07/01/2019

CSCD84 - AI

What is AI
↳ A computer program that behaves as if it was smart

Real world Applications
↳ finding patterns in data ie diagnoses system
↳ weather prediction
↳ robotic exploration
↳ computer vision
↳ self driving cars

Topics
→ Search
- Constraint satisfaction
→ Graphic models
→ inference
→ logic
→ Game trees (i.e. Chess, Go, etc)
→ Decision processes
→ neural network

AI-applications
↳ Planning - path finding ⇐ Google maps
↳ Computational Linguistics → Language understanding
Deep network ↓ Speech recognition → language understanding ⇐ Watson

I caught a fish, but couldn't put it in a basket because it was too big / small (Check for ~~Examples~~ to clarify)
↑ who
↳ Image / media applications ↳ automated annotation
↳ Recommendations → Amazon → Netflix
↳ Alpha Go (game playing)
↳ Prediction → stocks → weather ↳ security

Intelligent Agent

↳ "Something" that has a goal to fulfill
expressed as a utility function

↳ SimpleA
reactive agent
i.e. Smoke detector

CSC1084 - AI

Intelligent Agents → A program that has a goal to fulfill
System

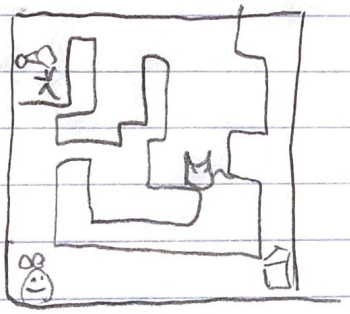
reactive! ← leave behind

utility fn

→ context
→ reactive is too limited
→ smart ⇒ planning ahead
PLAN

→ how to encode information the agents needs

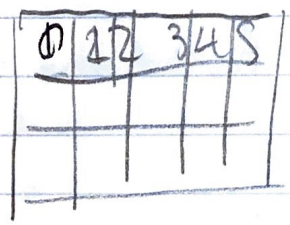
Maze

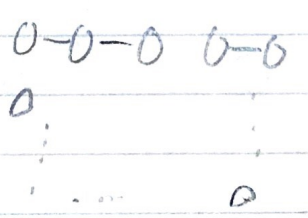


Agents - Cat (3)
↳ maze
→ environment → cheese (1)
↳ maze walls

→ actions ⇝ take turns
↳ move up, dn, left, right

Representation: Approach Grid





good but incomplete!
↳ no representation for agents

how about, Maze is stored in a grid Someone Agents can look at

Agents needs

Configuration
- state of environment at current time
  - (x_1, y_1) position of agent
  - (x_2, y_2) positions of other agents
  - → walls ?
  - → cheese?

set of configurations
↳ state space

Example
- → 1 mouse
- → 2 cats
- 1 cheese
- maze ↳ 4x4 grid cells

cats no overlap
- mouse 10
- Cat 16
- cat 15
- Cheese 14

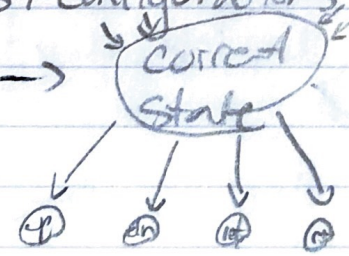
↳ increase agents ↑ cats make cheese $\geq 16^{(4)} = 65\,536$

What if 8x8 grid ≥ 64^4

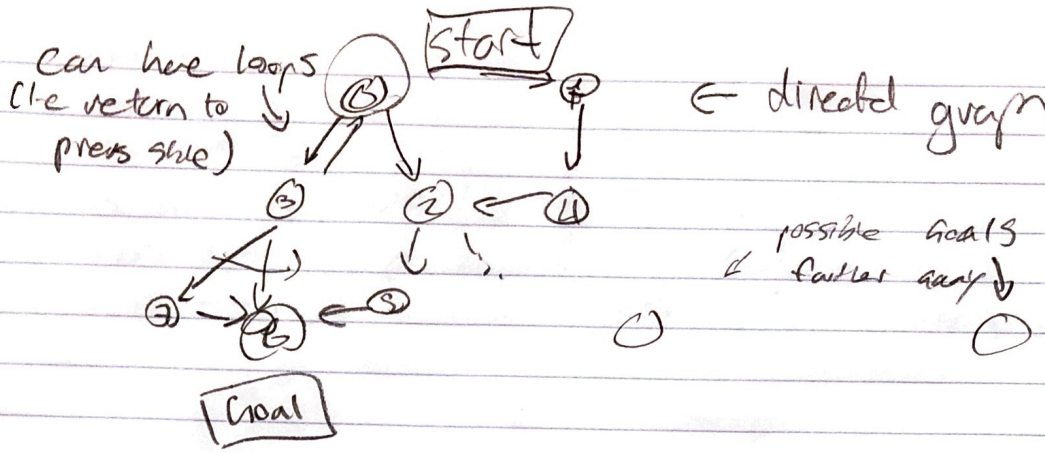
Solving problems by **Search**
↑ exploring state space

represent states / configurations as a graph ← predecessor states

Specific config → current state → next states → depends on possible actions by agent



bounded above by 4, because walls



← directed graph

* requires finding a path

Planning as Graph search
↳ explanandum (strategy)
↳ BFS - breadth first search
DFS - depth first search

BFS
→ expand a node → check it is a solution
↑ (check if goal!)
↓ not a solution
add neighbours to list of nodes to expand (exp)
↳ queue: 3 2 1 7 6 5 4
↑ possible successor states
expand 3 ✗ expand

CSC 384 - A7

I, A_0, $\sim A$, T_0

↳ Goal to fulfill

↳ utility

Agent

Configuration

values of all variables of interest

(x_{mo}, y_{mo})

1 mouse

n Cheese

m cats

Have (x,y)

3 Cheese

2 cats

6 things (x,y) = 12 values

" variables

K

maze

K

Mouse initial x, y

cats " "

Cheese " "

State Config

Mouse

Cats

successor states

Plan → graph search

action 1

action 2

action 3

action 4

goal

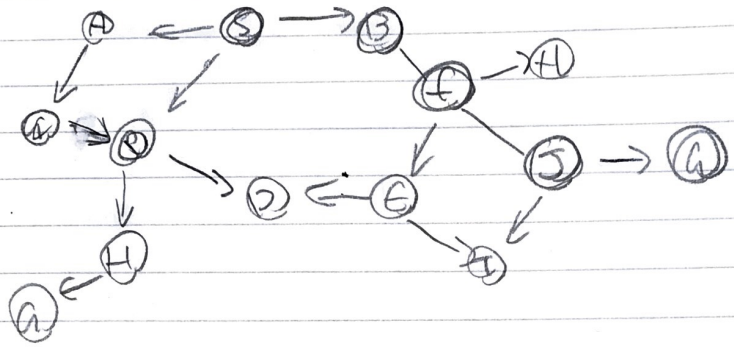
fulfill

Goal

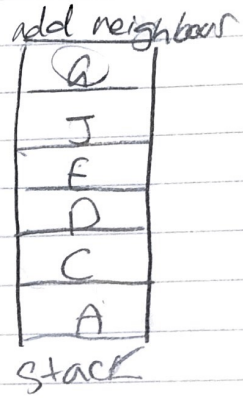
#traversals are actions

- BFS → closest goal
  - → in order from start, closest nodes first
  - → keeps a queue of nodes to expand

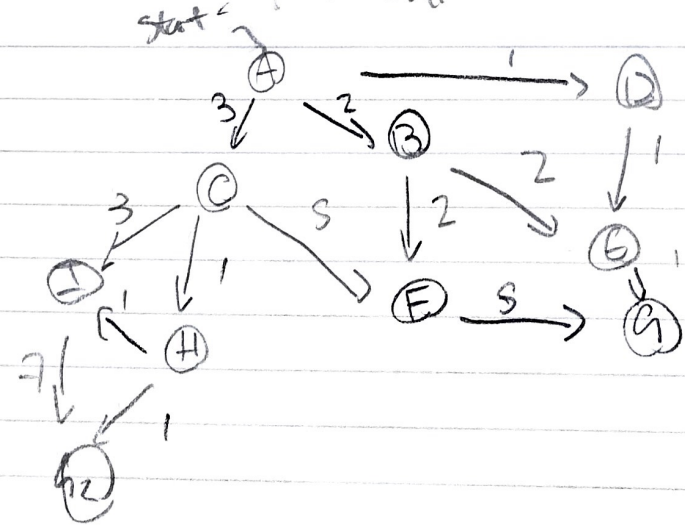
DFS



add neighbours



One major assumption / problem
- → no edge wts
- → reqs build edge weights into search
  - ↓ Uniform cost search (UCS)



expand first nodes w smaller cost

	node	cost	prev
expnd →	A	0	—
	B	2	A
	C	3	A
→	D	1	A
	E	2	D
	F	4	B
	G1	3	E
	G2	5	
	H	4	C
	J	6	C

Bryan Chen
UH 4-Sp ICAC 2

Q1

Queue Container : a b c e g d f i m j p k h l o q r s
expanded : a b c e g d f i m j p k h l o r
path: a → b → e → m → r

