Artificial Intelligence - Summary

École Centrale de Nantes, Davide Lanza

Art. In. Summary

1. Optimal Strategies in Deterministic Environments

• Directed Graphs:

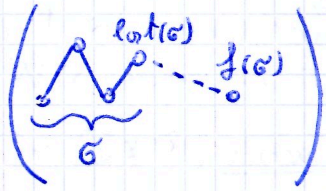
A D.G. is a pair (V, E) s.t. V = set of vertices and $E \subseteq V \times V$ = set of edges

(edge: $v \to v'$ when $(v, v') \in E$) (label sets Λ and Σ for E and V)

• Strategy:

A S. is a function $f: V^* \to V$ s.t. $\forall \sigma \in V^*, (last(\sigma), f(\sigma)) \in E$

(the set of finite sequences of elements of V) (last element of sequence σ)



• Positional Strategy

A S. f is P. (or memoryless) if $\forall \sigma, \sigma'$; $last(\sigma) = last(\sigma') \Rightarrow f(\sigma) = f(\sigma')$

(P.S.s are functions $V \to V$) ↳ (is an "array" with strat(state) = action with the MDP lexicon)

• Outcome

Outcome(v, f) of a strategy "f" from vertex "v" is the set of vertex sequences inductively defined by:
- $\Rightarrow v \in$ Outcome(v, f)
- $\Rightarrow$ if $\sigma \in$ Outcome$(v, f) \Rightarrow [\sigma . v' \in$ Outcome$(v, f) \Leftrightarrow v' = f(\sigma)]$

(is the sequence of vertices produced by the agent when it apply a strategy)
(the outcome represents the successive result of each action)

! [environment fully observable $\wedge$ only one agent $\wedge$ deterministic actions] $\Rightarrow$ [outcome of a strategy is a single path $\Rightarrow$ the strategy consists in, $\forall$ vertex of the path, going to the next one] !

• Objective

An OBJ. is a set of vertex sequences

• Maximal Sequence

A vertex sequence is MAXIMAL within a sequence set X $\Leftrightarrow$ (is infinite) $\vee$ (is NOT A PREFIX of a sequence $\in X$)

↳ (when $X = V^*$ the sequence is maximal)

• Max Outcome

MaxOutcome(v, f) is the subset of sequences of Outcome(q, f) that are maximal within Outcome(q, f)

• Reachability of the Objective

Given a target vertex g, the R.o.O. Reach(g) is the set of maximal seq. σ s.t. $g \in \sigma$

Winning Strategy

A strategy is winning from some vertex "V" to some objective "X" $\iff$ MaxOutcome$(v, f) \subseteq X$

Winning Vertex

A vertex "v" is winning if $\exists$ winning strategy from "v"

Spanning-Tree Walk

(S∅ → t reachable?) → r = false
(visited vertices) → P = ∅
(to-visit vertices) → W = S∅

while (W ≠ ∅ and not r)
- POP S_i from W
- **if** (S_i = t) → r = true
- **else** put S_i in P and:
  - ∀ S_{i+1} (successor of S_i) ∉ W, P
  - PUT S_{i+1} in W
  - REGISTER S_i AS A PREDECESSOR OF S_{i+1}

→ (if W is a queue → Breadth-First Search)^{BFS}

→ (if W is a STACK → Depth-First Search)^{DFS}

} PUSH-POP ORDER

POP S_1' FROM W — POP S_1'' FROM W

[START] $S_\emptyset$ → [SUCCESSORS OF $S_\emptyset$] → [SUCCESSORS OF S_1'] → [SUCCESSORS OF S_1'']

$S_1' \leftarrow S_0$ — $S_2' \leftarrow S_1'$ — $S_2''' \leftarrow S_1''$

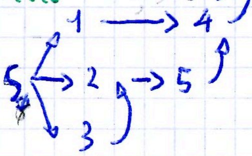
$S_1'' \leftarrow S_0$ — $S_2'' \leftarrow S_1'$ — $S_2''' \leftarrow S_1''$

REGISTER

REBUILD PATH TO S_2''' → ($S_2''' \leftarrow S_1'' \leftarrow S_\emptyset$) PATH

- EXAMPLE: (S → 4) reachable?

SUPP. THAT WE READ FROM TOP TO BOTTOM THE SAME LEVEL !



SO WE INSERT 1, THEN 2, THEN 3 !

(DETAILED VIEW) — POP — REACHED! — END so (r = true)

[BFS] ⇒

STEP #	∅	1	2.1 → 2.2	3	4	5
W	S∅	1, 2, 3	2, 3 → 2, 3, 4	3, 4, 5	4, 5	5
P	∅	S∅	S∅ → S∅, 1	S∅, 1, 2	S∅, 1, 2, 3	S∅, 1, 2, 3,
PATH	∅	S∅ / S∅ / S∅	S∅ / S∅ / S∅ → S∅ 1 / S∅ / S∅	S∅ 1 / S∅ 2 / S∅	S∅ 1 / S∅ 2 / S∅ 3	S∅ 1 4 / S∅ 2 / S∅ 3 ← PATH

! (POP ← 1, 2, 3 PUSH)

REACHED → r = (true)

[DFS] ⇒

STEP #	∅	1	2	3	4	5
W	S∅	1, 2, 3	1, 2	1, 5	1, 4	1
P	∅	S₀	S∅, 3	S∅, 2, 3	S∅, 2, 3, 5	S∅, 2, 3, 5
PATH	∅	S∅ / S∅ / S∅	S∅ / S∅ / S∅ 3	S∅ / S∅ 2 / S∅ 3	S∅ / S∅ 2 5 / S∅ 3	S∅ / S∅ 2 5 4 / S∅ 3 ← PATH

! (1, 2, 3 ⇄ POP / INSERT)

PG 81 "AI" BOOK

DFS → EXPLORE A PATH UNTIL THE END, THEN THE NEAREST TO THE END etc...

F-DFS → "FORGETFUL", REMEMBERS ONLY THE CURRENT PATH (SEE PG. AFTER)

BFS → EXPLORE A "LEVEL" OF DEPTH, THEN FOR EACH NODE OF THE CURRENT LEVEL THE "DEEPER" LEVEL

Forgetful DFS

~~($s_\emptyset \to t$ reachable?) $\to$ r = false~~
~~P = ∅, W = $s_\emptyset$,~~
~~while (W ≠ ∅ and not r)~~
~~pop s_i from W~~
~~if (s_i = t) $\to$ r = true~~
~~else put s_i in P and ↓~~
~~$\forall s_{i+1}$ (successor of s_i) ∉ W, P~~
~~put s_{i+1} in W~~
~~register s_i on PREDEC. of s_{i+1}~~

DFS/BFS

→ NO MEMORY →

⇒

OUTPUT   INPUT

(r, PATH) function fDFs (s_i, t, PATH)

PATH' = [PATH], [s_i] ← CONCATENATION

if (s_i = t) → r = true

else: r = false

$\forall$ successor s_{i+1} OF s_i AND while not r ↓

[(r, PATH'') ← f DFS (s_{i+1}, t, PATH')
if (r) → PATH' = PATH'']

RECURSIVE!

- example:

$s_\emptyset$ → 1 → 4
$s_\emptyset$ → 2 → 5
$s_\emptyset$ → 3

fDFS (S, 4, PATH$_\emptyset$) } PUSH 1, 2, 3 POP 3 (DFS)
PATH$_1$ = S$_\emptyset$
fDFS (3, 4, PATH$_1$) } PUSH 2 POP 2
PATH 2 = S$_\emptyset$, 3
fDFS (2, 4, PATH2) } PUSH 5 POP 5
PATH$_3$ = S$_\emptyset$, 3, 2
fDFS (5, 4, PATH3)
PATH$_4$ = S$_\emptyset$, 3, 2, 5
FOUND 4!

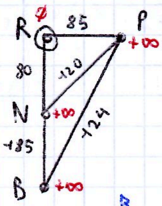
→ PATH$_3$ = PATH$_2$ = PATH$_1$ = PATH$_\emptyset$ = S$_\emptyset$, 3, 2, 5
(PATH$_4$ =)

[$S_\emptyset$ → 3 → 2 → 5 → 4]

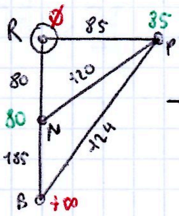
Completeness

A search algorithm is COMPLETE if whenever ∃ a path to the target vertex, it always finds one.   → (BFS complete, DFS NO.)(?)

Dijkstra's Algorithm



→

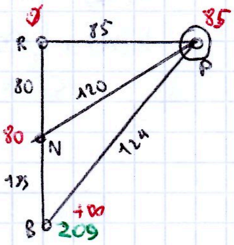


[ALL +∞ BUT START (R) ∅.
WE WANT TO ARRIVE IN (B)
WE CAN MOVE TO (P) AND TO (N).]

[85 < +∞
SO WE CAN MOVE TO (P)
↳ CHANGE THE WEIGHT TO 85]

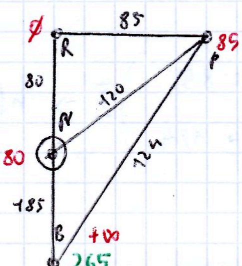
[IDEA FOR (N)]

MOVE ON P →



[FROM (P) I CAN MOVE EVERYWHERE BUT:
(R)? 85 + 85 > ∅ NO
(N)? 85 + 120 > 80 NO
(B)? 85 + 124 < +∞ YES
I CHANGE THE WEIGHT OF (B) TO 85 + 124 = 209]
BETTER (209)

MOVE ON N →



[(R) 80 + 80 > ∅ NO
(P) 80 + 120 > 85 NO
(B) 80 + 185 < +∞ YES
↳ WEIGHT (B) = 265]
WORSE (265)

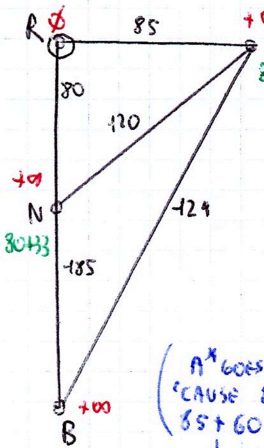
- # A^* ALGORITHM

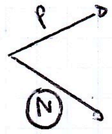
DIJKSTRA knows only the PATH-COST $g(n)$ TO ARRIVE TO VERTEX "n"

↳ a HEURISTIC $h(n)$ ESTIMATES the cost from vertex "n" to target "t"

DIJKSTRA minimizes $g(n)$, A^* MINIMIZES $f(n) = g(n) + h(n)$

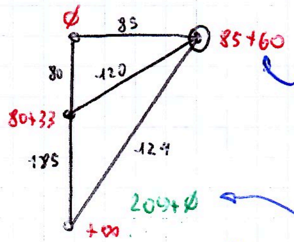
[FLIGHT TIMES TO B:
R -> 45
N -> 33
P -> 60
B -> Ø] HEURISTIC





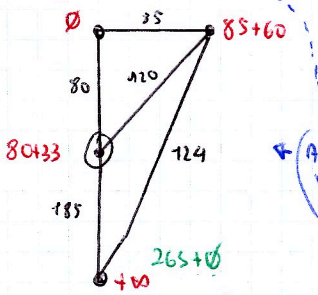
(A* GOES HERE 'CAUSE 80+33 < 85+60)

(BUT THE OPTIMAL SOLUTION IS FOR P) -> (NOT ADMISSIBLE / BAD HEURISTIC) -> (DOES NOT FIND AN OPTIMAL PATH)



HEURISTIC: WE USE IT ONLY FOR
85+60 < +∞
85+60 > 80
NOT IN THE CALCULATION OF THE TOTAL COST

(OPTIMAL SOLUTION)



(ACTUAL SOLUTION WITH THIS HEURISTIC)

A different heuristic could be "TRAIN TIMES TO B" ↳ $[h(R) = 45,\ h(N) = 185;\ h(P) = 124);\ h(B) = \emptyset]$ WE COULD TEST IF IT FINDS THE OPTIMAL PATH THIS TIME

- An A^* HEURISTIC is ADMISSIBLE ⇔ it never OVERESTIMATES the cost of the goal
- An A^* HEURISTIC is CONSISTENT ⇔ $\forall (x,y) \in E,\ h(x) \le w(x,y) + h(y)$ (COST FROM X TO Y)

- $\begin{cases} A^* \text{ finds an OPTIMAL PATH} \Leftarrow h \text{ ADMISSIBLE} \\ A^* \text{ never visits a vertex TWICE} \Leftarrow h \text{ CONSISTENT} \end{cases}$!

(MANHATTAN DISTANCE $\Delta x + \Delta y$
EUCLIDEAN DISTANCE $\sqrt{\Delta x^2 + \Delta y^2}$) !

- [h CONSISTENT => h ADMISSIBLE]

PROOF: h CONSISTENT -> $\forall x,\ h(x) \le c^*(x)$ (cost) with $c^*(x) = \sum_i^n w(x_i, x_{i+1})$

OPTIMAL PATH: $(x_\emptyset, \dots, x_n)$

$$\begin{pmatrix} h(x_\emptyset) - h(x_1) \le w(x_\emptyset, x_1) \\ h(x_1) - h(x_2) \le w(x_1, x_2) \\ \dots \end{pmatrix} \Sigma \to h(x_\emptyset) - \cancel{h(x_n)}_{(=\emptyset)} \le \sum_i^n w(x_i, x_{i+1})$$

√ QED

FOR THE "FLIGHT TIME" HEURISTIC:

R -> P -> B

$$\begin{pmatrix} h(R) - h(P) = 45 - 60 \le w(R,P) = 85 \\ h(P) - h(B) = 60 - \emptyset \le w(P,B) = 124 \end{pmatrix} \Sigma \to h(R) = 45 \le 209$$

-> [ADMISSIBLE AND CONSISTENT] (?)

