4, b=4, g=3. N=11
so $|x| = 3$, from 3 possible values

$P(r) = 4/11$
$P(b) = 4/11$
$P(g) = 3/11$

use K=1, now →

$P(r) = 5/14$
$P(b) = 5/14$
$P(g) = 4/14$

Now $Y = 0$, gives $P(y) = \frac{1}{(N + |x|)}$

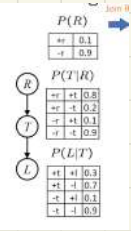
red = 4/9 → 6/19
blue = 3/9 → 5/19
green = 1/9 → 3/19
yellow = 1/9 → 3/19
Purple = 0 → 2/19

n=9, K=2, |x| = 5

$(N + K|x|)$ → denominator

$\frac{K}{(N + K|x|)}$

Variable Elimination with Bayes Net



Trying to Find P(L)

Must remove hidden variables

1) Join R Tables, multiply matching rows

+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

2) Sum out R

+t	0.17
-t	0.83

3) Join with L

+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

4) Eliminate T

+l	0.134
-l	0.866

Join A and B

+A	+B	0.028
+A	-B	0.012
-A	+B	0.12
-A	-B	0.056

Eliminate A

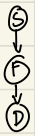
+B	0.3
-B	0.7

Join C

+B	+C	0.18
+B	-C	0.12
-B	+C	0.07
-B	-C	0.72

Sum C

+C	0.25
-C	0.9



1) Join S, F

P(S,F)

S	f	0.06
S	-f	0.24
-S	f	0.42
-S	-f	0.28

2) Sum out S

P(f)

+f	0.48
-f	0.52

3) Combine with f, D

P(f, d)

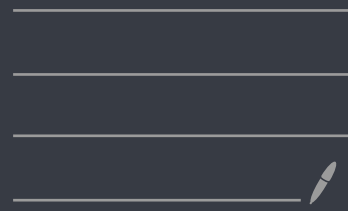
+f	+d	0.336
+f	-d	0.144
-f	+d	0.052
-f	-d	0.468

4) Sum out f

P(d)

+d	0.388
-d	0.612

Unit 4



Unit 4: Deep Learning

Decision Tree

- model that uses series of binary or multiway questions to classify data. 3 main node types
- Root node
- Internal nodes - test a specific attribute "question nodes" 2 or more branches
- Leaf Nodes: - final answers, that give class label

How to construct

- induction uses greedy strategy: making best descion at current split
- Hunts Algorithm: recursive procedure to split data into smaller subsets, until all records in subset belong to same class
- ID3: Creating shallowest tree by using information gain to select best feature to spit on each step

Choosing split

- to choose which algorithm makes best split, measure impurity - how mixed classes are in a group
- Entropy: the uncertainty or pureness of set. 1 class → 0 entropy
- Information gain: reduction in entropy is achieved by splitting specific feature
- Gini Index and Misclassification error

Pros and Cons

- Easy interpretation
- Efficiency
- Robust
- Lower Accuracy

The Neuron

- Perceptron (neuron) Flow:
- Inputs and weights: recieves feature values as input and assigns weight to determine importance
- Summation and Activation: Sum input to create activation value.
- Output: passed through activation function to produce classification label

Deep Neural Networks

- deep learning involves stacking many neurons
- Learning features: learns themselves from raw data
- Power of depth: more layers → better generalization and higher accuracy ?
- Universal Approximators: network with even 2 layers can approximate any continous function

Training Network

- finding best weights to maximize the probability of a correct predictions

Training Loop:

- Forward pass → make prediction
- Compare to true answer → Compute loss
- Backpropagation → Compute gradients
- Update weights → gradient descent

While training may feel automated, developer chooses:

- number of layers (more layers, more complex)
- width (neurons per layer)
- Learning rate (most important)
- Epoch, batch size, regularization

Training Techniques

- gradient ascent
- Backprop: use chain rule to calculate derivatives by passing information backward
- Early stopping to prevent overfitting

Real-world challenges

- Bias: toxic, racist, discriminatory content
- Correlation vs Causation
- Robustness and Manipulation (apple labelled "Ipod")