Q2 r

Q3 stack: a c b d g l p j m r
expanded: a b e m r
Path: a → b → e → m → r

Q4 r

Q5

node	cost	pred
a	0	—
b	2	a
e	1	a
d	33	b
e	3	b
f	4	c
g	5	b
h	5	d
i	2	c
j	6	e
k		
l		
m	6	e
n		
o	3	i
p	7	h

node	cost	pred
q	6	i
r		
s		
t		

Q6) a, c, e, s, t, u
u is optimal

CSC1584 - AI

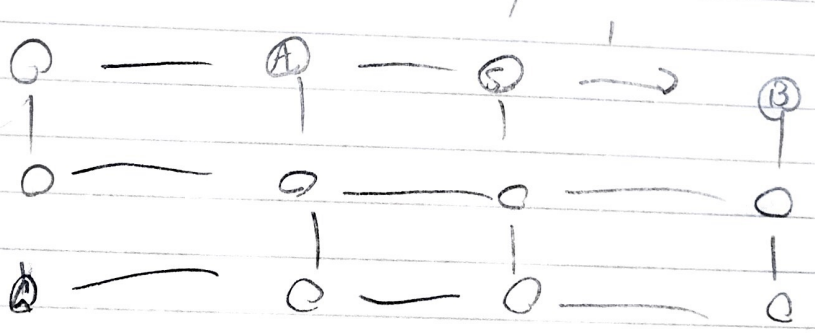
Review

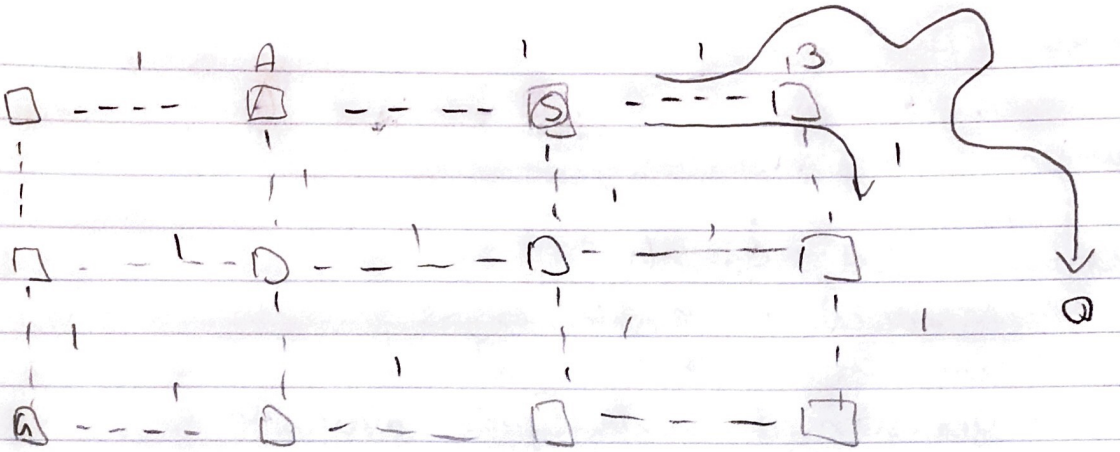
Uniform Cost Search
- introduced weighted edges
- want optimal path to goal
- variation of shortest path algorithm

Algorithm
- expands nodes in order of smallest cost from start

Limitations
it does a lot of work that unnecessarily going to pay off, because it doesn't use any knowledge of the problem or the graph

ex every edge has wt 1
taking the path w the arrow will make us go further away





- Heuristic Search ← h(n) ← heuristic value of node(n)
  - ↳ heuristic function → guess! (but a good one)
    - ↓
    - estimates how close a node is to goal
  - → Admissable: $h(n) \leq h^*(n)$

$h^*(n)$ - actual cost of getting to goal from n
tree cost

- one heuristic always admissable
  - $h(n) \neq 0$

Using heuristic function

1. Best first Search (BeFS)
   - expand in order of smaller h(n)
   - flaw: Doesn't account how we are going to get to the node with the best
   - $h(n) \approx \inf$

A* Search

- for every node in the list of nodes to be explored keep track of

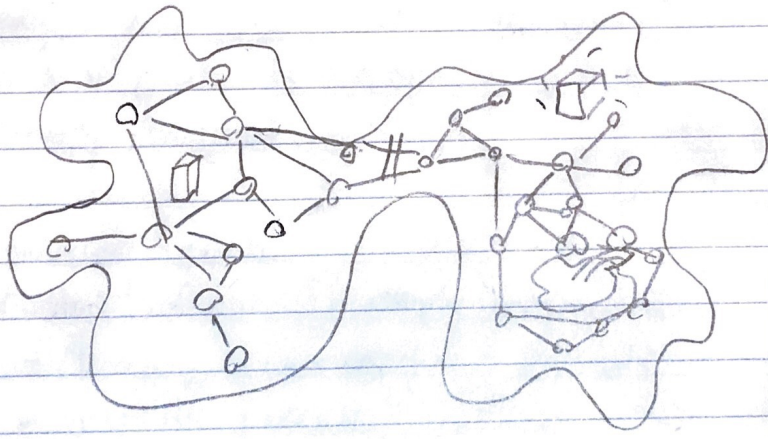
$$f(n) = g(n) + h(n)$$

$g(n)$ - cost to get from start to n

$h(n)$ - estimated cost to reach goal from n

A* expands in order of smallest $f(n)$ first

Paco's Park Level



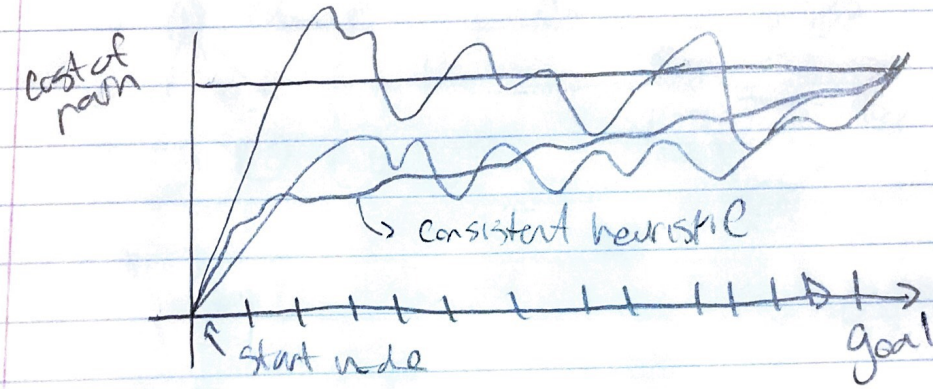
Warning!

compiler is unhappy with -fpic

- mount your own home dir on mathlab as a local dir in your machine

sshfs

→ compile on mathlab



cost of optimal path

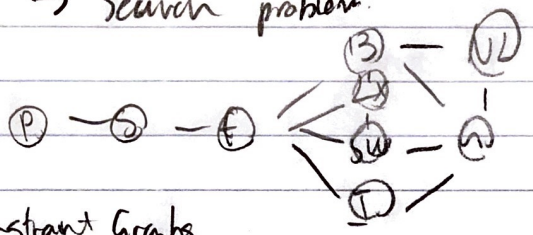
21/01/19

CSC1084 - AI

Constraint Satisfaction Problem

→ scheduling
limited resource, like rooms, instructors, ppl with other schedule assignments

→ finite # of vars → Generally n vars, d vals ⇒ n^d assignments
→ vars have discrete vals
→ Search problem!



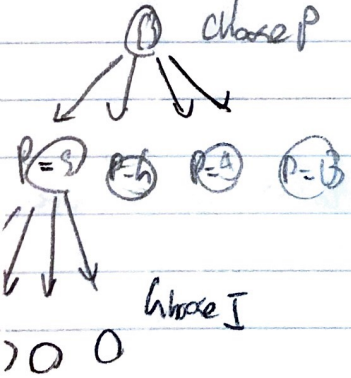
My colouring problem
colour a country s.t. no 2, neighbours, have same colour constraint. Make a path → neighbours have edge. nodes = var, edge = constraint.

Constraint Graphs

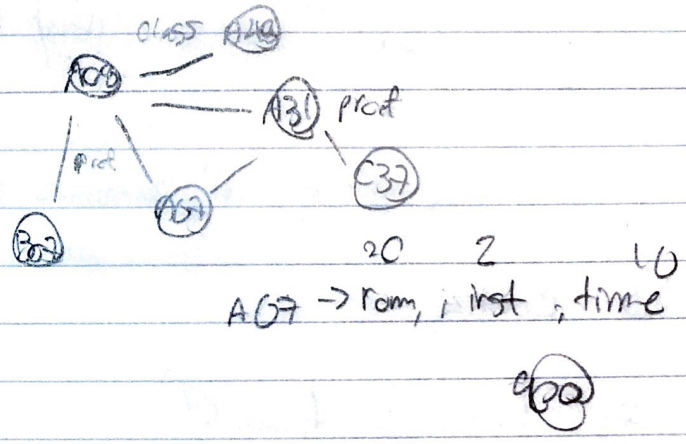
- nodes are variables
- edges are constraints btw variables
- unary constraints - specific vals for certain vars
- binary constraints

Backtracking Search

- start off w empty assignment
- Choose an unassigned variable
- try its possible values



ex



ck track

parent Nae a a c c c i i o s t
Expansion: a c i o s t u

Q11 yes because all uts are uniform

Q12 no, manhattan distance

Q13 - don't expect perfect heuristic
- but need to identify important factors that affect heuristic

Ex 2 - CSP

Q1 Countries: nodes
borders: edges

Q2 Countries (edges are constraints, but no adj countries can be same colour)

Q3 Colours

Q4 Germany (7)

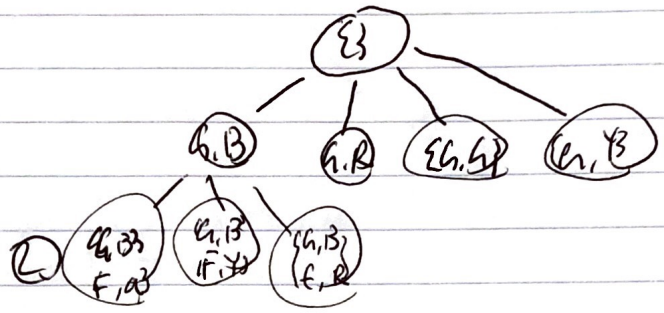
Q5 portugal

① Choose variable with fewest remaining valid values

② if ① ties, then choose variable w most active constraints

③ Select value that puts fewest constraints on remaining variables (most values left)

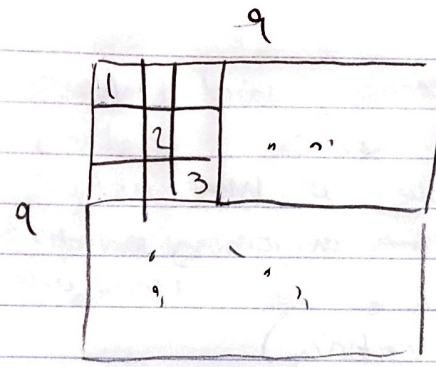
Q6 France / Germany (based on 2nd criteria)



Q7

Q8 81 - blocks

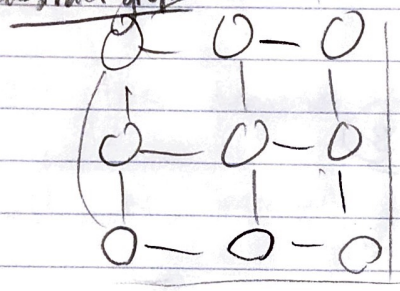
Q9 q^{81}



ex Sodoku · no repeats on the column sector

81 vars
9 vals

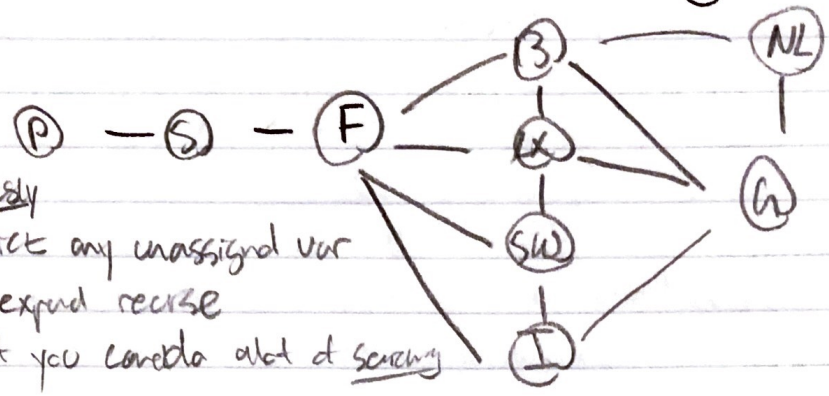
Constraint graph



23/01/2019

CSC084-A1

We talked about backtracking search



previously
- pick any unassigned var
- expand recurse

but you can do alot of searching

Want to pick unassigned var and its values carefully can help speed stuff up.