(LRTA* -> (...)) ← NOT DONE IN CLASS

2.7 Optimal Strategies in Non-Deterministic Environments

- **Hypergraph**

A Directed Hypergraph is a pair (V, E) where V = set of vertices and $E \subseteq V \times 2^V \setminus \{\emptyset\}$ = set of hyperedges (es: $(\underbrace{v}_{s}, \underbrace{\{v_1, v_2\}}_{K'}), \ldots$)

$\to$ state s, possible "successors set" K'

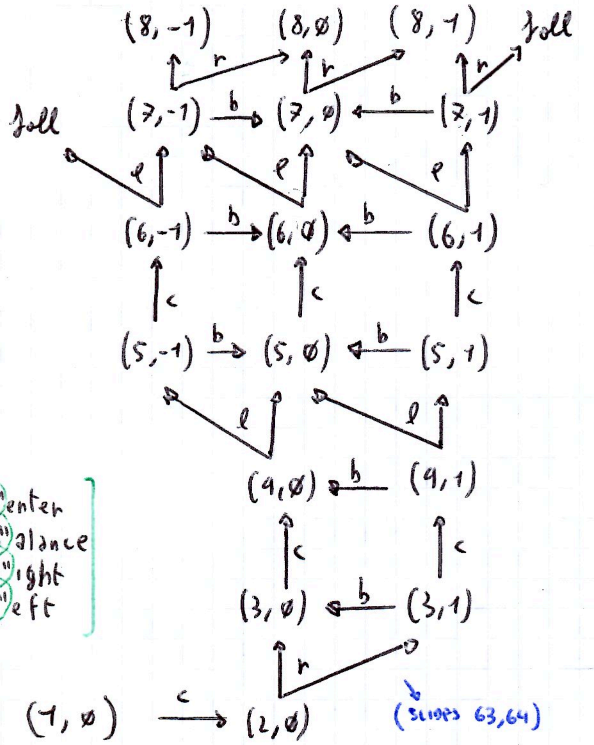
- **Strategy**

(... same as before but:) $f : V^* \to 2^V$

- **Outcomes**

(Same as before but:) a strategy may have several outcomes, and the outcome is not anymore a single path but a TREE.

- **Solve Algorithm**



(slides 63,64)

1°) G is $(8, \{-1, \emptyset, 1\}) \to$ put in W

2°) From my Waiting list W I'll pop a state "s" $\to (8,1)$. My situation is:

In waiting list: $(8,-1)$ W TRUE; $(8,\emptyset)$ W TRUE; $(8,+1)$ TRUE $\leftarrow$ (R[ ] value)

for this one look for predecessors $\to (7,\emptyset)$ and $(7,1)$

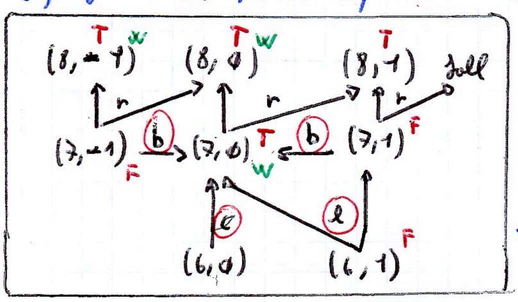
3°) I start with $(7,1)$:

$(8,1)$ TRUE, $(7,1) \xrightarrow{r}$ fall $\to$ are all the targets of (r) winning? NO $\to$ $R[(7,1)] = false$ (as it was before)

4°) Now I check $(7,\emptyset)$:

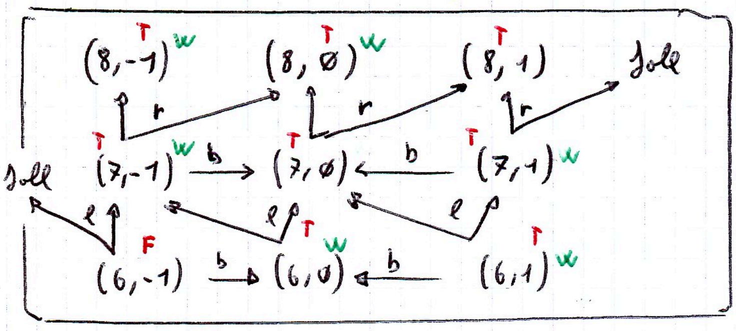
$(8,\emptyset)$ T, $(8,1)$ T, $(7,\emptyset)$ $\to$ all true! $R[(7,\emptyset)] = true$ and $(7,\emptyset) \to W$

So, after these 4 steps my situation is:



$\to$ I'll do the same for $(7,\emptyset)$ $\to$ the winning arrows are highlighted with ◯ $\to$ $(7,1)$ now is true because from $(7,1)$ "r" is not a good option but "b" yes, so it becomes true and becomes true also $(6,1)$ with his "l"

So, after the (7,∅) analysis, we have:



→ Now, if i'll pop (8,∅) or (8,-1) from W, all of their predecessors are already winning → NOTHING TO DO, continue with (7,1), (7,-1), (6,1) ...

- ALGORITHM:

INPUT: $s_\emptyset$ // source state/root → (1,∅)
G // target state → (8, {-1,∅,1})

OUTPUT: r // is G reachable from $s_\emptyset$?
strat // the strategy

$\forall s'$, strat[s'] ← ⊥ {VOID SYMBOL} // NO STRATEGY (INITIALIZATION) WHEN STARTS

r ← false {"IS s' REACHABLE"? → R[ARRAY OF REACHABILITY]}

$\forall s' \in G$, R[s'] ← true // all the targets are true = Reachable

$\forall s' \notin G$, R[s] ← false // all the other states are initially false

W ← G // put the target states in the waiting list

while W ≠ ∅ and not R[$s_\emptyset$]:

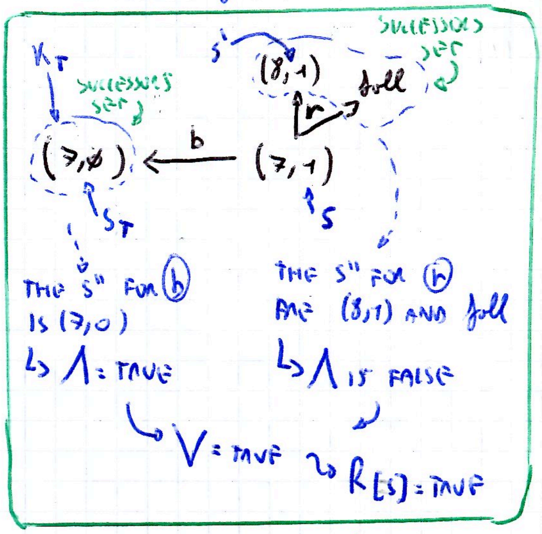
  s' ← next(W) // Pop the next state from W (could be LIFO, FIFO, ecc...)

  foreach s such that not R[s] // foreach state "false" s
    ↳ and $\exists (s,K) \in E$ with $s' \in K$: // that is a ...essor of s'

    $R[s] \leftarrow \bigvee_{(s,K') \in E} \left(\bigwedge_{s'' \in K'} R[s''] \right)$ LIKE BEFORE

(LOGICAL OR FOR EACH POSSIBLE SUCCESSORS SET K' OF s)

(LOGICAL AND FOR EACH SUCCESSOR IN THE RESPECTIVE SUCCESSORS SET K')



    if R[s]:

      W ← W ∪ {s} // put the true s in W

      strat[s] ← K_T s.t $\bigwedge_{s_T \in K_T} R[s_T]$

PUT THE "GOOD" SUCCESSORS SET, IN THE CASE SEEN, "(b)" FROM (7,1), CONSTITUTED ONLY BY (7,∅)

r ← R[$s_\emptyset$]

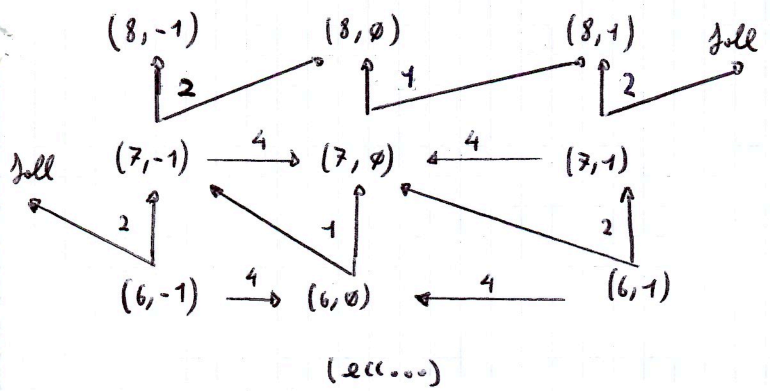
s' IS THE STATE POPPED OUT FROM W
s IS THE NODE THAT IS TESTED
K' IS THE INDEX FOR THE ITERATION THROUGH ALL THE SUCCESSORS SETS
s'' IS THE INDEX FOR THE ITERATION THROUGH ALL THE STATES IN K'

(I START WITH (8,1) → s')

PREDECESSOR s IS (7,1)
I LOOP THROUGH THE K' SUCCESSORS SET AN I'LL DO THE LOGICAL OPERATION.
THEN THE ONE WHICH HAS ALL TRUE VALUES (K_T) I'LL PUT IT IN THE STRATEGY

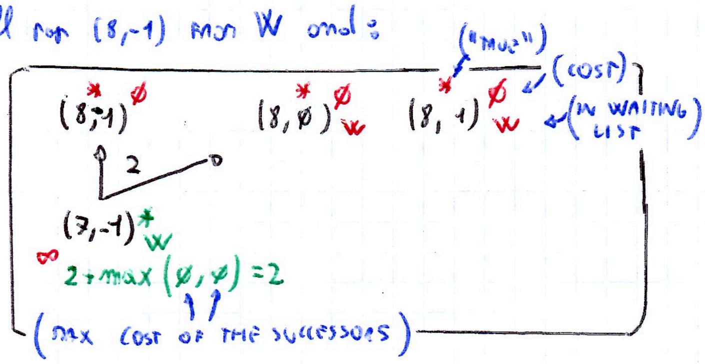
• Knuth Extension of Dijkstra's Algorithm

In the previous graph we'll add the costs in seconds of each moves: $\left[\begin{array}{l} r \\ l \\ c \end{array}\right\} 1 \text{ or } 2 \;\; b = 4 \Big]$ ← see slide 63



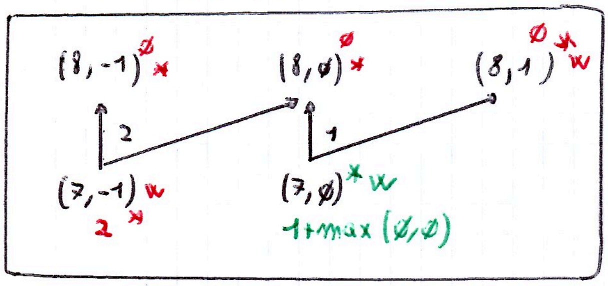
The steps are:

•) I'll pop (8,-1) from W and:

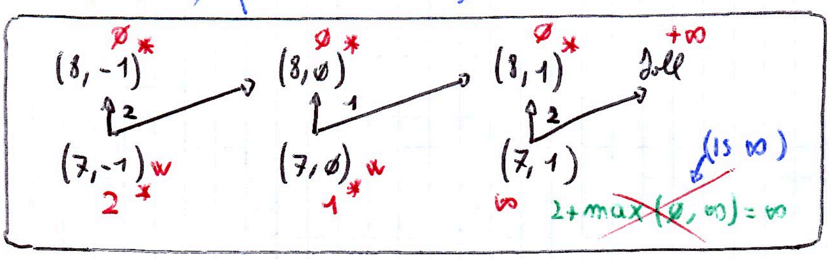


RED → old values
GREEN → new values

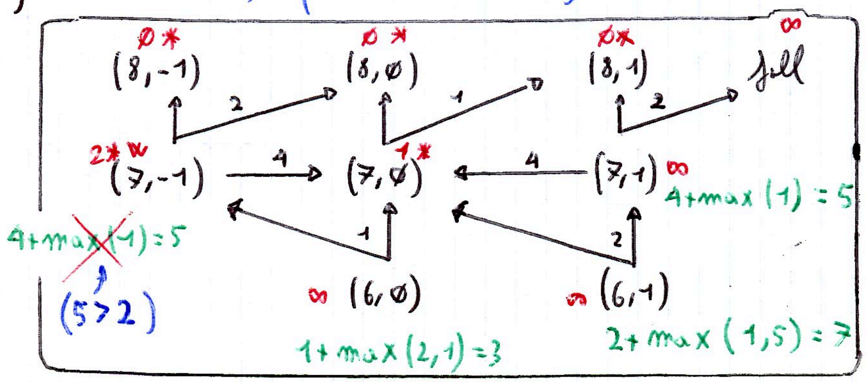
•) I'll not pop (7,1) cause cost 2 > cost ∅ of the others states in W
↳ I'll pop (8,∅):



•) I'll pop (8,1) (less cost in W):



•) I'll pop (7,∅) (less cost in W):



(Only for K' = (7,∅), the {(8,1), fall} set is not considered here, as we seen before)

→ And so on...

2.2 Partially Observable Non-Deterministic Environments

The evolution of the system is perceived through an OBSERVATION FUNCTION mapping states to a finite set of OBSERVATIONS $\mathcal{O}$.