Activation Functions

- nonlinear functions
- Sigmoid and tanh
- ReLu (max(0, x))
- Leaky ReLu

Loss Functions

- Score that explains how good/bad prediction was (Try to minimize)
- Classification:
  - ↳ Cross entropy: reward confident correct answers, punish confident wrong answers hard
  - ↳ Negative log-likelihood: take probability of correct answer, punish low prob, scales nicely
- Regression:
  - ↳ L1: Mean Absolute error (prediction - actual)
  - L2: Mean Squared error $(\text{prediction} - \text{actual})^2$

Regularization - prevent overfitting

- reducing generalization by adding L1/L2: penalties to avoid complexity
- Data Augmentation: Create synthetic training data by flipping/turning/color shift in images
- Dropout: randomly turn off neurons

Backpropagation

- figure out where error comes from and fix it

1.) make prediction
2.) compute loss
3.) send error backword through network
4.) Adjust weights reduce error

CNN's

- Process grid like data, same weights across different parts of input, much more efficien-
- replaces matrix multiplication with convolution, slides over input to create feature map
- Filter is group of weights, model learns these, tells us what patterns to look for
- Stride: how many pixels you jump each time, (low stride = walking, big stride = running)
- Padding: adding border to keep output consistent, and not to lose edges
- Pooling: Summarizing small regions

RNN's

- Used for sequential data, order matters. Reads one piece at time, combines memory, update and pass forward.
- loops: allow output to get fed back in as input
- unfolding: multiple copies of same network, each passing message to successor
- Configuration: many to one, one to many, many to many

Long-Short Term Memory

- RNN's struggle to remember information because of vanishing gradient. (longer it gets, harder to remember)
- Gating is a smart technique with a forget gate, input gate and output gate.
- Real world Applications: handwriting and action recognition, medical and robotics

Autoencoders - type of neural network

- models complex data by learning how to compress and then reconstruct it.
- The encoder takes input, shrinks it into (latent space representation).
- The decoder takes that code and tries to reconstruct the original as accurately as possible
- Model is trained by minimizing difference between original and reconstructed (MSE)

Variational Autoencoders (VAE's)

- instead of single fixed point, the encoder predicts probability distribution
  ↳ gives a whole range of possible codes (more info)
- Gumbel-softmax: turns discrete categorical data int soft smooth approximation

LatPlan

- learn how system works by watching it, then solve
- 1) Autoencodes turns raw images into simple binary code
- 2) Learns rules using PDDL
- 3) Planner finds solution in latent space, decodes back into sequence images

Transformers

- look at everything at once, uses Attention
- Self-Attention: model looks at every word simultaneously to see most relevant
- Bert: looks in both direction
- Gpt: only reads from left → Right. (can use zero-shot, few shot)

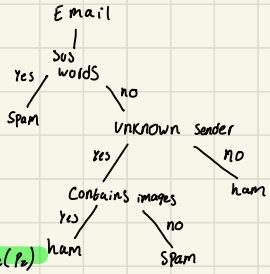
Possible types of questions

1) given Descion tree, classify test data (trivial likely will not show up)

2) given some data, figure out best splits for tree

Ex Each row is email: Predict ham/spam

Features: Suspicious words, Unknown Sender, Contains images



Hunts Algorithm: If "pure" stop, else pick feature, split, repeat.

ID3: Split at feature with most information gained.

Entropy: Calculates how mixed a dataset is. Entropy = $-p_1 \log_2(p_1) - p_2 \log_2(p_2)$

Compute entropy for a dataset (cleaner split = higher gain)

→ if $\frac{4}{10}$ spam, $\frac{6}{10}$ ham Entropy = $-0.4 \log_2(0.4) - 0.6 \log_2(0.6)$

ID3 Method

1) Find entropy of full dataset

2) try each feature calculate entropy if 2 spam, 1 ham $H(\text{feature } A) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}$

↳ calculates for both true/false. $H_{After} = \frac{\#\text{ of T}}{\text{datasize}}(\text{entropy}) + \frac{\#\text{ of F}}{\text{datasize}}(\text{entropy})$

3) Information gain for that feature is data entropy − H_{after}

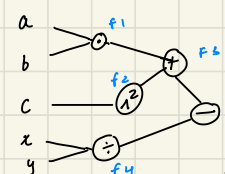
Full entropy = $-0.5\log 0.5 - 0.5\log 0.5 = 1$

Feature A = T $-3/4 \log\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4}$ → $\frac{3}{8}(0.\ ) +$