Rules to pick correct ones!

1st rule: Pick variable w fewest values left to try

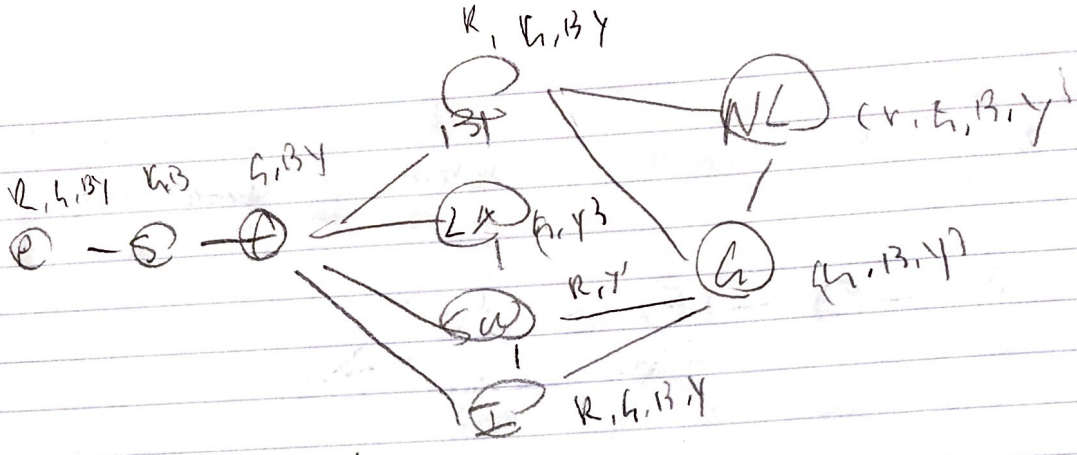
2nd rule: if there's a tie for Rule1, choose the variable w most active constraints*

→ eliminates more choices on neighbours, reduces search space

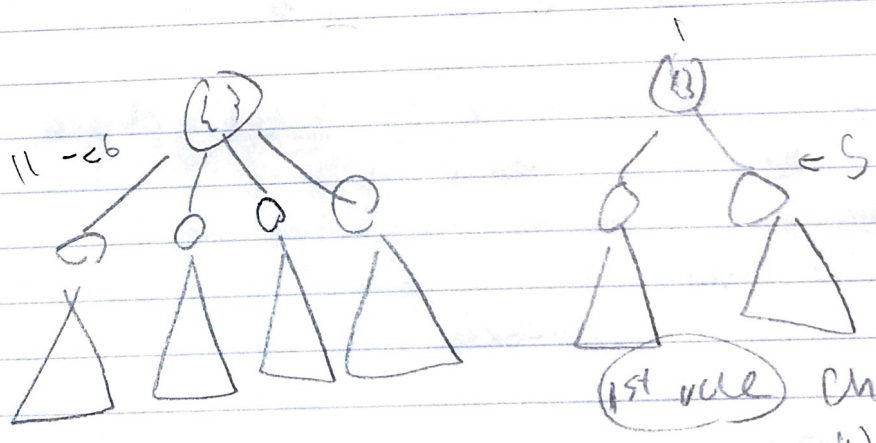
→ active constraint: if either of the neighbours of a constraint doesn't have an assignment so far

Once you pick a variable, which value do we assign to it first?

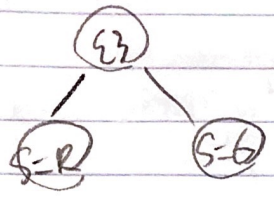
Rule: Choose the value that removes fewest options from neighbours



- Pick unassigned var
  expand C add assignments for all remaining valid values



Choose first variable w fewest values left to try!

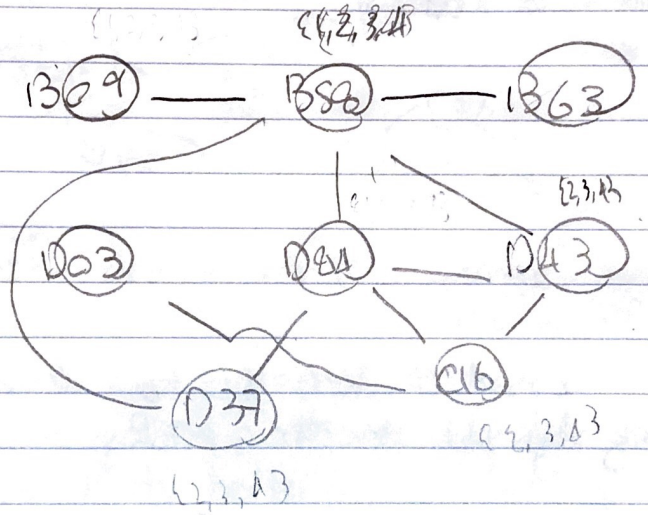


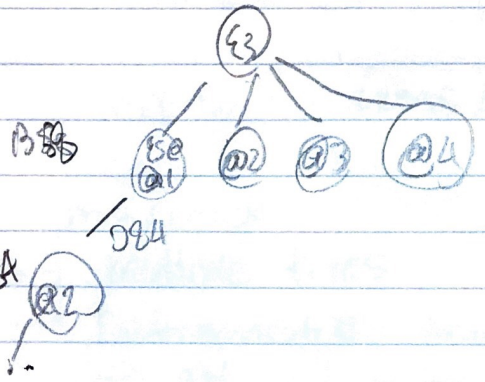
Rule 2 - If tie for R1, choose the variable w/ most active constraints

Once you choose a variable
which values to assign first

- choose value that removes fewest options from neighbours

Ex





Early termination

- After assigning value to var
  - ↳ remove conflicting assignments from connected vars
  - ↳ terminate if any variable has no assignments left

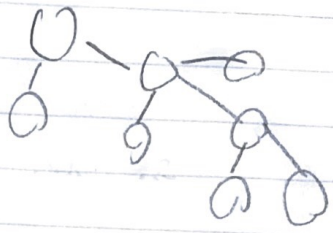
Assumes a "perfect" solution exists

Q1 -> How do you build a team level AI

Worst case: $O(d^n)$

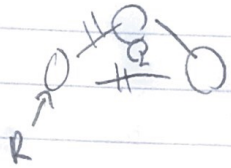
→ however for special shaped graphs

$O(nd^2)$



trees!

Suppose I have a nearly tree-structured CSP



Choose c variables

$O(d^c(n-c)d^2)$

Assign value

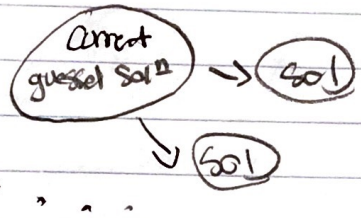
- Choose a subset of variables to give values to so remaining graph is tree-CSP

CSC1084 - Artificial Intelligence

N Queen problem: Arrange n queens on an N×N board s.t they don't threaten each other

→ Backtracking takes too long
→ Approx soln for unmanagable CSP

Local search (version of gradient descent but no gradient)
neighbour = at most 1 var changed val
repeat til good enough



→ Take current guess, change a var, if better keep, else revert

N Queens: pick queen, move to diff location in col, see if its better

→ can also try to pick space on col w min conflict

init sol? Guess or use prev stuff

Problems

→ for min/max
→ multiple tries
→ Deterministic annealing
→ get init values, setup init val of T (temperature)
loop: new queens = change on guess
compute ΔE = conflict(new) - conflict(old)
if better ($\Delta E < 0$): keep new
if $\Delta E > 0$, keep new w prob $\propto e^{-\Delta E/T}$ (proportional to)
* keep new if roll dice $e^{\Delta E/T}$
else decrement T by a bit

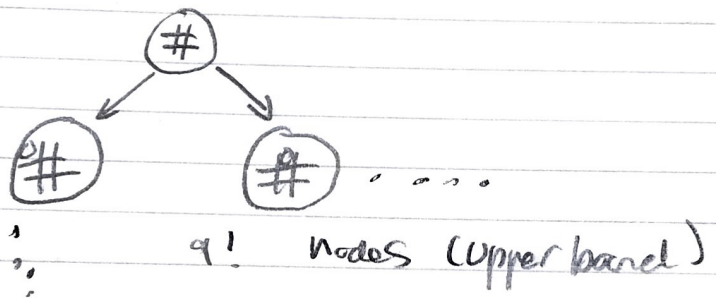
Beam Search

→ keep top k queens boards at given time

Art colouring opt

2 player adverserial games

Tic tac toe - Search problem!