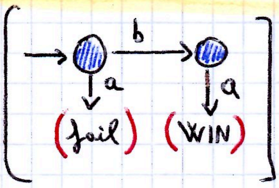
- **Strategy**

O. S. is a function $f: \mathcal{O}(R) \to \Sigma$ s.t. $\forall \sigma, \sigma' \in V^*$ s.t. $\mathcal{O}(\sigma) = \mathcal{O}(\sigma')$ $\exists$ an hyperedge labeled by $f(\sigma)$ from last (σ) $\Leftrightarrow$ $\exists$ one from last (σ')

- **Outcome**

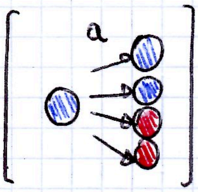
The Outcome (v, f) of a strategy "f" from vertex "v" is the set of runs inductively defined by:
-> $v \in$ Outcome (v, f)
-> if $\sigma \in$ outcome $(v, f) \Rightarrow \left[\sigma \cdot v' \in \text{Outcome}(v, f) \Leftrightarrow v' \in f(\mathcal{O}(\sigma)) \right)$

- **State Estimation (Filtering)**



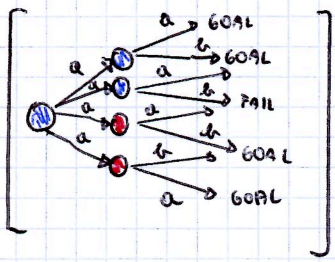
-> We observe only red/blue. STRATEGY: "The first time that I observe blue go (b), the second time go (a)"

↳ NON POSITIONAL STRATEGY! (-> POSITIONAL: [blue] are all the same, so I'll go (a) -> FAIL)



-> If I do (a) and I see blue, I FILTER the initial prediction (4 STATES) with an OBSERVATION (-> 2 STATES)

So, if we imagine a next step:

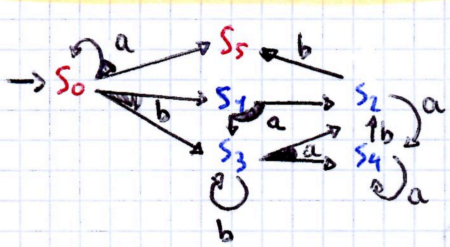


-> I'll do (a), then, if "Blue" ill do (a), if "Red" ill do (b)

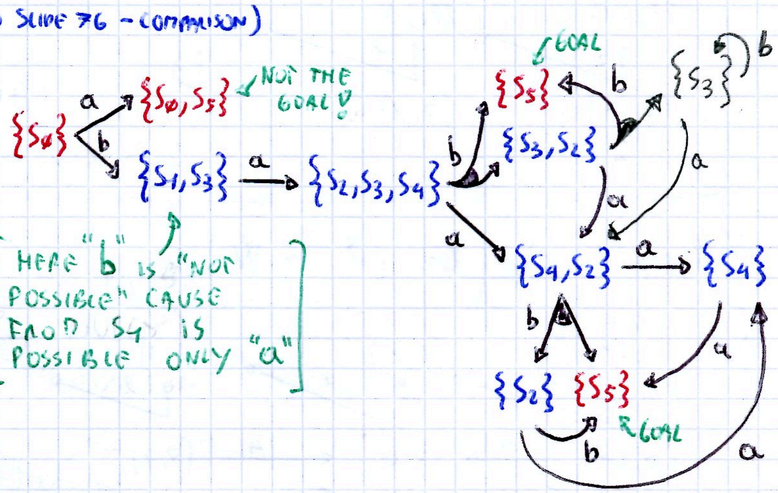
↳ $\left[a \overset{\text{BLUE}}{\to} a, \; a \overset{\text{RED}}{\to} b \right]$ FILTERING

We build INFORMATION SETS (BELIEF STATES)

- **Build a Belief Hypergraph** ->(SEE ALSO SLIDE 76 - COMPARISON)



BELIEF -> HYPERGRAPH



HERE "b" IS "NOT POSSIBLE" CAUSE FROM S4 IS POSSIBLE ONLY "a"

FINISH TO COPY FROM PAGE (3) EXES

3.1 Uncertain (Probabilistic) Environments

• Utility of Morgenstern and Von Neumann (1944)

Preferences: (for agent α) $A \preceq B$ → agent α weakly prefers outcome B over A

PROBABILISTIC OUTCOMES: (example: [" $R_1 = 10€$ if heads " / " $R_2 = ∅€$ if tails "] → $A = \underbrace{50\%}_{p_1} \cdot R_1 + \underbrace{50\%}_{p_2} R_2 = 5$)

Utility is a function s.t.:
1) $U(A) \leq U(B) \iff A \preceq B$
2) $U(p_1 R_1 + \dots + p_n R_n) = p_1 U(R_1) + \dots + p_n U(R_n)$

→ example: (SLIDE 82)

[A: "You have $P = 80\%$ of winning 4K€"
B: "You have $P = 100\%$ of winning 3K€"] → we normally prefer B ($A \leq B$)

Is the actual value of € a good utility function?

$$\left[\begin{array}{ll} A: & 0{,}8 \cdot \overbrace{4000}^{U(R_1)=R_1} + 0{,}2 + \overbrace{\emptyset}^{U(R_2)=R_2} = \underset{U}{3200} \\ B: & 1 \cdot 3000 + 0 \cdot 0 = \underset{U}{3000} \end{array}\right]$$

$3200 > 3000$ → "A is better" → [$U(R_i) = R_i$ IS NOT A GOOD UTILITY FUNCTION]

• Markov Decision Process

A **MDP** is a tuple $(S, s_\emptyset, A, P, R, \gamma)$ → ["IS AN HYPERGRAPH BUT WITH PROBABILITIES" AND REWARDS]

- S: (FINITE SET OF STATES)
- $s_\emptyset$: (INITIAL STATE)
- A: (FINITE SET OF ACTIONS)
- P: ($P: S \times A \to Dist(S)$ IS THE TRANSITION FUNCTION → PROBABILITIES)
- R: ($R: S \times A \times S \to \mathbb{R}$ IS THE REWARD FUNCTION)
- γ: (DISCOUNT FACTOR) → (PG 649 BOOK)

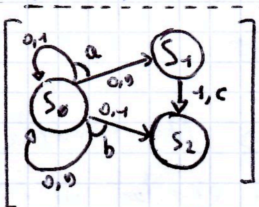
[→ In the "BALANCING ROBOT" example (→ SLIDE 85, PG 3)
we just add a REWARD of 1.0 on STATES $(8, \{-1, 0, +1\})$
and a REWARD of -0.1 on all the other states
with a 10% chance of getting unbalanced] → (GRAPH AT SLIDE 86)

• Strategy

A Strategy / Policy / Scheduler in a MDP is a function $\pi : S^* \to A$

MEMORYLESS STRATEGIES are sufficient ONLY in a INFINITE HORIZON (SLIDE 87)

! If only two moves are accepted:



MEMORYLESS: ["aa" STRATEGY: $s_\emptyset \xrightarrow{a} s_1 \xrightarrow{c} s_2$ → $0.9 \cdot 1$ PROBABILITY OF ARRIVING IN S_2 (I CANNOT ARRIVE TO S_2 WITH AN a IN A TURN)
$s_\emptyset \xrightarrow{a} s_\emptyset \xrightarrow{a}$ ✗ → ∅ " " " " " " } TOTAL: 0.9

"bb" STRATEGY: $s_\emptyset \xrightarrow{b} s_2$ → 0.1
$s_\emptyset \xrightarrow{b} s_0 \xrightarrow{b} s_2$ → $0.9 \cdot 0.1$ } TOTAL: $0{,}1 + 0{,}09 = 0{,}19$]

WITH MEMORY: ["ba" STRATEGY: $s_0 \xrightarrow{b} s_2$ → 0,1
$s_0 \xrightarrow{b} s_0 \xrightarrow{a}$ ✗ → 0 } → TOTAL: 0.1

"ab" STRATEGY: $s_\emptyset \xrightarrow{a} s_1 \xrightarrow{c} s_2$ → $0.9 \cdot 1$
$s_\emptyset \xrightarrow{a} s_0 \longrightarrow s_2$ → $0.1 \cdot 0.1$ } TOTAL: $0{,}9 + 0.01 = 0{,}91$ ← BEST!

Optimization in Probabilistic Environments

PG. 649 BOOK

[FINITE HORIZONS -> NONSTATIONARY OPTIMAL POLICIES
↳ WE WILL DEAL HERE WITH INFINITE HORIZON -> STATIONARY OPTIMAL POLICIES]

No "good state" or "winning" -> MAXIMISING THE REWARDS

For a single-state sequence (lenght = 1) -> $U(s) = R_s$ ← REWARD (SIMPLEST CASE)

For SEQUENCES of states ($l > 1$) Utility is computed on the DIFFERING PART (PREFERENCES STATIONARITY) [SEE PG 649 BOOK]

↳ we eliminate all the COMMON PREFIXES ↳ $\left[\begin{array}{c} U(s_0, s_1, \dots s_n) < U(s_0, s_1', \dots, s_n') \\ \Updownarrow \\ U(s_1, \dots, s_n) < U(s_1', \dots, s_n') \end{array}\right]$

So, for $\gamma \in [0,1]$: (PG 649, BOOK)

↳ $\left[U(s_0, s_1, \dots, s_n) = U(s_0) + \gamma\, U(s_1, \dots, s_n)\right] \rightarrow U(s_0, \dots, s_n) = \sum_{t=0}^{\infty} \gamma^t R(s_t)$

(THE FIRST REWARD IS VERY IMPORTANT BECAUSE IT'S NOT MULTIPLIED BY γ)

With discounted rewards ($\gamma < 1$) the utility of an infinite sequence is finite:

$$\left[U(s_0, s_1, \dots, s_\infty) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \le \underbrace{\sum_{t=0}^{\infty} \gamma^t R_{max}}_{\text{(GEOMETRIC SERIE)}} = \frac{R_{max}}{1-\gamma}\right]$$

Otherwise ($\gamma = 1$) -> PROPER POLICIES or AVERAGE REWARDS (see PG 650 BOOK I.A.)

Computing Optimal Strategy

Having decided that $U(s_i) = R(s_i)$ (SIMPLEST CASE UTILITY = REWARD), we can compare policies/strategies $\{\pi\}$ by comparing the EXPECTED UTILITIES obtained when executed.

So, given: -> "s" some initial state
↳ "S_t" the state that the agent reaches at TIME t ($S_0 = s$) when executing π

We have:
$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$
-> EXPECTATION is w.r.t. the probability distribution over STATE SEQUENCES determined by "s" and "π"

So, the optimal policy will be: $\pi^*(s) = \underset{\pi}{\operatorname{argmax}} \left[U^{\pi}(s)\right]$

[USING DISCOUNTED UTILITIES ($\gamma < 1$) WITH INFINITE HORIZONS IMPLY THAT THE OPTIMAL POLICY IS INDEPENDENT OF THE STARTING STATE (OFCOURSE THE ACTION SEQUENCE WON'T BE INDEPENDENT)
↳ [SO WE CAN WRITE ONLY "π^*"]] ! (PG. 651 BOOK)

The utility function $U(s)$ allows the agent to select action by using the principle of maximum expected utility, that is, choose the action that maximize the expected utility of the subsequent state: ↳

$$\left[\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a)\, U(s')\right]$$
-> [2 ALGORITHMS:
- VALUE ITERATION
- POLICY ITERATION]

• Value Iteration

MARTIN S. 6

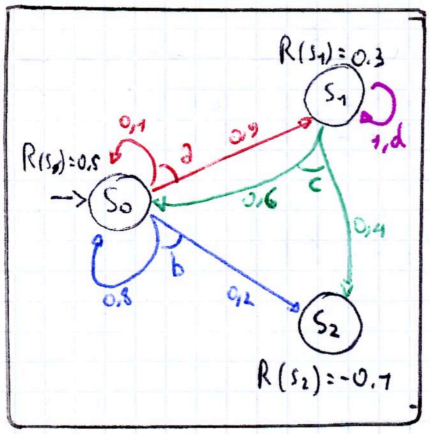
BELLMAN EQUATION for MDP }: $$U^{\pi^*}(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' | a, s)\, U^{\pi^*}(s')$$ → [" THE UTILITY OF A STATE IS THE IMMEDIATE REWARD FOR THAT STATE PLUS THE EXPECTED UTILITY OF THE NEXT STATE, ASSUMING THAT THE AGENT CHOOSES THE OPTIMAL ACTION "] → PG. 652 BOOK

This is implied by the $\pi^*(s)$ formula seen before.
The utilities of the states ($U^\pi(s) = E[\Sigma \dots]$) are solution of this eq.

These equations are non-linear, but we can solve them by iteration (up to an error ε):

ALGORITHM
1. $U_\emptyset(s) = \emptyset$
2. $U_{i+1}(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' | a, s)\, U_i(s')$ ← THIS ITERATION IS THE BELLMAN UPDATE $B(U_i)$
3. Until, $\forall s$, $|U_{i+1}(s) - U_i(s)| < \frac{\varepsilon(1-\gamma)}{\gamma}$

↳ EXAMPLE:



∅) $\underline{U_\emptyset(s_\emptyset)} = \emptyset$ $\quad \underline{U_\emptyset(s_1)} = \emptyset$ $\quad \underline{U_\emptyset(s_2)} = \emptyset$

1) $\underline{U_1(s_\emptyset)} = 0{,}5 + \gamma \cdot \max \Big[P(s_\emptyset | a, s_\emptyset)\, U_\emptyset(s_\emptyset) + P(s_1 | a, s_0)\, U_\emptyset(s_\emptyset),$
$P(s_\emptyset | b, s_\emptyset)\, U_\emptyset(s_\emptyset) + P(s_2 | b, s_\emptyset)\, U_\emptyset(s_2) \Big] =$

$= 0{,}5 + \gamma \cdot \max \left[0{,}1 \cdot \emptyset + 0{,}9 \cdot \emptyset,\ 0{,}8 \cdot \emptyset + 0{,}2 \cdot \emptyset \right] = 0{,}5$

$\underline{U_1(s_1)} = 0{,}3$ $\qquad \underline{U_1(s_2)} = -0{,}1$

[$\sum_{s' \in S}$ ⇒ [We get the PROBABILITY OUTCOME FOR a given action "a"]
↓
$\max_{a \in A}$ ⇒ [We get the action "a" that maximizes the outcome]] ← HOW THE ALGORITHM WORKS
↳ (ITERATIONS TO HAVE MORE ACCURACY)

2) $\underline{U_2(s_\emptyset)} = 0{,}5 + \gamma \cdot \max \Big[\underbrace{0{,}1 \cdot 0{,}5 + 0{,}9 \cdot 0{,}3}_{0{,}32},\ \underbrace{0{,}8 \cdot 0{,}5 + 0{,}2 \cdot (-0{,}1)}_{0{,}38} \Big] = 0{,}88$ ← [ASSUMING $\gamma = 1$]

action "b" maximizes the outcome in $S_\emptyset$

$\underline{U_2(s_1)} = 0{,}3 + \gamma \cdot \max \Big[\underbrace{0{,}6 \cdot 0{,}5 + 0{,}4 \cdot (-0{,}1)}_{0{,}26},\ \underbrace{1 \cdot 0{,}3}_{0{,}3} \Big] = 0{,}6$; $\underline{U_2(s_3)} = -0{,}1$

action "d" maximizes the outcome in S_1

If we stop the iterations in $i = 2$ → we would chose "b" in $S_\emptyset$ and "d" in S_1