Feature B =

Then choose feature with most information gained

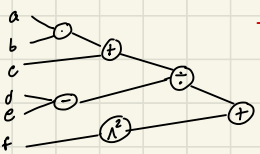
4)



$f_1 = a \cdot b$
$f_2 = c^2$
$f_3 = f_1 + f_2$
$f_4 = x/y$
$z = f_3 - f_4$

given variables, just plug and play to compute forward pass

Ex $z = \frac{(a \cdot b + c)}{(d - e)} + f^2$



given
$a = 5$
$b = 4$
$c = 3$
$x = 2$
$y = 1$
$z = 27$

5) Compute the value z, step by step compute all intermediate terms

6) Calculate gradients (back pass for same model) How does changing each input affect z

$f_1 = a \cdot b$
$f_2 = c^2$
$f_3 = f_1 + f_2$
$f_4 = x/y$
$z = f_3 - f_4$

Start at end

1) $\frac{\partial z}{\partial z} = 1$

2) from $z = f_3 - f_4$
$\frac{\partial z}{\partial f_3} = 1$ $\frac{\partial z}{\partial f_4} = -1$

3 from $f_3 = f_1 + f_2$
$\frac{\partial z}{\partial f_1} = 1 \cdot 1 = 1$ $\frac{\partial z}{\partial f_2} = 1 \cdot 1 = 1$

4 from $f_1 = a \cdot b$
$\frac{\partial z}{\partial f_1} = 1 \cdot b = 4$ $\frac{\partial z}{\partial b} = 1 \cdot a = 5$

6 from $f_2 = c^2$
$\frac{\partial z}{\partial c} = 1 \cdot 2c = 6$

6 from $f_4 = x/y$
$\frac{\partial z}{\partial x} = -1 \cdot (1/y) = -1$ $\frac{\partial z}{\partial y} = -1 \cdot (-x/y^2) = +2$

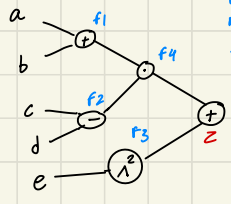
$\frac{\partial z}{\partial c} = \frac{\partial z}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial c}$

Chain rule: $\frac{\partial z}{\partial a} = \frac{\partial z}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_1} \cdot \frac{\partial f_1}{\partial a}$

$\frac{\partial z}{\partial b} = \frac{\partial z}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_1} \cdot \frac{\partial f_1}{\partial b}$

$= 1 \cdot 1 \cdot a$

$= a$



$f1 = a+b$
$f2 = c-d$
$f3 = e^2$
$f4 = f1 \cdot f2$
$z = f4 + f3$

$z = 36$

$\frac{\partial z}{\partial a} = \frac{\partial z}{\partial f4} \cdot \frac{\partial f4}{\partial f1} \cdot \frac{\partial f1}{\partial a}$

$= 1 \cdot f2 \cdot 1$

$= 4$

$\frac{\partial z}{\partial c} = \frac{\partial z}{\partial f4} \cdot \frac{\partial f4}{\partial f2} \cdot \frac{\partial f2}{\partial c}$

$= 1 \cdot df_1 \cdot 1$

$= 5$

$\frac{\partial z}{\partial e} = \frac{\partial z}{\partial f3} \cdot \frac{\partial f3}{\partial e}$

$= 1 \cdot 2e$

$= 8$

7.) Slide small matrix filter, over big matrix input, multiply and sum

Input: 4 by 5 matrix
Filter: 2 by 2 filter
Stride = 2 → skip 1 cell each move
1 → move 1 step
Padding = 0, no border
Padding = 1, zeroes around

output height $= \left[\frac{H - F_H + 2P}{S}\right] + 1$

output width $= \left[\frac{W - F_W + 2P}{S}\right] + 1$

(round down)

$\left(\frac{5-2+0}{2}\right] + 1 = \left[\frac{3}{2}\right] + 1 = 2$

$\left[\frac{4-2+0}{2}\right] + 1 = \left[\frac{2}{2}\right] + 1 = 1+1 = 2$

output height x output width

Ex

$input = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 2 \\ 1 & 3 & 2 & 0 \end{bmatrix}$

$filter = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$

$output = \begin{bmatrix} 14 & 16 \\ 12 & 13 \end{bmatrix}$

$output = \begin{bmatrix} 6 & 8 \\ 12 & 14 \end{bmatrix}$

$output = \begin{bmatrix} 10 & 16 \\ 7 & 10 \end{bmatrix}$

$output = \begin{bmatrix} 6 & 12 & 8 \\ 14 & 25 & 16 \\ 12 & 21 & 14 \end{bmatrix}$

$Input = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 \\ 0 & 5 & 4 & 3 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ 5 by 7

$filter = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$

$Output = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 \\ & & & & \\ 1 & 2 & 2 & 2 & 1 \end{bmatrix}$

- CNN share weights, focus on small regions, allows pattern learning, sliding across image, receptive field is how much of image neuron sees.