9! nodes (upper bound)

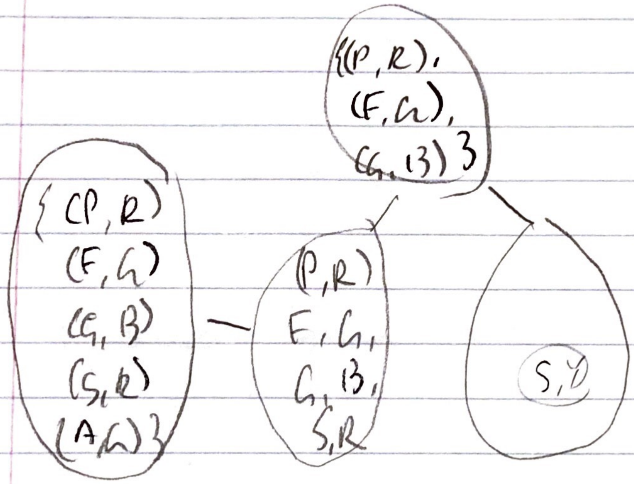
Utility fns

\+ if I win

\- if I lose

CSC D84 - AI tut

1) Pick variable w least remaining valid values
2) Pick variable with most active constraints
3) Choose value that allows the most remaining valid val's for neighbours



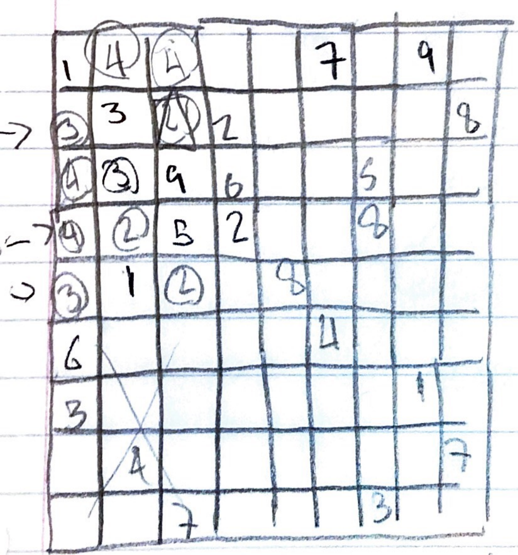
Q2 (4^{10})
$O(d^n)$

Q2) take out Germany/Switz

Q2) $d(4^2 8 \cdot 4^2)$
$O(d^c \cdot (n-c) d^2)$

- d^c: brute force vals for Switz, Ger
- $(n-c)$: remaining values in tree
- d^2: every pairwise node's only affected by that adj SC

Q3



Constraints

Q4
Extra

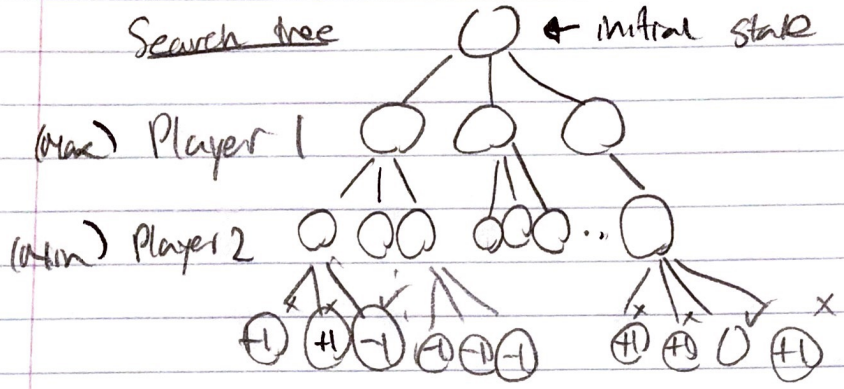
Apple map: S→L
using CSP (have V, E)

n vars ← ith node on the path
Constraints whether node is visited
whether can go from i to i+1

CSC084-AI

Game Playing

Search tree



utility function

(+) P1 wins

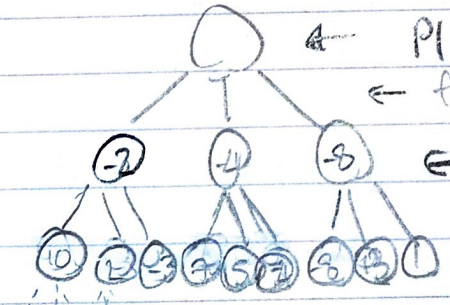
(-) P2 wins

Minimax Search Algo

→ create start node (init game)

while (game on)

- determine if it is min or max turn
- search tree to predefined depth
- Apply utility fn at each of the "leaf" nodes
- "Back-up" (propagate back) utilities choosing min/max as appropriate
- choose best move



for dcgs

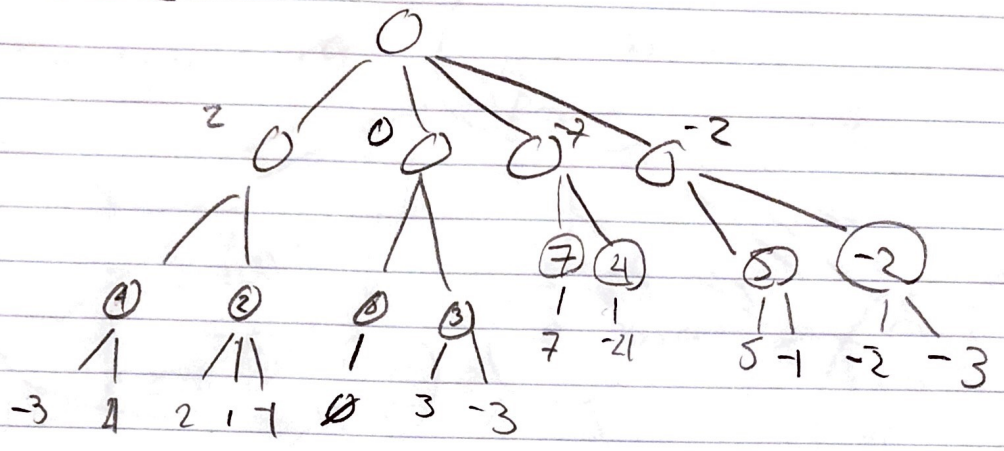
$b \approx 35$

$m = 106$

$O(b^m)$ ← max depth

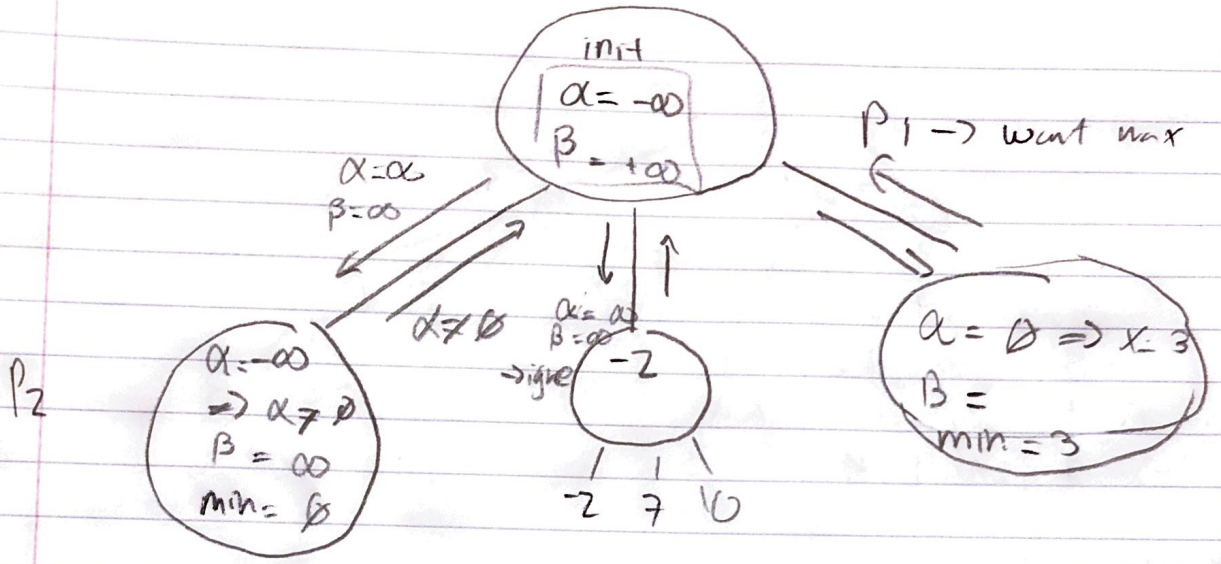
↑ branching factor

CSCI 561 - AI



Utility(c)
↳ f_i → features can eval, like # of knights
↳ $\sum w_i f_i$

Make faster search
- give any branch that won't provide better sol than sol I already have
↳ Branch & bound
= α, β pruning



Rules:

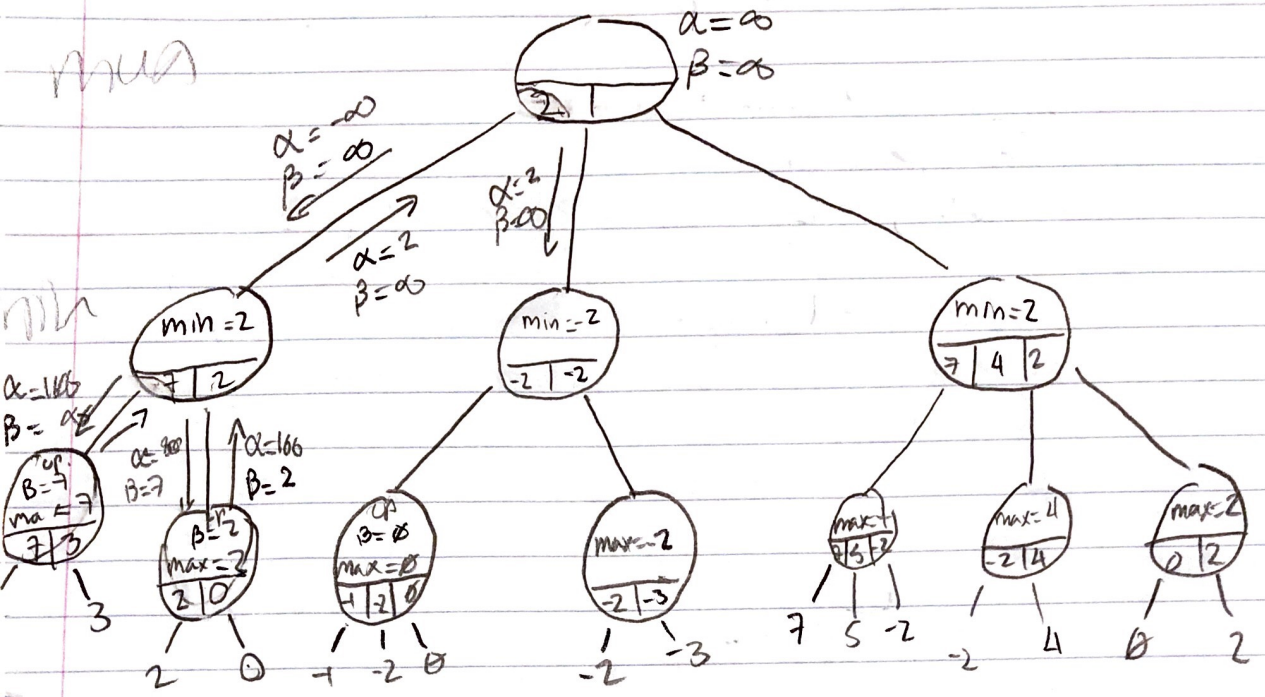
min node - update α whatever smaller val found
↳ Stop prune if found val < α

max node - update α when larger val found
↳ Stop/prune if val > β

- P_1 - max nodes
  - Update α (>)
  - prune β (>)