CONVERGENCE OF VALUE ITERATION:

The Bellman update $B(U_i) = U_{i+1}$ is a CONTRACTION (by a γ factor) for the MAX NORM $\|U\| = \max|U(s)|$

$$\|B(U) - B(U')\| \leq \gamma \|U - U'\|$$

The utilities are bounded by $(R_{max}/1-\gamma)$ so:

AFTER N ITERATIONS

INITIAL ERROR: $\|U_0 - U^*\| = \|U^*\| \leq R_{max}/1-\gamma$

N-TH ERROR: $\|U_n - U^*\| \leq \gamma^n \|U_0 - U^*\| \leq \gamma^n (R_{max}/1-\gamma)$

↳ So if we wanna ERROR $< \varepsilon \Rightarrow n = \left\lceil \log\left[R_{max}/\varepsilon(1-\gamma)\right] / \log(1/\gamma) \right\rceil$

(We can also proof that $\|U_{i+1} - U_i\| \leq \varepsilon \frac{1-\gamma}{\gamma} \Rightarrow \|U_{i+1} - U^*\| < \varepsilon$)

$$\left[\|U_i - U^*\| \leq \varepsilon \Rightarrow \|U^{\pi_i} - U^*\| < \varepsilon \frac{1-\gamma}{\gamma} \right]$$ ← (CONVERGENCE OF VALUE ITERATION - POLICY LOSS)

(π_i: the strategy defined using U_i (might not to be optimal))

- **POLICY ITERATION** → (PG 656-6 ,7 BOOK)

In practice, the policy becomes optimal BEFORE the VALUE ITERATION CONVERGES

↓ ALTERNATIVE APPROACH

ALGORITHM

1. Fix an arbitrary policy π_0
2. Compute the corresponding utilities U^{π_0} [PG ⑤ REAR]
3. Compute the MEU ← MAXIMUM EXPECTED UTILITY strategy π_1 maximizing U^{π_0} with one-step look-ahead
4. Repeat until $\pi_{i+1} = \pi_i$

↳ VALUE ITERATION: $B(U_i) = U_{i+1} = R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in S} P(s'|a,s) U_i(s')$

↳ POLICY ITERATION: $\left[B(U_i) = U_{i+1} = R(s) + \gamma \sum_{s' \in S} P(s'|a,s) U_i(s') \right]$ ← (THE POINTS 2. & 3.)

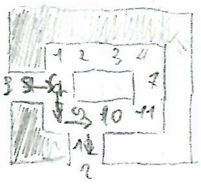
↓

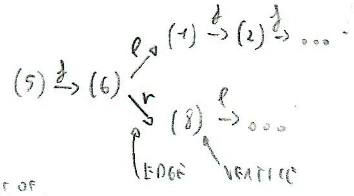
It simplifies the BELLMAN EQUATION! (no max → IS A LINEAR EQUATION → EASIER) (CAN BE SOLVED IN $O(n^3)$ WITH LINEAR ALGEBRA)

And if we substitute STEP 4. with "4. Repeat for n steps" → MODIFIED POLICY ITERATION ! → (NOT EXACT BUT MORE EFFICIENT)

SUMMARY

DETERMINISTIC ENVIRONMENT





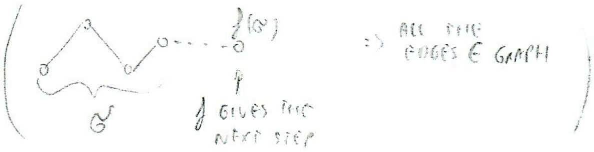
DIRECT GRAPH : (V, E) (set of vertices)

(EDGES) $\to E \subseteq V \times V$

POSITIONAL STRATEGY : $f : V^* \to V$ (set of finite sequences of elements of V)

s.t. $\forall \sigma \in V^*$, $(\text{last}(\sigma), f(\sigma)) \in E$

OUTCOME OF f = SEQUENCE OF VERTICES PRODUCED BY THE AGENT WHEN IT APPLIES A STRATEGY f FROM VERTEX "v" — outcome(v, f) !



=> ALL THE EDGES $\in$ GRAPH

FULLY OBSERV. $\wedge$ ONLY ONE AGENT $\wedge$ DETERMINISM => OUTCOME IS A SINGLE PATH

A SEQUENCE σ IS MAXIMAL IN A VERTEX SET V_1 IF IS NOT A PREFIX OF ANOTHER SEQUENCE $\in V_1$ OR IF IT IS INFINITE (WHEN $V_1 = V^*$)

OF ALL THE $\sigma \in$ OUTCOME OF f, IF WE TOOK ONLY THE MAXIMAL ONES σ_{max} →

$$\text{max outcome}(v, f) = \{\sigma_{max}\}$$

OBJECTIVE $\triangleq X$ = IS A SET OF VERTEX SEQUENCES

f IS WINNING FOR (v, X) $\Leftrightarrow$ $\text{Max Outcome}(v, X) \subseteq X$
(starting vertex, objective)

⇓

FOR THE MAZE: WE CAN USE AS f THE DFS, BFS, f_{DFS}, f_{BFS}... TO SEARCH FOR A PATH

FOR OPTIMAL PATH: DIJKSTRA, A^* → WE FIND

$$5 \xrightarrow{f} 6 \xrightarrow{r} 8 \xrightarrow{l} 9 \xrightarrow{r} 10$$

NON-DETERMINISTIC

→ WE ADD TO THE MAZE THE UNBALANCING

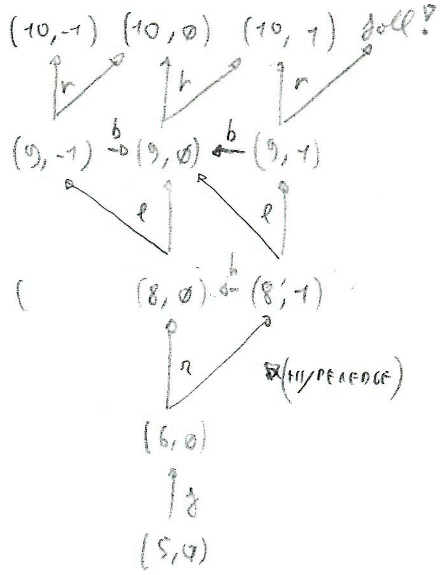
$(-1, \emptyset, 1)$
LEFT, BALANCED, RIGHT

↓

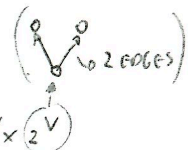
WE USE THE WINNING OPTIMAL STRATEGY FROM BEFORE

$$(5) \xrightarrow{f} (6) \xrightarrow{r} (8) \xrightarrow{l} (9) \xrightarrow{r} (10)$$

BUT ↓



DIRECTED HYPERGRAPH : (V, E)
$\hookrightarrow E \subseteq V \times 2^V$



STRATEGY : $f : V^* \to 2^V$

THE OUTCOME IS A TREE (THE ONE IN FIGURE)

↓

WITH "SOLVE ALGORITHM" I ELIMINATE THE FAILING POSSIBILITIES FROM THE TREE → I HAVE AGAIN A WINNING STRATEGY

IF I PUT WEIGHTS ON HYPERGRAPHS
↳ OPTIMAL STRATEGY : KNUTH'S EXTENSION OF DIJKSTRA

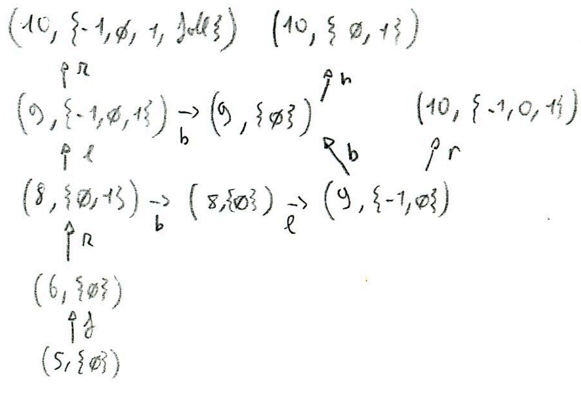
PARTIALLY OBSERVABLE

OBSERVATION: $\mathcal{O}(v) : V \to O_{BS}$

STRATEGY: $f : \mathcal{O}(V^*) \to \Sigma$ (set of hyperedges)

$$\left[\text{last}(\sigma) \begin{cases} f(\mathcal{O}(\sigma)) \\ f(\mathcal{O}(\sigma')) \end{cases} \Leftarrow \mathcal{O}(\sigma) = \mathcal{O}(\sigma')\right]$$

⇒ [BELIEF HYPERGRAPH] →



PROBABILISTIC ENVIRONMENTS

$$\text{MDP}: \Big(\underset{\text{STATES}}{S}, \; s_\emptyset, \; \underset{\text{ACTIONS}}{A}, \; \underset{\text{TRANSITION PROBABILITIES}}{P}, \; \overset{\text{REWARD}}{R^{\delta}}, \; \underset{\text{DISCOUNT FACTOR}}{\gamma} \Big)$$

↳ IS A HYPERGRAPH WITH PROBABILITIES AND REWARD

STRATEGY → POLICY : $\pi : S^{*} \to A$

NO OUTCOME → UTILITY OF POLICY

$$U^{\pi}(s) = E\left[\sum_{t=\emptyset}^{\infty} \gamma^{t} R(S_t) \right]$$

FIND THE OPTIMAL WITH VALUE/POLICY ITERATION

3.2 Partially Observable Probabilistic Environments

• Partially Observable MDP

Observation function: $S \rightarrow Dist(O)$ -> associates to each state a Prob. Distribution of Observations

Markov Assumption: $o \in O$ depends only on the current state ⇘ we have probabilities $P(o, s)$

↳ difficult! [will focus on...]

• Hidden Markov Chains

PoMDP where [is possible only ONE action at each time.] ↙ DEFINITION OF MARKOV CHAIN

HIDDEN WE WANT

- FILTERING: in what states are we likely to be now
- PREDITION: in what states are we likely to be in the next steps
- OBSERV. LIKELYHOOD: how likely are the observations made?
- SMOOTHING: In what states were we likely to be some steps before
- MOST LIKELY EXPLANATION: what is the most likely sequence of states up to now?

BUT BEFORE...

• Probability Summary Reminder

$$\left(\begin{array}{l} \text{ALEATORY VARIABLE } X \rightarrow P(X = x),\ x \text{ "value" } (\text{or } P(x)) \rightarrow P(x, y \mid z) \left(\begin{array}{l}\text{KNOWING THAT } x, y \\ \text{PROB. OF } z\end{array}\right) \\ \mathbb{P}(X) = \text{PROBABILITY VECTOR (contains all } x) \rightarrow \mathbb{P}(X, Y \mid Z) \end{array}\right) \text{ NOTATIONS}$$

• PRODUCT RULE: $P(x, y) = P(x \mid y)\, P(y)$ • MARGINALIZATION (SUMMING OUT) $\mathbb{P}(Y) = \sum_{z \in Z} \mathbb{P}(Y, z)$

• CONDITIONING: $\mathbb{P}(Y) = \sum_{z \in Z} \mathbb{P}(Y \mid z)\, P(z)$

$$[X, Y \text{ independent}] \Rightarrow [\mathbb{P}(X \mid Y) = \mathbb{P}(X)] \wedge [\mathbb{P}(X, Y) = \mathbb{P}(X)\mathbb{P}(Y)]$$

$$[X, Y \text{ independent given } Z] \Rightarrow [\mathbb{P}(X, Y \mid Z) = \mathbb{P}(X \mid Z)\, \mathbb{P}(Y \mid Z)]$$

• BAYES RULE: $\mathbb{P}(\underset{\text{CAUSE}}{Y} \mid X, \underset{\text{EFFECT}}{e}) = \mathbb{P}(X \mid Y, e)\, \mathbb{P}(Y \mid e) / \mathbb{P}(X \mid e)$ (SWITCH BETWEEN CAUSALITY AND DIAGNOSTIC)

• Forward Algorithm

X_t is X at time t ("t-th step") -> OBSERVE AT EACH STEP => $O_{1:t} = \{ o_1, \dots, o_t \}$

↳ How to compute $P(X_t, O_{1:t})$? ⟨ ENUMERATE ALL SEQUENCES AND SUM THE PROBABILITIES (SLOW) / FORWARD ALGORITHM ⟩

$$P(X_t, O_{1:t}) \underset{\substack{\text{SUMMING} \\ \text{OUT}}}{=} \sum_{x_{t-1}} \mathbb{P}(X_t, x_{t-1}, O_{1:t}) \underset{\substack{\text{PRODUCT} \\ \text{RULE}}}{=} \sum_{x_{t-1}} \mathbb{P}(o_t \mid X, \underbrace{\cancel{x_{t-1}, O_{1:t-1}}}_{\text{MARKOV ASSUMPTION}})\, \mathbb{P}(X_t \mid x_{t-1}, \underbrace{\cancel{O_{1:t-1}}}_{\text{MARKOV ASSUMPTION}})\, P(x_{t-1}, O_{1:t-1})$$

$$\hookrightarrow \left(P(X_t, O_{1:t}) = \overbrace{P(o_t \mid X_t)}^{\text{(THIS IS THE OBSERVATION)}} \sum_{x_{t-1}} \mathbb{P}(X_t \mid x_{t-1})\, P(x_{t-1}, O_{1:t-1}) \overset{\wedge}{=} \text{FORWARD}\left[\mathbb{P}(X_{t-1}, O_{1:t-1}), o_t\right] \right)!$$

-> ITERATIVE ALGORITHM!