8. Apply pooling function (max / sum) and dimensions, stride padding (Does not take filter matrix)

- Applies simple functions, outputs one number
- given: pool size, function, stride, padding

$\begin{bmatrix} 4 & 4 & 4 & 2 \\ 5 & 4 & 5 & 1 \\ 1 & 3 & 2 & 2 \\ 5 & 3 & 5 & 4 \\ 5 & 1 & 2 & 4 \end{bmatrix}$

$\begin{bmatrix} x & x \\ x & x \\ x & x \end{bmatrix}$

$\begin{bmatrix} 7 & 8 \\ 9 & 8 \end{bmatrix}$

$\begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$

3) run simple train process

Objective: correct class score > all other class score

Score = $wc \cdot x$

goal: update weight vectors, so each sentance gets classified properly

1) Compute scores
2) Predict class
3) Compare with true label
4) if correct, nothing
5) if wrong update

comeback to

"win the vote"

Sports score = 1 $[1\ 0\ 0\ 0\ 0] - [1\ 1\ 0\ 1\ 1]$

For 1 = $[0\ -1\ 0\ -1\ -1]$

For politics 0 = $[1\ 1\ 0\ 1\ 1]$

"win election"

For Sports 1 = $[0\ -1\ 0\ -1\ -1] \cdot [1\ 1\ 0\ 0\ 1]$

$= -2$

For politics = $[1\ 1\ 0\ 1\ 1] \cdot [1\ 1\ 0\ 0\ 1]$

$= 3$

Entropy for whole set = 1

Feature A: T: 4W, 1 Loss

$-(0.8 \log_2 0.8) - (0.2 \log_2 0.8)$

Entropy = 0.72

F: 4 2 1W

= 0.72

$= 0.5(0.72) + 0.5(0.72)$

$= 1 - 0.72 = 0.28$

Feature B: 6 True: 4W 2L

T: $-(-0.390) - (-0.528)$

$= 0.918$

F: = 4: 1W 3L

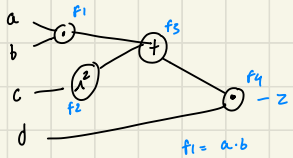
$-(-0.481) - (-0.3)$

$= 0.781$

$= (0.918)(0.6) + (0.4)(0.781)$

$= 0.8632$

$= 1 - 0.8632 = 0.16$



$z = 110$

$f1 = a \cdot b$

$f2 = c^2$

$f3 = f1 + f2$

$f4 = d$

$z = f3 \cdot f4$

$\frac{\partial z}{\partial a} = \frac{\partial z}{\partial f4} \cdot \frac{\partial f4}{\partial f3} \cdot \frac{\partial f3}{\partial f1} \cdot \frac{df1}{\partial fa}$

$= 1 \cdot$

$\frac{\partial z}{\partial f3} = f4 = 5$, $\frac{\partial z}{\partial f4} = f3 = 22$

$\frac{\partial z}{\partial f1} = \frac{\partial z}{f3} \cdot \frac{\partial f3}{\partial f1}$

$= 5 \cdot 1$

$= 5$

$\frac{\partial z}{\partial c} = \frac{\partial z}{\partial f3} \cdot \frac{\partial f3}{\partial f2} \cdot \frac{\partial f2}{\partial c}$

$= 5 \cdot 1 \cdot 2 \cdot 4$

$= 40$

Neural Nets Conceptually

Batching - sending in multiple inputs at once {5, 10, 15, 20}, update weights after every batch

→ Faster computation, smooth learning

→ Large batch = overfitting, memory limits

Feature expansion 1 sample → 10 features → 50 features (every layer gets more features)

network makes features by doing new feature = activation(old feature × weight + bias)

CNN's number of features = number of filters

After every batch, compute loss, update weights, features get smarter