- P_2 min nodes
  - Update α (<)
  - prune β (<)

CSE024 - AI

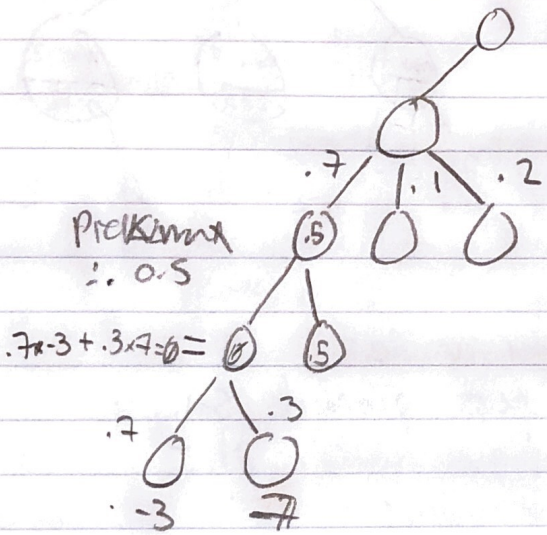


α & β is diff per node:
Update based on next potential sol.

Q. B gives 2x deeper search possible

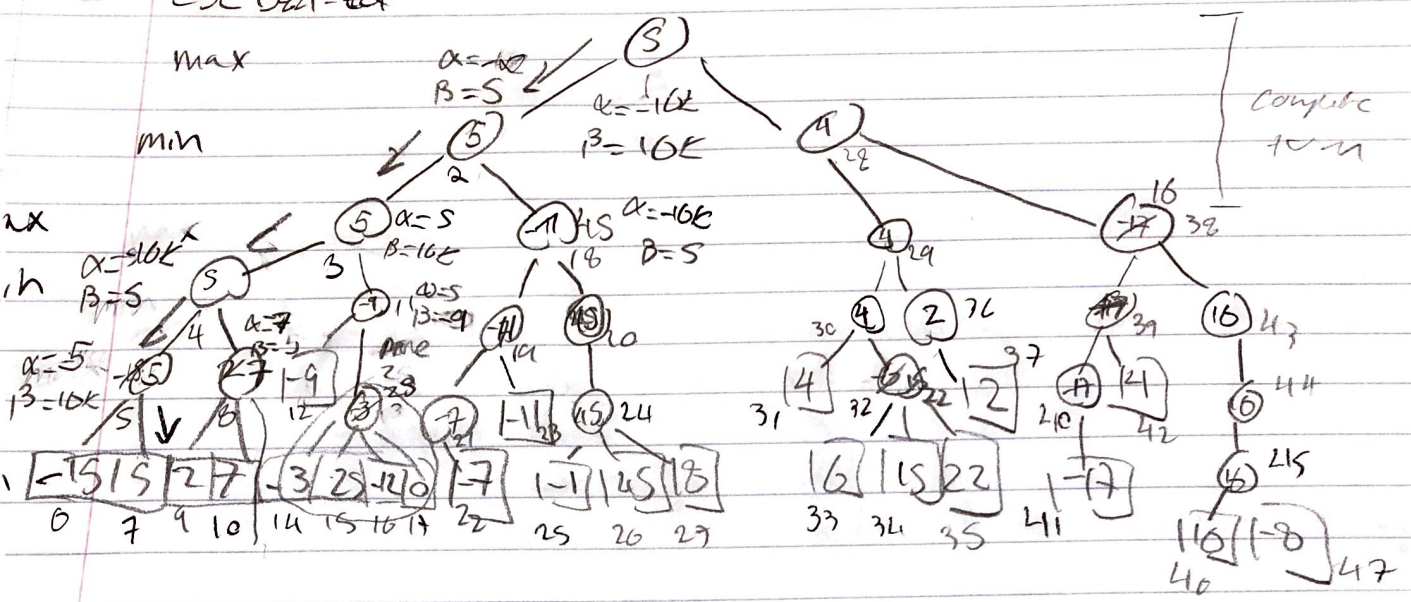
Games w/ randomness
Util fn need prob in it
→ weight options w/ prob => prob × val
↳ use expected

Rational agent: max expected utility
expecti-minimax, expectiminimax, expectimax



prune condition $\alpha \geq \beta$

CSC D22 - tut



Q1 On the tree above, write next to each node the order in which the node is expanded by minimax search

Q2 for each level of tree above where nodes are max or min nodes. How many complete terms are there? = 5

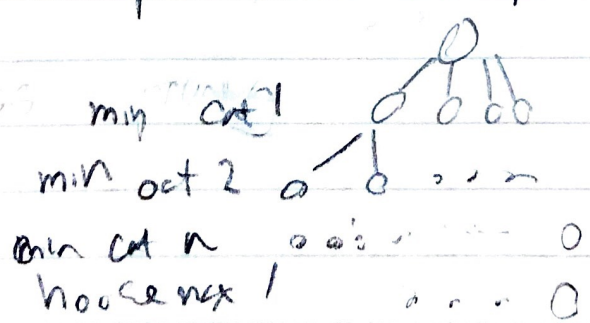
"once every player makes a move, count as a term"

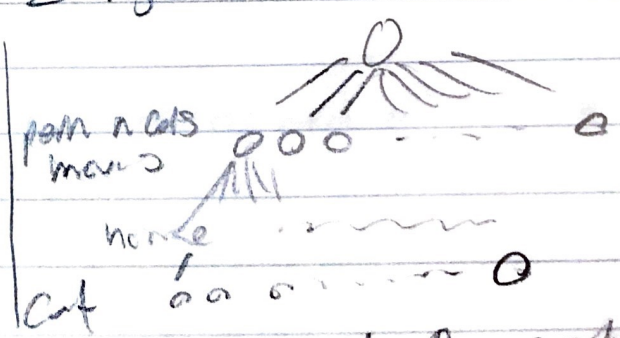
single agent max is max depth	all agents; 2 moves

Q3 max each node is scores

Q4 node pts (n)

Q5 prune → multiple cuts & single max





resolves depth problem

ISC D84- AI

Reinforcement Learning

→ figures out stats of what actions result in rewards

Framework

- discrete time steps
- agent gets info about env → sense, get state/data
  let Δ = state
- agent does action a.
  ↳ change is δ'
- agent get reward R(+/-) as result of a, δ'
- Rpt to step 1

Goal: Max expected reward

Model:
- Set of state S
- Set of actions A
- domain of reward fn (D)

Agent learns policy π which maps to states → a
(aka more than 1 possible neighbor)

Assumptions:
- state transitions are not deterministic
- env is static
- rewards are immediate.

Goal: $\mathcal{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$ $\gamma < 1$

- γ → discount factor
- r_t ↳ reward @ time t
- $\mathcal{E}$ → Expected

aka Markov Decision Process (MDP)

Transition fn T
$T(s, a, s') =$ probability

Learning: obtain estimate / learn

$$V^*(s) = \max_{\pi} \mathcal{E}\left(\sum_{t=0}^{\infty} \gamma^t r^t\right)$$

opt vals of state s

→ i.e. find π / max expected val

equiv → $V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^*(s') \right)$

$\pi^*(s) = \arg\max_a (V^*(s))$ how good next state is? val for a passive reflex

↳ gives action that maxes $V^*(s)$
↳ aft $V^*(s)$ is obtained

find V^* - value function method
- init $V^*(s) = 0$
- loop until π is good enough
  for s in S:
    for a in A
      $Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')$ ↳ s'

$V(s) = \max_a Q(s,a)$
$\pi(s) = \arg\max_a Q(s,a)$

@ beginning Q(s,a) is all 0; so only time val ↑ is when eat cheese / dead

CSC084-AI

Last Class

R2 -> utility not known
↳ don't want to hard craft

↳ MDP - States -> Configuration of games
- actions -> actions, a
- reward function R(s,a)
- Transition fn - T(s, a, s') = probability
Learn π^* optimal policy mapping s -> optimal action

We want

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right)$$

↑ value iteration method

start $V(s) = 0$
$Q(s,a) = 0$

Loop until policy π is good enough

for $s \in S$
  for $a \in A$
    $Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')$

$O(|S| \cdot |A|)$
?
Quite big

$V: V(s) = \max_a Q(s,a)$
$\pi(s) = \arg\max_a Q(s,a)$

Problem - T(s, a, s') -> mostly impractical to compute

- Q-learning
↳ Learn $Q^*(s,a)$ -> Expected reward of taking action a from state s

Q-Learning

↳ $Q(s,a) = 0$ initially

- Repeat until policy good enough
  - from current state s, agent chooses (+) action a at random
  - agent receives a reward r
  - and is told resulting state s'

(s,a,r,s') experience tuple

Update

$$Q(s,a) = \alpha\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

α: learning rate $\ll 1$

γ: discount factor < 1

$Q(s,a)$: current

Note: No longer need to store T, but now takes long time to converge

random (+) — training program

$t = 0$ Choose random 100% time

Choose random 90% time, π 10%

" " 80% , π 20%

training till Convergence

$t = \infty$ (practically)

Choose 10% , π 90%

CSCD84 - tut

Markove decision Process (MDP)

State space S
action space a
reward fnc $R: S \times a \to \mathbb{R}$
transition fnc $T: S \times a \times S \to [0,1]$
discount factor $\gamma: \to [0,1]$

cat	
	mouse

$|S| = 4 \times 4 = 16$
cats (mouse)

Q table: $\underbrace{10}_{|S|} \times \underbrace{2}_{|a|}$ → mouse has 2 moves at any given time

~~Pseudocode~~

init all $\emptyset$
loop until convergence → terminate if converge even if in middle of episode

observe our state S
make action a, based on π

Ep greedy
when to choose opt*
us rnd fr e
explore:
(small = choose best ?.
check.

observe S', reward r
tuple $\langle s, a, r, s' \rangle$
update:

$Q(s,a) \mathrel{+}= \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$

Temporal difference (-1) for
{ $r + \gamma Q\max_{a'}$ = actual outcome (a
{ $Q_{s,a}$ = est outcome (guess + outcome)

conv: when opt move doesn't change but not nessarity its val/

CSC084 - AI lec

In changing env, RL learning doesn't work well
→ can encode changing env into states

- Learn from some setting of problem a policy that on similar setting 8

- replace state based learning w/ "features"

feature = measurement of some property (s) of problem that provides useful info for agent.