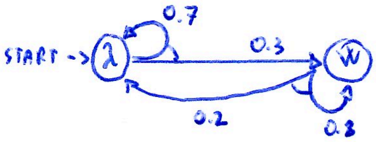
- **MATRIX FORM:**

$$\ell_{1:t} \triangleq P(X_t, \sigma_{1:t}) \; ; \; T_{ij} \triangleq P(X_t = j \mid X_{t-1} = i) \rightarrow \text{TRANSITION PROBABILITY MATRIX } \underline{\underline{T}} \; ; \; \underline{\underline{O}}_t \triangleq \begin{pmatrix} P(O_t, x_1) & & \emptyset \\ & \ddots & \\ \emptyset & & P(O_t, x_m) \end{pmatrix}$$

DIAGONAL MATRIX OF $P(O_t, x_i)$

$$\hookrightarrow \left[\underline{\ell}_{1:t} = \underline{\underline{O}}_t \cdot \underline{\underline{T}}^T \cdot \underline{\ell}_{1:t-1} \right] \rightarrow \text{DISCRETE KALMANN FILTER}$$

- example: (SLIDE 113/220)



$$\begin{array}{c} \\ j \end{array} \quad \begin{array}{cc} \lambda & W \end{array}$$

$$\underline{\underline{T}} = \begin{array}{c} \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix} \end{array} \begin{array}{c} \lambda \\ W \end{array} \Big\} \, i$$

(3 OBSERVATIONS: S, C, P)

WHEN LISTENING (λ) STUDENTS	30% LOOK AT SCREEN (C) 10% SMILES (S) 60% LOOK AT PROF (P)
WHEN WEB-SURFING (W) STUDENTS	30% SMILES (S) 60% LOOK SCREEN (C) 10% LOOK AT PROF (P)

$\rightarrow$

$$\underline{\underline{O}}_s = \begin{array}{c} \lambda \\ W \end{array} \begin{pmatrix} 0.1 & \emptyset \\ \emptyset & 0.3 \end{pmatrix}$$

$$\underline{\underline{O}}_c = \begin{pmatrix} 0.3 & \emptyset \\ \emptyset & 0.6 \end{pmatrix}$$

$$\underline{\underline{O}}_p = \begin{pmatrix} 0.6 & \emptyset \\ \emptyset & 0.1 \end{pmatrix}$$

I'LL DO THE FOLLOWING OBSERVATIONS: $O_{1:t} = \{S, C, P\} \leftarrow (t=3)$

$$\ell_{1:1} = O_s T^T \ell_\emptyset \quad \leftarrow \text{N.B} \left[\ell_\emptyset = \begin{pmatrix} 1 \\ \emptyset \end{pmatrix} \right] ! \; ; \; \ell_{1:2} = O_c T^T \ell_{\emptyset:1} \; ; \; \ell_{1:3} = O_p T^T \ell_{\emptyset:2}$$

$$\ell_{1:1} = \begin{pmatrix} 0{,}07 \\ 0{,}09 \end{pmatrix} \; ; \; \ell_{1:2} = \begin{pmatrix} 0{,}027 \\ 0{,}058 \end{pmatrix} \; ; \; \ell_{1:3} = \begin{pmatrix} 0{,}015 \\ 0{,}005 \end{pmatrix} \leftarrow \text{IN OCTAVE} \rightarrow \text{OK!}$$

$\rightarrow$ TO HAVE THE PROBABILITY: NORMALIZE:

$$F_3 = \frac{\ell_{1:3}}{\text{sum}(\ell_{1:3})} = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix} \begin{array}{l} \rightarrow P(\lambda, S_1, C_2, P_3) \\ \rightarrow P(W, S_1, C_2, P_3) \end{array}$$

FILTERING

(NOW WE CAN START)

- **FILTERING**

(FILTERING: in what states are we likely to be now?)
(in the last passage of the example)

$$\mathbb{P}(X_t \mid \sigma_{1:t}) = \mathbb{P}(X_t, \sigma_{1:t}) / \mathbb{P}(\sigma_{1:t}) \xrightarrow[\text{FORM}]{\text{MATRIX}} \left[\underline{F}_{1:t} = \frac{\underline{\underline{O}}_t \underline{\underline{T}}^T \underline{\ell}_{1:t-1}}{\text{sum}(\underline{\ell}_{1:t})} = \frac{\underline{\ell}_{1:t}}{\text{sum}(\underline{\ell}_{1:t})} \right]$$

- **PREDICTION**

(PREDICTION: in what states are we likely to be in the next steps?)

$$\mathbb{P}(X_{t+1} \mid \sigma_{1:t}) \overset{\text{SUMMING OUT}}{=} \sum_{x_t} \mathbb{P}(X_{t+1} \mid x_t) P(x_t \mid \sigma_{1:t}) \xrightarrow[\text{FORM}]{\text{MATRIX}} \left[\mathbb{P}(X_{t+m} \mid \sigma_{1:t}) = (\underline{\underline{T}}^T)^m \underline{F}_{1:t} \right]$$

("t+1" OBSERVING TILL "t")

- for the example:

CONVERGES TO 40% AND 60% IF WE STOP

(We computed for S_1, C_2, P_3) $\rightarrow$ FOR $t = 13 \rightarrow \underline{F}_{1:13} = \begin{pmatrix} 0{,}40034 \\ 0{,}59996 \end{pmatrix}$, FOR $t = 1003 \rightarrow \underline{F}_{1:1003} = \begin{pmatrix} 0{,}4000 \\ 0{,}6000 \end{pmatrix}$

ARTIN
S. 8

- **Observation Likelihood**

(how likely are the observations made?)

$$P(O_{1:t}) = \sum_{X_t} P(X_t, O_{1:t}) \xrightarrow{\text{MATRICIALLY}} \left[P(O_{1:t}) = \underline{1}^T \underline{l}_{1:t} \right]$$

- for the example:

for $O_{1:3} = \{S_1, C_2, P_3\}$ → ASK FOR THE PROF'S CODE

- **Smoothing**

(in which states we were likely to be some steps before?)

We want to compute $P(X_{k<t} \mid O_{1:t})$ ← (GIVEN SOME NEW EVIDENCES WE REFINE A PREVIOUS ESTIMATION)

(PROOF ON SLIDES)

$$\mathbb{P}(X_k \mid O_{1:t}) = \frac{1}{P(O_{k+1:t})} \underline{F}_{1:k} \overset{\text{(HADAMARD PRODUCT)}}{\circ} \underbrace{\mathbb{P}(O_{k+1:t} \mid X_k)}_{(1)}$$

$$\mathbb{P}(O_{k+1} \mid X_k) = \sum_{X_{k+1}} P(O_{k+1} \mid X_{k+1}) P(O_{k+2:t} \mid X_{k+1}) \mathbb{P}(X_{k+1} \mid X_k) \quad \text{MATRICIALLY}$$

$$\underline{B}_{i:t} \triangleq \mathbb{P}(O_{i:t} \mid X_k) \triangleq \text{BACKWARD}(\underline{B}_{i+1:t}, O_i) \quad \text{with } \underline{B}_{t+1:t} = \underline{1}$$

✓ THE PROBABILITY OF OBSERVING AN EMPTY SEQUENCE IS ONE IN EACH STATE

$$\left[\underline{B}_{i:t} = \underline{\underline{T}} \cdot \underline{\underline{O}}_i \cdot \underline{B}_{i+1:t} \right] \quad (1) \text{ OK}$$

So: $$\mathbb{P}(X_k \mid O_{1:t}) = \frac{1}{P(O_{k+1:t})} \underline{F}_{1:k} \circ \underline{B}_{k+1:t} \Rightarrow \underline{S}_{1:k:t} \triangleq \underline{F}_{1:k} \circ \underline{B}_{k+1:t}$$

$$\left[\mathbb{P}(X_k \mid O_{1:t}) = \frac{1}{\underline{1}^T \underline{S}_{1:k:t}} \cdot \underline{S}_{1:k:t} \right] \checkmark$$

- for the example:

After the filtering (compute $\underline{F}$ vector) we want to smooth the computation

SEQUENCE: $S_1\ C_2\ P_3\ S_4\ C_5\ P_6$ → FILTERING ON FIRST 3 → $(k=3)$ → done before

So: $\underline{B}_7 = \text{ones}(2,1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$\underline{B}_6 = \underline{\underline{T}} * \underline{\underline{O}}_P * \underline{B}_7 = \begin{pmatrix} 0{,}45 \\ 0{,}2 \end{pmatrix}$

$\underline{B}_5 = \underline{\underline{T}} * \underline{\underline{O}}_P * \underline{B}_6 = \begin{pmatrix} 0{,}1305 \\ 0{,}1230 \end{pmatrix}$ BACKWARD

$\underline{B}_4 = \underline{\underline{T}} * \underline{\underline{O}}_P * \underline{B}_5 = \begin{pmatrix} 0{,}020285 \\ 0{,}032130 \end{pmatrix}$

(SMOOTHED) $\underline{S}$	$\underline{F}$
65%	75%
34%	25%

$\underline{S}_3 = \underline{l}_3 \circ \underline{B}_4$ ← ".*" IN MATLAB

NORMALIZE: $\underline{S}_3 = \underline{S}_3 / \text{sum}(\underline{S}_3) = \begin{pmatrix} 0{,}65263 \\ 0{,}34737 \end{pmatrix}$

- MOST LIKELY EXPLANATION (DECODING)

(what is the most likely sequence of states up to now?)

We want to compute: $\rightarrow$ $\max\limits_{x_1,\dots,x_t} P(X_{1:t} \mid O_{1:t})$ $\rightarrow$ (I'll found the sequence of hidden states with max prob given the observation $O_{1:t}$)

(MOST PROBABLE SEQUENCE)

IF we fix X_t, the MPS is made of:

→ a single transition from X_{t-1} to X_t

→ the most likely path to X_{t-1} given $O_{1:t-1}$ (O_t does not matter cause X_t is fixed)

→ We can compute it with VITERBI ALGORITHM

VITERBI

SLIDE 125/229

4. SUPERVISED LEARNING

• Supervised Learning

We have an UNKNOWN FUNCTION f s.t. $\forall i, y_i = f(\vec{x_i})$ VECTOR GIVING VALUES TO SOME FINITE SET OF FEATURES OR ATTRIBUTES

So we have some GIVEN INPUT-OUTPUT PAIRS $(\vec{x_1}, y_1), \dots, (\vec{x_m}, y_m)$

↳ Supervised learning is finding a FUNCTION h (HYPOTESIS) taken from a chosen HYPOTHESIS SPACE $\mathcal{H}$ that APPROXIMATES f

If output of f is finite → CLASSIFICATION (otherwise is REGRESSION)

• Generalisation vs Overfitting

h should be consistent with GIVEN DATA but also generalize it → (TESTING: K-fold cross validation)

↳ test data should never be used to select HYPERPARAMETERS (this would introduce biases)

[a SupLearn problem is REALISABLE] ⟺ [$f(\vec{x_i}) \in \mathcal{H}$] BUT (if $\mathcal{H}$ BIG) → (complex learning, ↑Prob. of OVERFITTING, complex computations)

- HOW TO CHOOSE THE SIZE OF $\mathcal{H}$:
1) Let's say $\mathcal{H}$ has functions h_i that are polinomial of degree $= n$
2) Split DATASET in TRAINING SET (T_s) & VALIDATION SET (V_s)
3) Then, for increasing values of n, train model with T_s and validate (K fold) with V_s
4) Chose the value of n that has the best validation, then train again with $T_s + V_s$.

• Loss Functions

Measures the QUALITY OF ERRORS (better than the ERROR RATE) ↓

$$\text{Given } y = f(\vec{x}) \text{ and } \hat{y} = h(\vec{x}) \Rightarrow \left[L(\vec{x}, y, \hat{y}) \triangleq U(y, \vec{x}) - U(\hat{y}, \vec{x}) \right]$$

- EXAMPLES:
  - ↳ $L_1(y, \hat{y}) = |y - \hat{y}|$ for real-valued classifiers
  - ↳ $L_2(y, \hat{y}) = (y - \hat{y})^2$ " " " " "
  - ↳ $L_{0/1}(y, \hat{y}) = \begin{cases} 0, & y = \hat{y} \\ 1, & y \neq y \end{cases}$ for discrete-valued classifiers

STIPE@UNIGE.IT

12/05 24/06 → BOSSA TEST

We don't minimize Expected Error Rate but the "Ex~~p~~ected" loss ↳ WE DON'T KNOW THE ASSOCIATED PROBABILITIES → minimize Empirical Loss

$$\text{EMPIRICAL LOSS}: \left[\hat{L}_E(h) = \frac{1}{N} \sum_{(\vec{x}, y) \in E} L(y, h(\vec{x})) \right]$$

- REGULARIZATION OF THE LOSS FUNCTION

We add : $\left[L(\vec{x}, y, \vec{y}) \triangleq U(y, \vec{x}) - U(\vec{y}, \vec{x}) + \lambda \, Reg(h) \right]$ with $\lambda > \emptyset$

$Reg(h)$ -> REGULARIZATION FUNCTION : is bigger when the hypothesis h is more complex

(To find the best λ ? ? ? ?): EARLY STOPPING: stop the training when performance on a VALIDATION SET first becomes worse

FEATURE SELECTION: discard statistically irrelevant attributes of the input

- **P.A.C. LEARNING ALGORITHMS**

Let say that each DATA SAMPLE $(\vec{x_i}, y_i)$ is a value of the random variable E_i and that E_i are independent and identically distributed (I.I.D.)

$$\begin{cases} \mathbb{P}(E_i \mid E_{i-1}, E_{i-2}, \dots) = \mathbb{P}(E_i) \\ \mathbb{P}(E_i) = \mathbb{P}(E_{i-1}) = \mathbb{P}(E_{i-2}) = \dots \end{cases} \text{IID}$$

(A LEARNING ALGORITHM is PROBABLY APPROXIMATELY CORRECT) $\Longleftrightarrow$ ($\exists \delta < 1, \varepsilon > \emptyset$ s.t. for a SAMPLE COMPLEXITY: $\left[\mathbb{P}(\text{ERROR} \leq \varepsilon) = 1 - \delta \right]$)

("for a big enough training set")

- EXAMPLE : BOOLEAN FUNCTION :

(f = BOOLEAN FUNCT) $\to$ ERROR RATE: $err(h) = E\left[L_{0/1} \right]$ -> (we don't know the probabilities here, so we don't know $err(h)$)

(EXPECTED VALUE)

Suppose we have a learning algorithm s.t. $\exists \varepsilon > \emptyset$ s.t. $err \leq \varepsilon$ and suppose that for N-samples an hypothesis h_b is CONSISTENT but has an $err(h_b) > \varepsilon$

The probability of this happening is $(1-\varepsilon)^N$ WHY??

The probability that $\mathcal{H}$ contains such a bad hypothesis h_b is bounded by $|\mathcal{H}|(1-\varepsilon)^N$

And $|\mathcal{H}|(1-\varepsilon)^N \leq \delta$ if we have : $|\mathcal{H}|(1-\varepsilon)^N \leq |\mathcal{H}| e^{-\varepsilon N} \leq \delta$ -> $\left[N \geq \frac{1}{\varepsilon} \left(\log\left(\frac{1}{\delta}\right) + \log\left(|\mathcal{H}|\right) \right) \right]$

CONDITION FOR PAC (?)

- **ENSEMBLE LEARNING**

Not ONE h but more hypotheses (K) and combine their K predictions. If the ERRORS are DISJOINTED we can decrease the error a lot

- BOOSTING:

Reusing the examples in the training DATASET that are MISCLASSIFIED by the hypotesis. They are then duplicated (or given a greater weight) to augment the original training set. The new training set is used to obtain a new h, etc...

- BAGGING:

BOOTSTRAP AGGREGATING (→ BAGGING) consists in generating different training sets by sampling from the original training set. SAMPLING is done UNIFORMLY, and with REPLACEMENT.
Each of those new training sets (→ BOOTSTRAP SAMPLES) are used to produce hypotesis.

- RANDOM SUBSPACE:

Producing many hypotesis using only a subset of the input features for each. These subset are determined randomly for each hypothesis. The number of features can also be random, but is often the same across hypothesis.

4.1 DECISION TREES

- BUILD A TREE

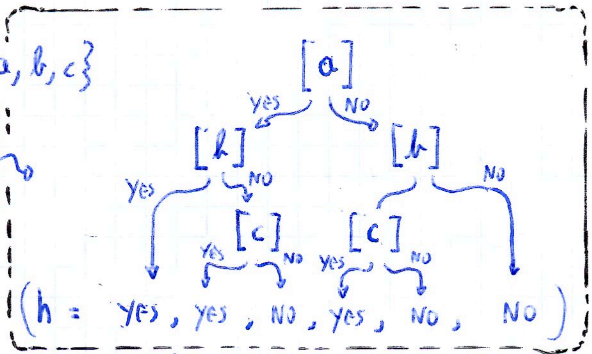
$(f : A \to Y \rightsquigarrow f(\vec{x}) = y \in Y)$ A = finite set of ATTRIBUTES / FEATURES

In trees the hypotesis test ONE ATTRIBUTE at each NODE.

- example:

$A = \{a, b, c\}$

here: [a, b, c, $f(\vec{x})$ are BINARY (YES/NO)]



TESTS ARE for a finite number of values and lead to different nodes depending on the values of attributes (here: binary attributes)

LEAVES contains the final output value for the set of attributes.

(We will focus on the case where the output value is finite (CLASSIFICATION) on the possible VALUE RANGES of attributes $\in A$ are given)

Decision trees are fairly expressive (they can model exactly, for example, boolean functions: in the example, $f(\vec{x}) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$.) For boolean functions the can be "folded" into BDDs (BINARY DECISION DIAGRAMS, as in the example)

- LEARNING DECISION TREES

From a set of A and the corresponding values of f we wanna find the SMALLEST TREE.
We cannot use brute force (computational) → HEURISTICS is what we need (small but not smallest trees)
} THE GREEDY ALGORITHM

- THE GREEDY ALGORITHM: [EXAMPLES = VALUES OF ATTRIBUTES + CORRESPONDING VALUE of UNKNOWN f]

function DECISION-TREE-LEARNING (examples, attributes, parent_examples) **returns** a tree
  if "examples" is empty **then return** PLURALITY VALUE (parent_examples) // if we have no more examples, we return the plurality value (the answer that is most frequent) of the PARENT → none of the examples were matching the combination of attribute on that branch
  else if all "examples" have the same classification **then return** then classification // try to DIRECTLY ANSWER for as many examples as possible
  else if "attributes" is empty **then return** PLURALITY VALUE (examples) // if we have no more attributes we return the PLURALITY VALUE of the remaining examples
    → there are hidden attributes to which we do not have access; or the data is inconsistent (e.g. because of noise)
  else
    $A \leftarrow \underset{a \in \text{attributes}}{\text{argmax}} \left[\text{IMPORTANCE}(a, \text{examples}) \right]$ (WE USE INFORMATION THEORY → SEE NEXT PARAGRAPH → ENTROPY) // the idea is to test the MOST DISCRIMINATING attributes first
    tree ← a new decision tree with root test A
    for each value v_K of A **do**
      $exs \leftarrow \{e, \text{s.t.} (e \in examples) \wedge (e.A = v_K)\}$
      subtree ← DECISION-TREE-LEARNING (exs, attributes − A, examples)
      add a branch to tree with label $(A = v_K)$ and subtree subtree
    return tree

for examples for which we have no direct answer we call the algorithm recursively with less examples to fit, and less attribute

- FINDING IMPORTANT ATTRIBUTES: ENTROPY

Entropy of a variable V: $H(V) = -\sum_K P(v_K) \log_2 \left(P(v_K) \right)$ (POSSIBLE VALUE OF V) → [MEASURES THE UNCERTAINTY ON V AS THE NUMBER OF BITS OF INFORMATION OBTAINED FROM A VALUE OF V]

- examples:
  - entropy of a random variable with only one value: $H(V) = -1 \log_2(1) = 0$
  - entropy of a fair coin flip: $H(V) = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = -0.5(-1) + 0.5 = 1$
  - entropy of a 4 sided dice: $H(V) = -4 \cdot \frac{1}{4} \log_2(1/2^2) = 2$

The set of EXAMPLES can be seen as a random variable with: [VALUES → THE OUTPUT VALUES; PROBABILITIES → " " " " FREQUENCIES]

↳ We will consider only BINARY OUTPUTS (yes/no) → $P(V = \text{true}) = \frac{p}{p+n}$ ← (FREQUENCIES) P = POSITIVE FREQ. (#YES), N = NEG. FREQ (#NO)

For a random boolean variable that is true with probability q, the entropy is:

$$\left[H_B(q) = -\left(q \cdot \log_2(q) + (1-q) \log_2(1-q)\right) \right]$$

If we have an attribute $A_{TRR} = \{ v_K \mid K \in [1, d] \}$ we'll divide the examples in d subsets E_K:
Each E_K contains P_K examples with output "yes" and n_K with output "no".
The corresponding entropy is: $H_B\left(\frac{P_K}{P_K + n_K}\right)$

The probabilities of having subset E_K value (Prob. that A has the K^{TH} value) is $\frac{P_K + n_K}{P + n}$
So the EXPECTED ENTROPY after choosing A is:

$$\boxed{R(A) = \sum_{K=1}^{d} \frac{P_K + n_K}{P + n} H_B\left(\frac{P_K}{P_K + n_K}\right)}$$

Since we want to be as certain as possible of the value of the output after choosing A, we want to minimize this value.

- INFORMATION GAIN:

minimizing $R(A) \equiv$ maximizing INFORMATION GAIN $\boxed{IG(A) = H_B\left(\frac{P}{P+n}\right) - R(A)}$
(is not suitable when an A has many different values)

To correct the BIAS of the IG we need to compute:

INFORMATION GAIN RATIO: $\boxed{IGR(A) = \frac{G(A)}{IV(A)}}$ INTRINSIC VALUE: $\boxed{IV(A) = -\sum_{K=1}^{d} \frac{n_K + P_K}{n + P} \log_2\left(\frac{n_K + P_K}{n + P}\right)}$

(is an estimation of the entropy of the random variable that gives a value for this attribute)

- example:

A_1 = PROBLEMS	A_2 = SIZE	Y = DECISION
BIG	SMALL	Yes
NO	SMALL	No
SMALL	BIG	Yes
BIG	BIG	Yes
BIG	BIG	No
SMALL	SMALL	No
SMALL	SMALL	Yes
BIG	SMALL	Yes
NO	BIG	No
SMALL	BIG	Yes
SMALL	BIG	No
SMALL	SMALL	Yes
SMALL	BIG	No

$\Rightarrow$ (FREQUENCIES)

PROBLEMS	SIZE
$P_{BIG} = 3$	$P_{BIG} = 3$
$N_{BIG} = 1$	$N_{BIG} = 4$
$P_{SMALL} = 4$	$P_{SMALL} = 4$
$N_{SMALL} = 3$	$N_{SMALL} = 2$
$P_{NO} = \emptyset$	
$N_{NO} = 2$	

$$R(\text{PROBLEM}) = \left(\frac{4}{13} \times H_B\left(\frac{3}{4}\right)\right) + \left(\frac{7}{13} \cdot H_B\left(\frac{4}{7}\right)\right) + \left(\frac{2}{13} \cdot H_B(\emptyset)\right)$$
BIG ; SMALL NO

$$H_{BIG}\left(\tfrac{3}{4}\right) = -\left(\frac{3}{4}\log_2\left(\frac{3}{4}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right)\right) \cong 0.81 =$$

$$R(\text{PROBLEM}) = \left(\frac{4}{13} \cdot 0{,}81\right) + \left(\frac{7}{13} \cdot 0.985\right) + \left(\frac{2}{13} \cdot \emptyset\right) \cong 0.7796$$

$R(\text{SIZE}) = (\dots)$

$$IV(\text{PROBLEM}) = -\frac{4}{13}\log_2\left(\frac{4}{13}\right) - \frac{7}{13}\log_2\left(\frac{7}{13}\right) - \frac{2}{13}\log_2\left(\frac{2}{13}\right) \cong 1.06$$

$IV(\text{SIZE}) = (\dots)$

NO!

$$IGR(\text{PROBLEM}) = \frac{IG}{IV} = \frac{\cancel{0.7796}}{1.06} = 0{,}735$$

- GENERALIZATION AND OVERFITTING

The greedy algorithm can infer PATTERNS FROM NOISE $\rightarrow$ the algorithm will OVERFIT creating nodes for useless attributes.

(The more training data, the less there is this problem)

Decision tree PRUNING is a technique to address this problem.

- DECISION TREE χ^2 PRUNING: (PG 705 "A.I" BOOK)

We start with a full tree. We test nodes that have ONLY LEAF NODES as successors.
How we decide to prune or not a node? If its proportion of positives and negatives p_k, n_k calculated with $\frac{p_k}{p_k+n_k}$, is roughly similar ($\approx$ the same) as the general $\frac{p}{n+p}$ it is an irrelevant attribute. In this case, his IG will be close to $\phi \rightarrow IG(A)$ good indicator of IRRELEVANCE. How measure it? with a

SIGNIFICANCE TEST:
1. The test begins assuming that $\nexists$ any underlying pattern (NULL HYPOTESIS ASSUMPTION)
2. We calculate how much the actual data DEVIATES from the perfect absence of pattern hypotesis
3. If the degree of deviation is statistically unlikely (usually taken to mean 5% or less) then that is considered to be a good evidence for the presence of a SIGNIFICANT PATTERN, If not we accept the null hypothesis (→ WE PRUNE THE NODE)

Computation → NULL HYP. ⇒ $IG(Attr) = \phi$ for very large samples $v = n+p$ (DIM. OF SAMPLE)

How to misure the deviation Δ? We can express the expected values with these PROPORTIONS: $\hat{p}_k = p_k \frac{p_k+n_k}{p+n}$; $\hat{n}_k = n_k \frac{n_k+p_k}{n+p}$ (ASSURING TRUE IRRELEVANCE)

$$\Delta = \sum_{k=1}^{d} \frac{(p_k-\hat{p}_k)^2}{p_k} + \frac{(n_k-\hat{n}_k)^2}{n_k}$$

Let's consider an aleatoric variable X that follow the χ^2 DISTRIBUTION with $(v-1)$ DoF (SEE WIKIPEDIA TABLE)
If NULL HYPOTESIS TRUE, then Δ is DISTRIBUTED as X!

example: $v = 4$, $DoF = 3$ → $\chi^2 = 7.92$ → $1-\alpha = 95\%$ (FROM TABLE)

THIS MEANS THAT $P(X < 7.92) = 95\%$

So we will accept the NULL HYP. if $\Delta \leq \chi^2 = 7.92$ with a confidence of $\alpha = 5\%$

→ $P(\Delta < 7.92) = 95\% = 1-\alpha$
$P(\Delta \geq 7.92) = 5\% = \alpha$

So, the number of nodes PRUNED will be in function of the confidence α.

When we PRUNE A NODE, we replace it with a LEAF with the PLURALITY VALUE.

- **RANDOM FORESTS**

Are an ENSEMBLE LEARNING METHOD, they consist in using BAGGING and the RANDOM SUBSPACE METHOD for the decision trees. This usually improves the accuracy of decision trees, while remaining quite fast and fairly explainable.

- **DECISION TREES: SUMMARY**

Decision trees are FAST, the results are EXPLAINABLE and they can be combined in random forests to gain ACCURACY.
We can use also LINEAR REGRESSION in the leaves to REGRESSION TREES

4.2 LINEAR MODELS

- **LINEAR REGRESSION**

We consider now $\mathcal{H} = \{ \sum_{i=0}^{n} w_i x^i \mid i \in [0,n], w_i \in \mathbb{R}\ \forall i \}$ ← LINEAR FUNCTIONS SPACE
For n attributes we have $n+1$ parameters w_i. For the REGRESSION PROBLEM we want to find the straight line $h_{\vec{w}}$ that approximates the best set of examples.