Set of features:

$f_i(s) = \text{value} \in [-1, 1]$

↑

looks @ config

$F(s) = \begin{bmatrix} f_1(s) \\ f_2(s) \\ \vdots \\ f_n(s) \end{bmatrix}$ → if encode all possible ru too large

→ easy to compare 2 states b/c of vector

Replace $Q(s, a) - Q(a, s)$

$Q(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

↳ not stored, evald as func

how to get weight?

Eg of features
a) α dist to cheese f_1
b) dist to cat f_2
path

reward f_1 → shortest dist → want to make Q(s) ... Q(s) if close to cheese
a cat
die
⇒ $\frac{1}{\text{shortest dist}}$ → $\frac{45}{1+\text{dist}}$
↓
not well defined

$f_2 = \frac{-45}{1+\text{dist}}$

Learning → init weight: small, and non-0 val
eg (-0.5, 0.5)

Standard Q-learning (process)
given (s, a, r, s')
update $w_i = w_i + \alpha(r + \gamma Q(s') - Q(s)) f_i(s)$
grad descent w/ weights
→ can get stuck in local min

starting weight sign doesn't matter → will converge to correct sign

f_n has to be smaller than r to guarantee conv to the correct dir

$(r + \gamma Q' - Q) f_i$

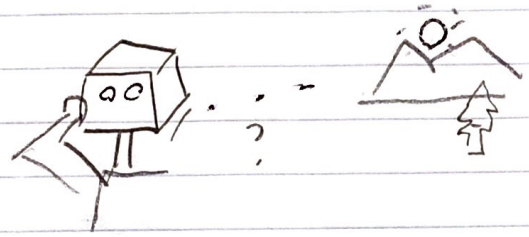
if $w_i(-)$ → means bad r corrects for it

Q state s', eval $Q(s')$ to simulate Q pilot

4/3/2019

CSC 484 - AJ Lee

Probability & Statistics ← Decision Making under [uncertainty]

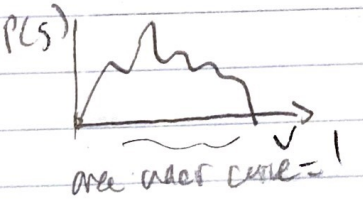


- Sensor noise!
- ambiguity! → bad incomplete model
- imperfect sensors → unobserved variables

Random variable

↳ variable whose value depends on the outcome of a random event

representation: discrete ≈ Prob mass fn
Continuous ≈ prob distribution fn

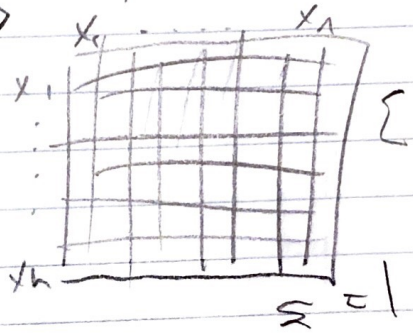


Joint distribution: Variables $X_1, X_2, \ldots, X_n$ $P(X_1, X_2, \ldots, X_n)$

$P(X_1 = v_1, X_2 = v_2, \ldots, X_n = v_n)$ ← specific event!

discrete vars X_1, X_2

figure this out or approximate from other distributions?



- Marginal distributions

Temp	Weather	P(T,W)
hot	sun	.4
hot	rain	.1
cold	sun	.2
cold	rain	.3

→ p(hot) = .5

= $\sum_{T=hot} P(T,W)$

marginal prob of T = hot

hot cold

Σ = 1

	T=0	T=1
W=∅	.4	.2
W=1	.1	.3

Conditional distribution

a=v_a ← we know value of b

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

- $p(W=rain \mid T=cold) = \frac{.3}{.3+.2=.5} = .6$

Product rule

$$p(x_1, x_2, \ldots x_n) = p(x_1 \mid x_2 \ldots x_n)\, p(x_2 \mid x_3 \ldots x_n)$$
$$p(x_3 \mid x_4 \ldots x_n) \cdots \times p(x_{n-1} \mid x_n) \cdot p(x_n)$$

↳ breaking up joint dist → to smaller ones

Bayes Rule

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

p(a|b): disease, symptom — posterior

p(b|a): fever, likelihood, cold

p(a): prior — p=cold?

p(b): P(have a fever) — evidence

p(h|a) ← check this for all a

↑ ↳ choose max

map (low disease, prob 100% the

p(cold) >> prob(m cd)

CSC084 - AI - lec

Conditional Independence

$P(Smoke=1, alarm=1, fire=1) = P(smoke=1 \mid alarm=1, fire=1)$
$* = P(alarm=1 \mid fire=1) * (P(fire=1)$
↳ so complicated

observation

obvs smoke = 1
↳ p(fire=1) is indep of p(alarm=1)
↳ fire & alarm are conditionally independent given smoke

fire cause smoke cause alarm
fire → smoke → alarm
→ all indep on ea other
but being smoke vs P(alarm=1) regardless of fire or not(

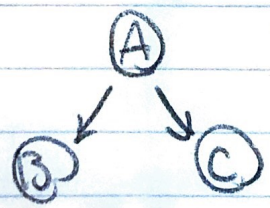
Bayes Network

model cond Problem among vars for some problem or environment using graph (DAG)

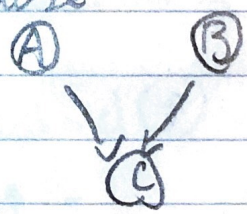
ex Panam → traffic → lateness

direct cause

indirect cause



"common cause"

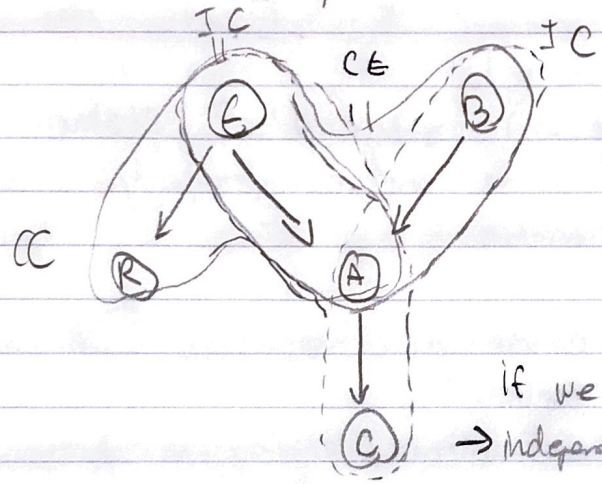


"common effect

Ex w SD Arr $P(R, E, B, A, C) = P(R|E,B,A,C) \cdot P(E|B,A,C) \cdot P(B|A,C) \cdot P(A|E,C) \cdot P(C)$

Ex w DAG $P(\dots) = P(E) \cdot P(B) \cdot P(R|E) \cdot P(C|A) \cdot P(A|E,B)$

↳ Slightly different? (if not ↳ cool guess you)



Suppose $P(R=\emptyset, E=1, \dots) = 3$

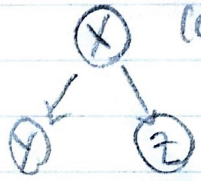
	$E=\emptyset$	$E=1$
$R=\emptyset$	0.6	0.7
$R=1$	0.4	0.3

if we observe R=1 instead
→ independence on chance of E=1

(obvs. smoke ↑ chance of fire)

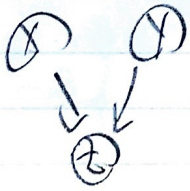
if Y is known → $X \perp\!\!\!\perp Z$

Chain y makes x, y decouple. i.e
x, z are cond given y



$Y, Z \not\perp\!\!\!\perp | X$

if X known → $Y \perp\!\!\!\perp Z$



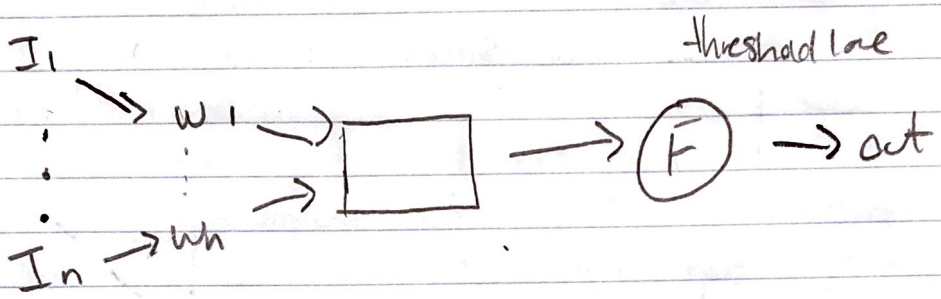
$\Rightarrow X \not\perp\!\!\!\perp Z$

if Y known
$X \perp\!\!\!\perp Z$ not necessarily

CSC484-AI

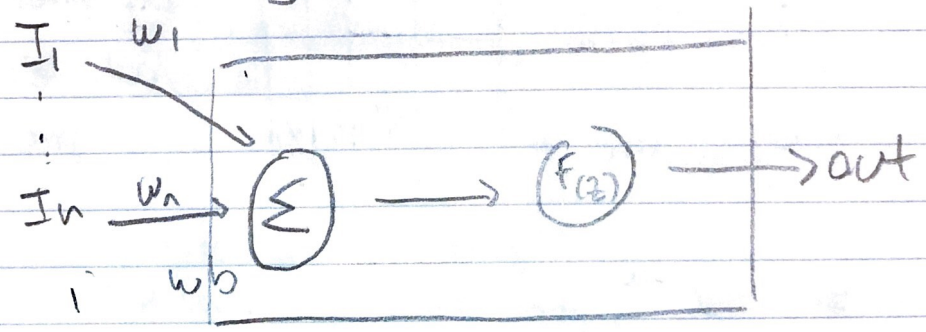
Neural Networks - McCulloch-Pitts Model

Inputs Weights (learned)