Gauss told us that if the noise on data is normally distributed then this means minimizing the EUCLIDEAN DISTANCE between the actual output and the prediction.
↳ minimizing L_2 over N given examples:

$$\text{Loss}(h_{\vec{w}}) = \sum_{k=1}^{N} L_2(y_k, h_{\vec{w}}(\vec{x}_k))$$

→ We can find the GLOBAL MINIMUM (the function is convex) by writing that all PARTIAL DERIVATIVES ARE 0

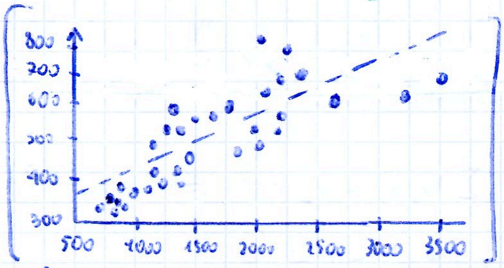
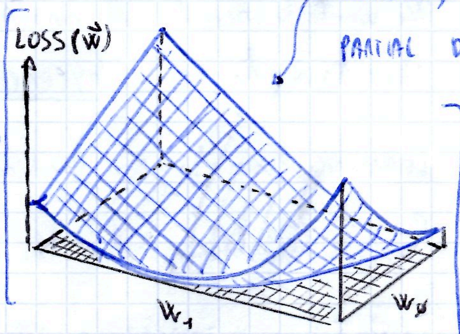


FIGURE 18.13 (a) "IA" BOOK: data points of price vs. floor space for houses in Berkeley (CA) in 2009 → $y = (0.232)x + (246)$ (PG. 718 BOOK)

LOSS FUNCTION



REMIND: (TO COMPUTE IT)
$L_2(y, \hat{y}) = (y - \hat{y})^2$

- OVERFITTING AND REGULARIZATION:

In dimension > 2 some attributes may exhibit spurious relations in the example set that will be captured by learning (→ OVERFITTING). To mitigate this we use REGULARIZATION on the loss function.

We consider REGULARIZATION TERMS of the form:

$$\text{Loss}(h_{\vec{w}}) = \sum_{k=1}^{N} \Big(y_k - \sum_{j=0}^{n} w_j x_k^j\Big)^2 + \overbrace{\sum_{i=0}^{n} |w_i|^q}^{\text{REG. TERMS } L_q}$$

Two interesting cases are $q=1$ (L_1 Regularization), that tends to produce SPARSE HYPOTHESES by setting weights to $\emptyset$; and $q=2$ (L_2 Regularization), more appropriate if the choice of the features is a bit arbitrary (e.g. coordinates in a rotationally invariant problem).

• Gradient Descent

With L_1 regularization $Loss(h_{\vec{w}})$ is not differentiable anymore → instead of computing the solution in closed-form, we can use iterative methods such as GRADIENT DESCENT:

$\vec{w} \leftarrow$ any point in the parameters space
loop until convergence do
    for each w_i in $\vec{w}$ do
        $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(h_{\vec{w}})$

The STEP SIZE α is also called "LEARNING RATE". It can be fixed in time or it can decay over time as the learning process proceeds.

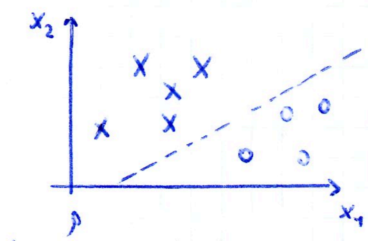
Since Loss is a big sum on a batch of examples, this is called BATCH GRADIENT DESCENT.

Convergence is guaranteed provided that α is SMALL ENOUGH, but it could be VERY SLOW.

So, we can also RANDOMLY select one of the examples at each iteration (→ STOCHASTIC GRAD. DESC)

It is usually faster, but convergence is guaranteed only if we make α decrease at each iteration with $\left(\sum_{t=1}^{\infty} \alpha(t) = \infty\right)$ and $\left(\sum_{t=1}^{\infty} [\alpha(t)]^2 < \infty\right)$

• Linear Classification



(example for dimension = 2)

We want to separate all points in two categories (1 and $\emptyset$). We want the separation to be an HYPERPLANE (i.e. defined by a linear equation).

LEARNING consists in finding $\vec{w}$ s.t. $\left[\sum_i w_i x_i + w_\emptyset \geq \emptyset \iff answer(\vec{x}) = 1\right]$

So, an hypothesis has the form $\left[h_{\vec{w}}(\vec{x}) = T(\vec{w} \cdot \vec{x})\right]$ where (DOT PRODUCT)

$T(\vec{w}\cdot\vec{x})$ is a THRESHOLD FUNCTION such that $T(z) = \begin{cases} 1, & z \geq \emptyset \\ \emptyset, & \text{otherwise} \end{cases}$

We also assume a DUMMY COORD. $x_\emptyset$ that is always equal to 1.

- HARD THRESHOLD:

For $h_{\vec{w}}(\vec{x}) = T(\vec{w}\cdot\vec{x})$, $Loss(h_{\vec{w}})$ is not differentiable and the gradient is almost always $\emptyset$.

We have to find something different.

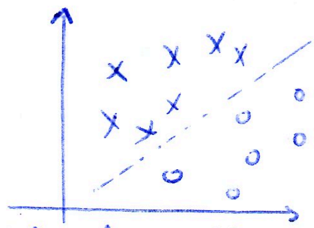
For any example $(\vec{x_K}, y_K)$, intuitively:

1. if $y = h_{\vec{w}}(\vec{x})$: the weights $\vec{w}$ are fine
2. if $y = 1$ and $h_{\vec{w}}(\vec{x}) = \emptyset$: we want $h_{\vec{w}}$ to be bigger (i.e. w_i to be bigger if x_i is positive and smaller otherwise)
3. if $y = \emptyset$ and $h_{\vec{w}}(\vec{x}) = 1$: we want $h_{\vec{w}}$ to be smaller

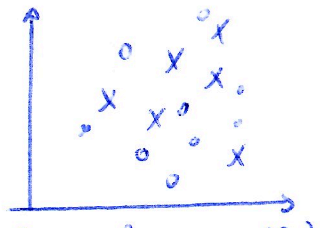
↓

The PERCEPTRON LEARNING RULE does this: $\left[w_i \leftarrow w_i + \alpha\left(y - h_{\vec{w}}(\vec{x})\right) x_i \right]$

[CONVERGENCE is guaranteed if the data is LINEARLY SEPARABLE. ⇝ Otherwise, we need to make α decrease over iterations]

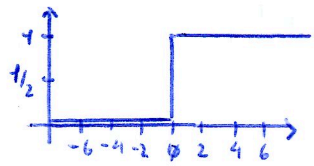


(linearly separable)

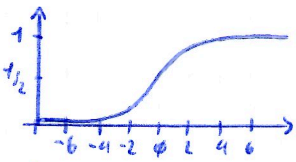


(non linear separable)

- LOGISTIC THRESHOLD:



(HARD) THRESHOLD



LOGISTIC FUNCTION

Instead of a hard threshold we can use other SIMILARLY SHAPED function. The LOGISTIC FUNCTION (a sigmoid) is a good one:

$$\left[\mathcal{L}(z) = \frac{1}{1 + e^{-z}} \right]$$

It has values between $\emptyset$ and 1, so we decide by rounding to the nearest value. It is DIFFERENTIABLE. We can then compute the gradient update (using $\mathcal{L}'(z) = \mathcal{L}(z)\left(1 - \mathcal{L}(z)\right)$.):

$$\left[w_i \leftarrow w_i + \alpha\left(y - h_{\vec{w}}(\vec{x})\right) h_{\vec{w}}(\vec{x})\left(1 - h_{\vec{w}}(\vec{x})\right) x_i \right]$$

- EXAMPLE: LEARNING BOOLEAN FUNCTIONS

X_1	X_2	$X_1 \vee X_2$
0	0	0
0	1	1
1	0	1
1	1	1
0	0	0

Use the perceptron learning rule with ($\alpha = 1$) and starting from ($w_\emptyset = \emptyset$, $w_1 = \emptyset$, $w_2 = 1$) to compute a linear classifier for $X_1 \vee X_2$ (using Hard Threshold)

I ROW: $\underbrace{X_\emptyset}_{(\text{DUMMY}=1)} w_0 + X_1 w_1 + X_2 w_2 = \emptyset + \emptyset \cdot \emptyset + \emptyset \cdot 1 = \emptyset \xrightarrow{T(\emptyset)} = 1$ (BUT $X_1 \vee X_2 = \emptyset$)

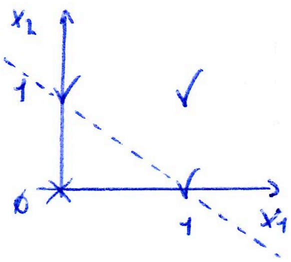
UPDATE: $\left\{ \begin{array}{l} w_\emptyset \leftarrow w_\emptyset + 1(\emptyset - 1) \overset{(\cdot X_\emptyset)\,(\cdot 1)}{} = -1 \\ w_1 \leftarrow w_1 + 1(\emptyset - 1)\underset{X_1}{\emptyset} = \emptyset \\ w_2 \leftarrow 1 \end{array} \right\}$

II ROW: $-1 + \overset{x_1}{\emptyset}\cdot\emptyset + \overset{x_2}{1}\cdot 1 = \emptyset \xrightarrow{T(\emptyset)} 1$ $\quad (x_1 \vee x_2 = 1)$ OK

III ROW: $-1 + \overset{x_1}{1}\cdot\emptyset + \overset{x_2}{\emptyset}\cdot 1 = -1 \xrightarrow{T(-1)} \emptyset$ $\quad (x_1 \vee x_2 = 1) \rightarrow$ UPDATE: $\{w_\emptyset \leftarrow \emptyset,\ w_1 \leftarrow 1,\ w_2 \leftarrow 1\}$

IV ROW: $\emptyset + \overset{x_1}{1}\cdot 1 + \overset{x_2}{1}\cdot 1 = 2 \xrightarrow{T(2)} 1$ $\quad (x_1 \vee x_2 = 1)$ OK

V ROW: $\emptyset + \overset{x_1}{\emptyset}\cdot 1 + \overset{x_2}{\emptyset}\cdot 1 = \emptyset \xrightarrow{T(\emptyset)} 1$ $\quad (x_1 \vee x_2 = \emptyset) \rightarrow$ UPDATE: $\left\{\begin{array}{l} w_\emptyset \leftarrow w_\emptyset + 1(\emptyset - 1)\cdot \underset{\sim}{\overset{1}{}}x_0 = -1 \\ w_1 \leftarrow w_1 + 1(\emptyset - 1)\cdot x_1 = 1 \\ w_2 \leftarrow w_2 + 1(\emptyset - 1)\cdot x_2 = 1 \end{array}\right\}$



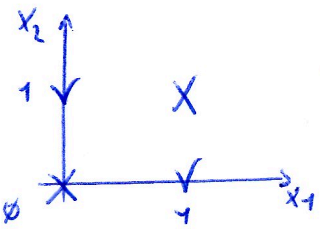
$\leftarrow w_\emptyset + w_1 x_1 + w_2 x_2$ AFTER THE 5 EXAMPLES

$-1 + x_1 + x_2 = \emptyset \rightarrow x_2 = -x_1 + 1$

FOR PLOTTING THE LINE

↳ OVER THE LINE: ALL ✓ ; UNDER THE LINE: ALL ✗ $\Rightarrow$ ["LEARNING GRAPHICALLY CONFIRMED"]

What for the XOR?



→ NOT LINEARLY SEPARABLE! → no convergence with previous method

But with decreasing α yes?? BAH...

- CONCLUSION

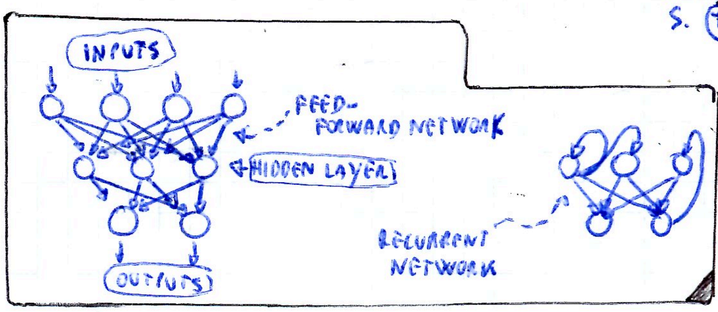
Linear classification is often the FASTEST classification technique.
Many interesting problems are linearly separable.
Instead of computing any hyperplane, we can try to compute the one with the BIGGEST MARGIN to the nearest examples: this is the principle of SUPPORT VECTOR MACHINES (SVM), one of the most efficient classification techniques.

To deal with non linearity, there are two many ideas:

- The KERNEL TRICK: consider the data in a higher dimension space. Under suitable conditions, this representation need not be explicitly computed. This is in particular applicable to SVM.

- COMBINE several linear classifiers.

4.3 Artificial Neural Networks

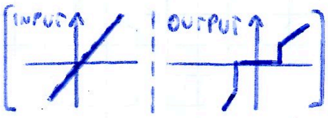


Each artificial neuron is a LINEAR CLASSIFIER.