ANN
- ↳ simple units → neurons
- ↳ connected to form network
- ↳ trained by RL

-
- compute output
- give feedback

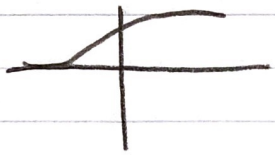


y to shift threshold

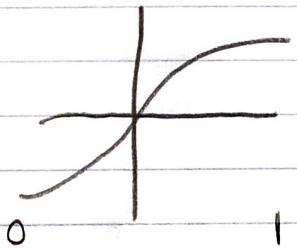
$out = f(\sum_i I_i \cdot w_i = T)$

activation fns

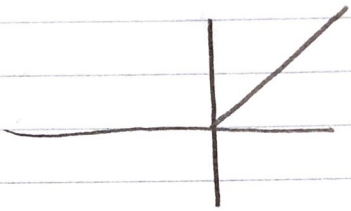
(step) threshold



sigmoid (logistic)

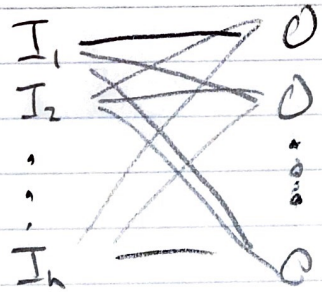


sigmoid (tanh[I])

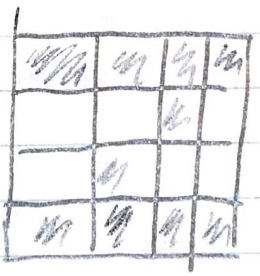


RELU (rectified linear unit)

example network



Pixels



on = 1, off = 0

ex. pixel is an input output. detect if Z is present

eg. $w_{=}$ 0 $-w$ for black

$w_{==}$ 10 for range

Now we can do each node is different & we can say all letters!

How to handle noise and/or bad writing?

init weight $[-.5, .5] \neq 0$

Algorithm

loop:

a) feed foward pass
- give sample to network, compute output

b) Compute

err = target - out

→ suppose 6 detector 1 sample is on S

target = 1, out = ? (e.g 0.5)

∴ err = 0.5

squared err = (target - out)^2

c) Adjust weight to minimize err (Chain rule)

→ Error back propagation d = derivative

$$\frac{d\,Error_S}{d\,weight_{AB}} = \frac{d\,In_B}{d\,W_{AB}} \cdot \frac{d\,Out_B}{d\,In_B} \cdot \frac{d\,Err_B}{d\,out_B}$$

$$\frac{d\,Err_B}{d\,Out_B} = -(tgt_B - out_B) \rightarrow \text{from } (tgt - out)^2$$

absorbed into weight, just use $(tgt_B - out_B)$

$$\frac{dOut_B}{d\,In_B} = \frac{d\,f(In_B)}{d\,In_B}$$

if act fnc is logistic fnc

eg $f(x) = \frac{1}{1+e^{-x}}$, then

$$\frac{df(x)}{dx} = f(x)(1 - f(x))$$

if $f(x) = \tanh$

Then $\frac{df}{dx} = 1 - \tanh^2(x)$

What about $\frac{dI_{1B}}{dw_{AB}} = ?$ $\quad I_{1B} = \sum_i w_i I \rightarrow \frac{dI_{1B}}{dw_{AB}} = \alpha I_{1B}$

$\therefore \frac{dE_B}{dw_{AB}} = \frac{dI_{1B}}{dw_{AB}} \cdot \frac{dO_B}{dI_{1B}} \cdot \frac{dE_B}{dO_{1B}} = \alpha o_A \frac{df(I_B - \alpha I_B)}{dx}$ or we act from prev node

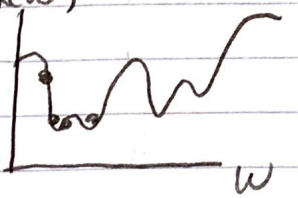
$w_{A1B} = w_{AB} + \alpha \frac{dE_B}{dw_{AB}}$

↑ learning rate; small ϵ

ESC084 - AI - ~~ed~~

Loss fcns

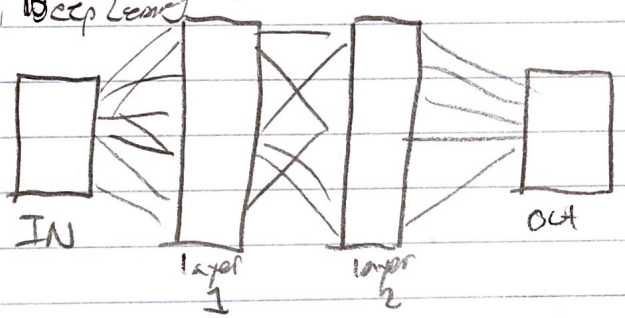
→ measure how well model on training set
→ Goal: min loss wrt weights $\mathcal{L}(w)$
1) find ~~closed~~ closed form
2) gradient descent



Deep Learning

→ A class of ML Algos
→ Use layer to conceptualize hierchies and abstractions

eg: Deep Learning

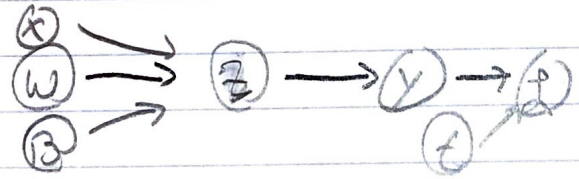


Forward pass

1) $z = wx + b$
2) $y = \sigma(z)$
3) $\mathcal{L} = \frac{1}{2}(y-t)^2$

$x = in$
w, b = params
$z = ax + b$
$y = \sigma(z)$ (prediction)
$\alpha = \frac{1}{2}(y-t)^2$
t = target

Computational Graph



How does w, b affect $\mathcal{L}$

$\Rightarrow \frac{\partial \mathcal{L}}{\partial w}, \frac{\partial \mathcal{L}}{\partial b}$

$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w}$

$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial b}$

(by chain rule and diagram)

Backward Pass

$\frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$

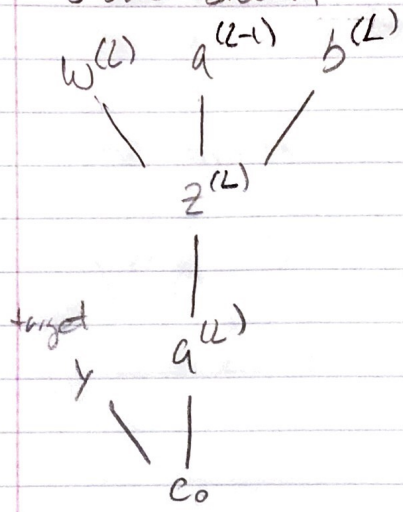
$\frac{\partial \mathcal{L}}{\partial y} = y - t$

$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial z} = (y-t)\frac{\partial y}{\partial z}$

$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w}$ also $\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial b}$

$\Rightarrow \frac{\partial \mathcal{L}}{\partial w} = (y-t)\cdot\frac{\partial y}{\partial z}\cdot\frac{\partial z}{\partial w} = (y-t)\sigma'(z)\cdot x$

3 blue 1 brown



$C_0 = \frac{1}{2}(a^{(L)} - y)^2$ "error"

$z^{(L)} = w^{(L)} a^{L-1} + b^{(L)}$

$a^L = \sigma(z^{(L)})$

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C^0}{\partial a^{(L)}}$$

$$\frac{\partial C_0}{\partial a^{(L)}} = a^{(L)} - y$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

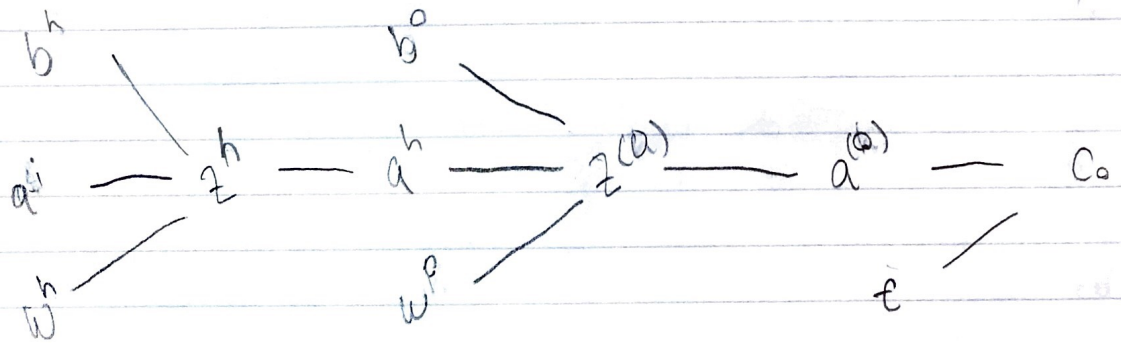
$$\frac{\partial (z^{(L)})}{\partial w^L} = a^{L-1}$$

$$= (a^{(L)} - y) \cdot \sigma'(z^{(L)}) \cdot a^{L-1}$$

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$$

derivative of full cost fn

average of all training examples

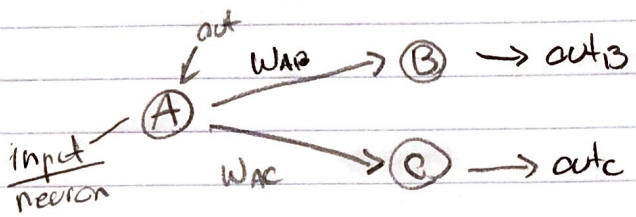


2 layer NN

$$\frac{\partial C_0}{\partial w^0} = \frac{\partial C_0}{\partial a^0} \cdot \frac{\partial a^0}{\partial z^0} \cdot \frac{\partial z^0}{\partial w^0} = (a^0 - y) \cdot (\sigma'(z^{(0)})) \cdot a^h$$

CSCI 384 - AI lee

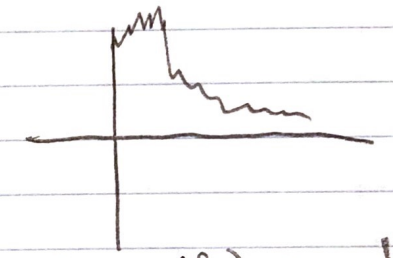
→ 1 layer Net works
↳ weight updates



Backpropagation

$$\frac{\partial out_B}{\partial In_B} \rightarrow \frac{\partial out_B}{\partial W_{AB}} \rightarrow \frac{\partial E}{\partial A_{1B}}$$

$$\frac{\partial Err_B}{\partial W_{AB}} = \frac{\partial I_{nB}}{\partial W_{AB}} \cdot \frac{\partial out_B}{\partial In_B} \cdot \frac{\partial Err}{\partial out_B}$$



$$In = \sum w_i I_i \quad out_A \quad \frac{\partial f(x)}{\partial x} (Tgt - out)$$

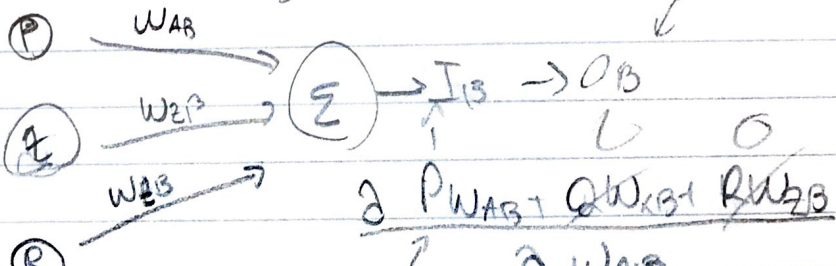
$$Err_O = \frac{\partial f(x)}{\partial x} \cdot (tgt - out_B)$$

↑ activation fn derivative

$$W_{AB} = W_{AB} + \alpha\left(\frac{\partial Err_B}{\partial W_{AB}}\right)$$

→ Input Layer

$\sigma(I)$ activation



$$\frac{\partial P W_{AB} + Q W_{KB} + R W_{ZB}}{\partial W_{AB}}$$

P

$$err_B = \frac{\partial out_B}{\partial in_B} \cdot \frac{\partial Err_B}{\partial out_B} = (Tg_B - out_B)^2$$

$$err_C = \frac{\partial out_C}{\partial in_C} \cdot \frac{\partial Err_C}{\partial out_C} = (Tg_C - out_C)^2$$

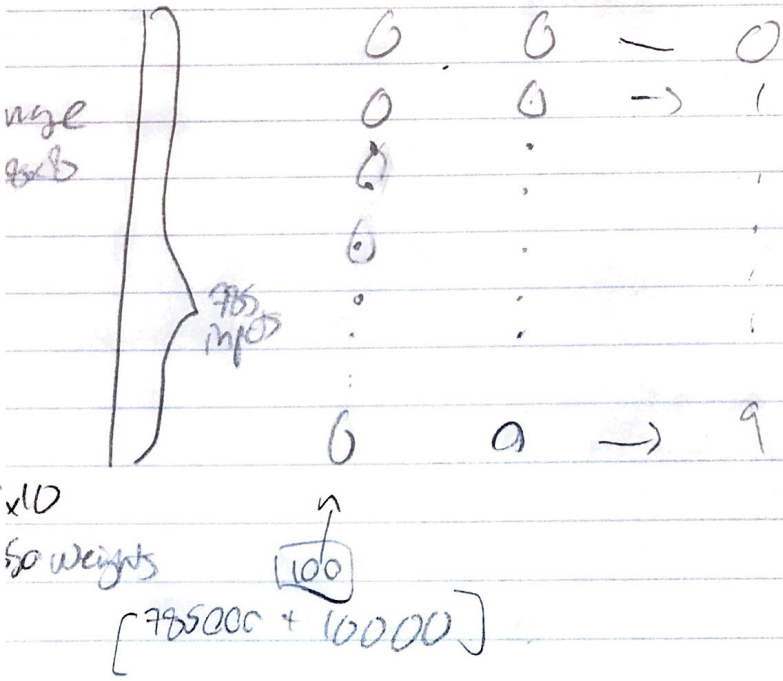
$$\frac{\partial Err_A}{\partial out_A} = W_{AB}\, err_B + W_{AC}\, err_C$$

$$\frac{\partial Err_A}{\partial W_{BA}} = \frac{\partial in_A}{\partial W_{BA}} \cdot \frac{\partial out_A}{\partial in_A} \cdot \frac{\partial Err_A}{\partial out_A}$$

$$= out_B \frac{\partial \ell(x)}{\partial \ell(x)}$$

NB. - Store every nodes out
- alot of storage for 1 layer

Questions → What shape is network?



x10

50 weights

100

[785000 + 10000]

→ Hard deep! Deep Learning

↳ 4+ layers — weights — large # input samples — large + costly

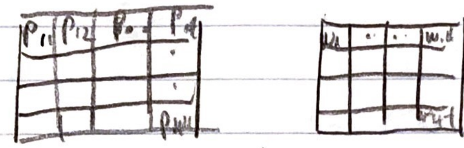
↳ Weight update harder

CSCD84-AI

Convolutional Neural Networks

→ neural node learns a little of of all theory

→ indent better in image, take part of img (same size as temple, eg ☐)



Sampling from img

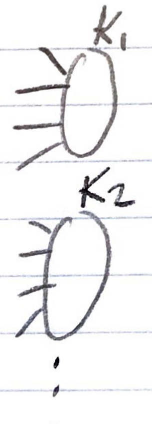
$\sum P_{ij} \cdot W_{ij}$ → like in * weight!

convolution in 2D

→ Pattern is called "kernel", small relatively compare to img

→ reduce # weights to learn into neurons = size of kernel

→ Can do layers of NN



eg learn diff kernel

→ K features maps (where on each find pattern)

→ feed feature map in to another NN

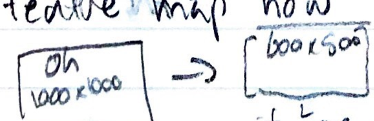
→ feature map on 1st is same size as able img

NB: Neurons don't need to be (:) things are recognizing, but # out nodes need to be same as # classification

Apply to every pixel is increasing work

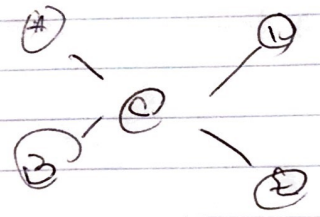
→ skip every 2, 3, ... pixels (can do b/c of trained to recognize slightly diff)

→ feature map now shrinks for each it exponentially



→ help w size lines

if ... are oks 250 x 250 ...



$\frac{\partial in}{\partial w} \quad \frac{\partial out}{\partial in} \quad \frac{\partial err}{\partial out}$

$out * (1 - out)$

ACDE

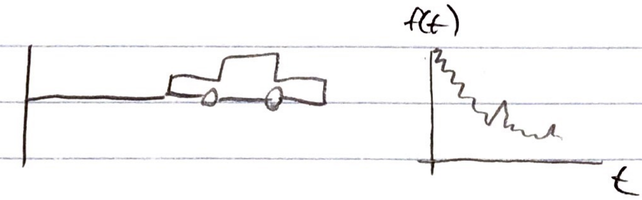
CSED984-tut

Convolutional Neural Nets (CNN, conv net)

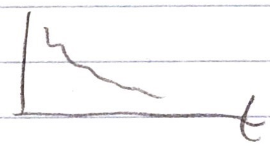
ml4a.github.io/demos/ convolution_all
& cs.toronto.ca/andrej/ ... /index.html

Convolution: $(f*g)(t) = \int f(t-a)g(a)\,da = \sum_a f(t-a)g(a)$

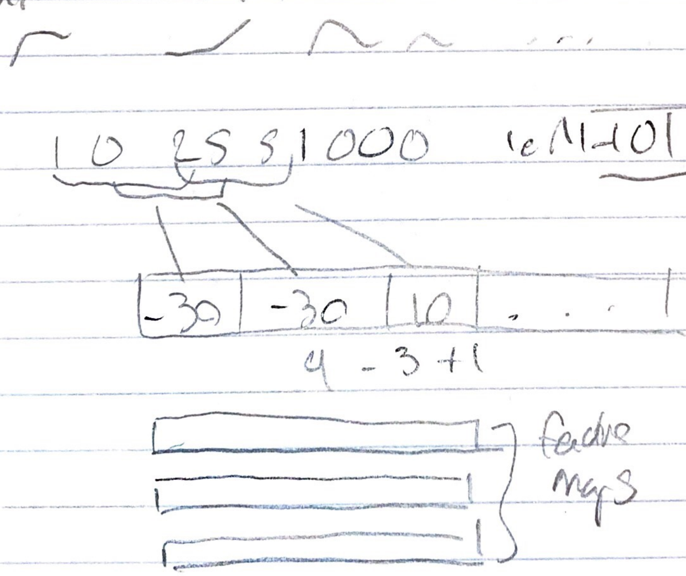
f: input; g: kernel/filter; result: feature map



f(t) → relative position btw wall and car weighted probability fn g(a) (age is age of measurements)



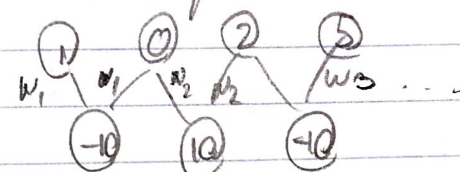
ex image
want to detect feature



← feature map

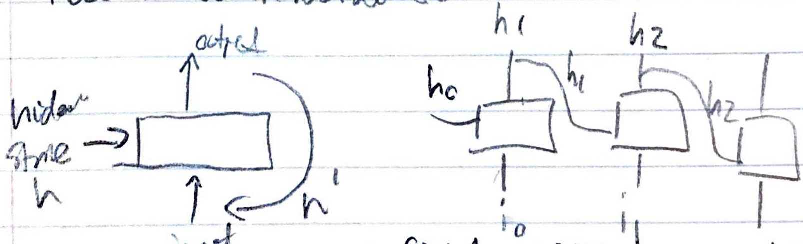
input 10
hidden units 100
# of params 10x100

input 10
hidden units 100
# of params 10x100



conv: 3 params
FC: 4x3 params

Recurrent Neural Networks



String to compressed, e.g. d

LSTM BPTT | Word2Vec
GRU | BERT