- **Perceptrons**

A perceptron is a LINEAR CLASSIFIER
↳ ANN with a single layer (directly depends on input)
↳ each neuron has an ACTIVATION FUNCTION, it could be:

- hard threshold
- logistic
- tanh(x)
- identity
- rectified linear unit: $ReLU(x) = \max(\emptyset, x)$ } → most used in DEEP NETWORKS



logistic, tanh(x) } ↳ most used in SHALLOW NETWORKS

[VIDEO TIP:
- Welch Labs (YT channel)
- 3Blue1Brown (YT channel)
↓
(VIDEOS ON NEURAL NETWORKS)]

- **Multilayer ANN → Learning** (MULTILAYER PERCEPTRONS)

$L \triangleq$ # layers, $g \triangleq$ activation function ; FULL INTERCONNECTED LAYERS (K-th INPUT OF NEURON IN LAYER ℓ IS THE K-th NEURON OF LAYER $\ell-1$)

$w^{\ell}_{jk} \triangleq$ weight of K-th INPUT of J-th NEURON in ℓ-th LAYER

$z^{\ell}_{j} \triangleq$ weighted input of J-th NEURON in ℓ-th LAYER ($\rightarrow a^{\ell}_{j} \triangleq$ the output)

($\forall$ LAYER $\exists$ DUMMY NEURON ($j = \emptyset$ th) s.t. $a^{\ell}_{\emptyset} = 1$) ⇒ So in the layer $\ell+1$ ($\forall \ell$) we can model the BIAS ↗ $w^{\ell+1}_{j\emptyset}$

So we can write: ↓

$$\left[z^{\ell}_{j} = \sum_{k} w^{\ell}_{jk}\, a^{\ell-1}_{k} = \sum_{k} w^{\ell}_{jk} \cdot \underbrace{g}_{\text{ACTIV. FUNCT.}}\left(z^{\ell-1}_{k}\right) \right]$$

[Let's calculate the OUTPUT LAYER ERROR] ↓

Assuming that the loss function is ADDITIVE across the components of $\vec{y}$: ↓ (we use L_2 here)

$$\left[L_2(\vec{x}, \vec{y}, \hat{\vec{y}}) = \| \vec{y} - \hat{\vec{y}} \|^2 = \| \vec{y} - \vec{h}_{\vec{w}}(\vec{x}) \| = \sum_{k} (y_k - a^{L}_{k})^2 \right]$$

["L_2" IS A PARTICULAR "LOSS" FUNCTION, WE CAN CHANGE WITH OTHERS]

(ERROR ON THE J-th NEURON) → $\left[\dfrac{\partial L_2}{\partial a^{L}_{j}} = -2(y_j - a^{L}_{j}) \right]$ $\xrightarrow{\text{MORE CONVENIENT}}$ (MODIFIED ERROR) → $\left[\Delta^{L}_{j} = \dfrac{\partial L_2}{\partial z^{L}_{j}} = g'(z^{L}_{j}) \dfrac{\partial L_2}{\partial a^{L}_{j}} \right]$

↓

$(\Delta^{\ell}_{j} \triangleq \partial L / \partial z^{\ell}_{j})$

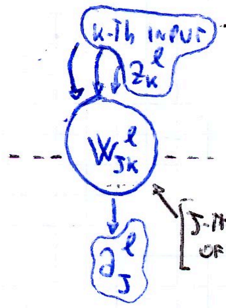
- UPDATING WEIGHTS:

$$\frac{\partial L_2}{\partial w^{L}_{jk}} = \frac{\partial L_2}{\partial z^{L}_{j}} \frac{\partial z^{L}_{j}}{\partial w^{L}_{jk}} = \Delta^{L}_{j} \frac{\partial \left(\sum_i w^{L}_{ji} a^{L-1}_{i} \right)}{\partial w^{L}_{jk}} = \Delta^{L}_{j}\, a^{L-1}_{k} \quad \text{[for OUTPUT LAYER]}$$

$$\frac{\partial L_2}{\partial w^{\ell}_{jk}} = \Delta^{\ell}_{j}\, a^{\ell-1}_{k} \quad \text{[for layer } \ell \text{]}$$

← WEIGHTS ERRORS → we compute Δ^{ℓ}_{k} with BACKPROPAGATION ↓

◦ BACKPROPAGATION:



[IS THE SUM OF THE OUTPUTS OF ALL THE "J-TH" NEURON OF $\ell-1$ LAYER]

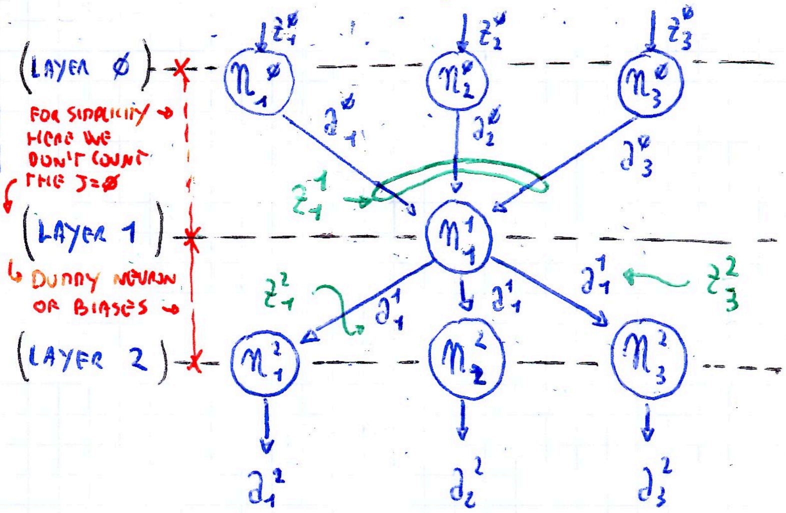
[ℓ-th LAYER]

[J-TH NEURON OF LAYER ℓ]

↳ [IS THE "K-TH" INPUT OF THE $\ell+1$ LAYER]

We got before that $\left(\dfrac{\partial L_2}{\partial w_{JK}^\ell} = \Delta_J^\ell \, a_K^{\ell-1} \right)$

Let's consider this example:



[n_1^1 => NEURON #1 OF LAYER 1 HAS 3 INPUTS: z

$z_1^1 = w_{11}^1 a_1^0 + w_{12}^1 a_2^\varnothing +$
$+ w_{13}^1 a_3^\varnothing$

NEURON OF LAYER 1 — INPUT FROM LAYER Ø]

We compute Δ_J^ℓ of each layer starting from the output layer error

↓ BACKPROPAGATION of Δ_J^L

[1) COMPUTE OUTPUT LAYER ERROR: $\Delta_1^2 = -2(y_1 - a_1^2)$; $\Delta_2^2 = \dots$, $\Delta_3^2 = \dots$

2) BACK PROPAGATE TO LAYER 1 : $\Delta_1^1 = \sum_m g'(z_1^1) \cdot w_{m1}^2 \Delta_m^2$

LOOP THROUGH THE LAYER 2 — (IS NOT a_1^1! IS THE DERIVATIVE) — # NEURON OF LAYER 2 — # INPUT OF LAYER 2 (# NEURON OF LAYER 1)

3) DO THE SAME FOR LAYER Ø : $\Delta_1^\varnothing = \dots$; $\Delta_2^0 = \dots$; $\Delta_3^\varnothing = \dots$]

$$\Delta_{J_\ell}^\ell = \frac{\partial L_2}{\partial z_{J_\ell}^\ell} = \sum_{J_{\ell+1}} \frac{\partial L_2}{\partial z_{J_{\ell+1}}^{\ell+1}} \cdot \frac{\partial z_{J_{\ell+1}}^{\ell+1}}{\partial z_{J_\ell}^\ell} = \sum_{J_{\ell+1}} g'(z_{J_\ell}^\ell) \, w_{J_{\ell+1} J_\ell}^{\ell+1} \, \Delta_{J_{\ell+1}}^{\ell+1}$$

ERROR! SHOULD BE INSIDE THIS!

(# NEURON OF LAYER ℓ) (LOOP THROUGH THE NEURONS OF $\ell+1$ LAYER) ($J_{\ell+1}$ TH NEURON OF LAYER $\ell+1$) (J_ℓ TH INPUT FROM J_ℓTH NEURON OF LAYER ℓ)

◦ MATRIX FORM:

[$\underline{\underline{W}}^\ell = \{ w_{JK}^\ell, \forall_{J,K} \}$; $\underline{z}^\ell = \{ z_J^\ell, \forall_J \}$; $\underline{a}^\ell = \{ a_J^\ell, \forall_J \}$; $\underline{\Delta}^\ell = \{ \Delta_J^\ell, \forall_J \}$ ← (MATRIXES FOR LAYER ℓ)

$\underline{\nabla_a} L_2 = \left\{ \dfrac{\partial}{\partial a_J^L} L , \forall_J \right\}$]

HADAMARD PRODUCT (ELEMENT BY ELEMENT → DOT MATRIX PRODUCT IN MATLAB)

(OUTPUT LAYER ERROR) → $\left[\underline{\Delta}^L = \underline{\nabla_a} L_2 \circ g'(\underline{z}^L) \right]$

(INPUT FOR THE FEED FORWARD) → $\left[\underline{\underline{W}}^\ell \cdot \underline{a}^{\ell-1} = \underline{z}^\ell \right]$! NOT THE CONTRARY !

(BACKPROPAGATION) → $\left[\underline{\Delta}^{\ell-1} = (\underline{\underline{W}}^\ell)^T \underline{\Delta}^\ell \circ g'(\underline{z}^{\ell-1}) \right]$

(WEIGHT UPDATE) → $\left[\underline{\underline{W}}^\ell \leftarrow \underline{\underline{W}}^\ell - \alpha \, \underline{\Delta}^\ell (\underline{a}^{\ell-1})^T \right]$

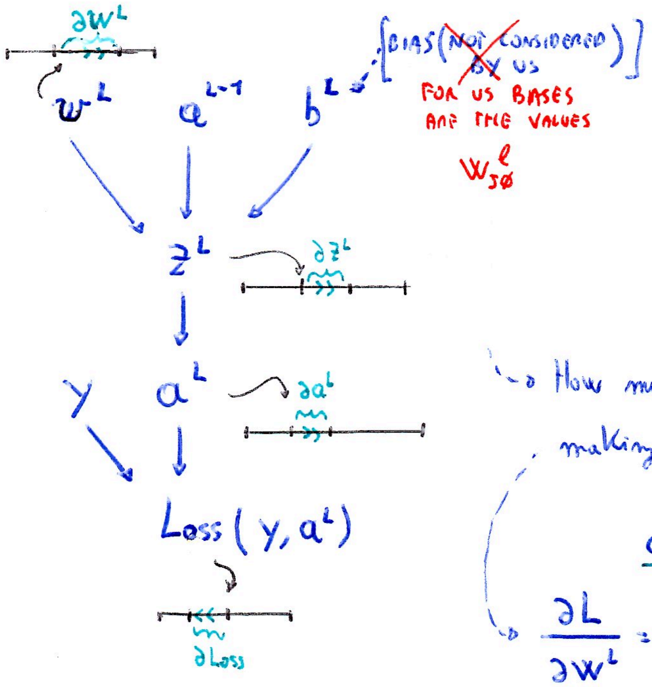
GRADIENT STEP (LEARNING RATE) > Ø

→ [USE BACKPROPAGATION TO DO GRADIENT DESCENT AS IN THE LINEAR CASE]

↓

(CONTINUE AFTER THE "VIDEO NOTES")

- FROM 3Blue1Brown "BACKPROPAGATION CALCULUS" VIDEO: (ALTERNATIVE/INTEGRATIVE EXPLANATION OF BACKPROPAGATION ALGORITHM)



If Loss $= L_2 \rightarrow$ loss $= (y - a^L)^2$

$$\leftarrow \begin{bmatrix} z^L = w^L a^{L-1} + b^L & (1) \\ a^L = g(z^L) & (2) \end{bmatrix}$$

$\hookrightarrow$ How much a variation on w^L (∂w^L) influences Loss making it vary of $\partial Loss$?

CHAIN RULE

$$\frac{\partial L}{\partial w^L} = \underset{(3)}{\frac{\partial z^L}{\partial w^L}} \cdot \underset{(2)}{\frac{\partial a^L}{\partial z^L}} \cdot \underset{(1)}{\frac{\partial L}{\partial a^L}}$$

$\hookrightarrow$ (1) $\dfrac{\partial L}{\partial a^L} = 2(y - a^L)$

$\hookrightarrow$ (2) $\dfrac{\partial a^L}{\partial z^L} = g'(z^L)$

$\left.\right\}$ ((1) $\cdot$ (2) $= \Delta^L$ FOR US)

$\hookrightarrow$ (3) $\dfrac{\partial z^L}{\partial w^L} = a^{L-1}$

NOTE:
[$\hookrightarrow$ How much a variation of BIAS b^L (∂b^L) influences $\partial Loss$? (WE DON'T SEE BIAS) WE USE $W^l_{J\varnothing}$

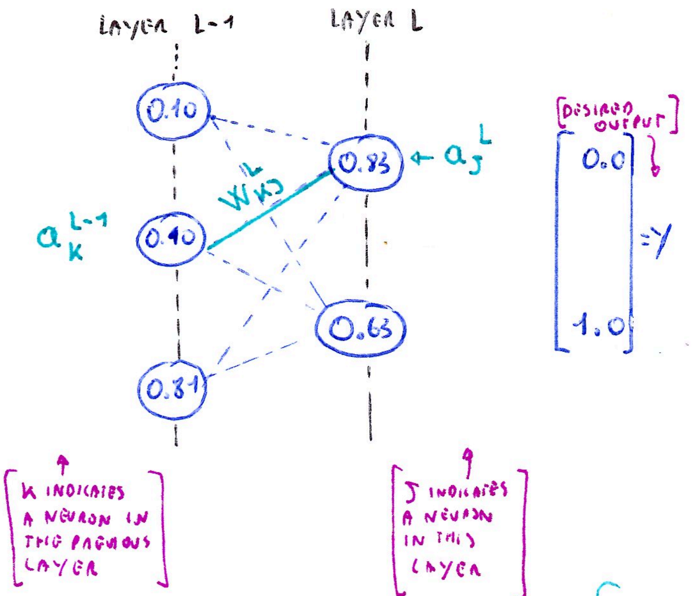
$\hookrightarrow$ (1) & (2) AS FOR w^L

(3) $\dfrac{\partial z^L}{\partial b^L} = 1$ $\leftarrow$ AS WE CAN SEE FROM EQUATION (1)]

NOTE:
$\hookrightarrow$ [How much a variation of a^{L-1} (∂a^{L-1}) influences $\partial Loss$? $\rightarrow$ (1) & (2) SAME FOR w^L

(3) $\dfrac{\partial z^L}{\partial a^{L-1}} = w^L$]

- GENERALIZE THE NOTATION



$\rightarrow$ Loss $= \sum_{J=\varnothing}^{n_L - 1} (y_J - a^L_J)^2$

$\rightarrow$ $z^L_J = \sum_{K=\varnothing}^{n_{L-1}-1} (w^L_{JK}\, a^{L-1}_K) + b^L_J$

$\rightarrow$ $a^L_J = g(z^L_J)$

$\rightarrow$ [CHAIN RULE]: $\dfrac{\partial L}{\partial w^L_{JK}} = \dfrac{\partial z^L_J}{\partial w^L_{JK}} \dfrac{\partial a^L_J}{\partial z^L_J} \dfrac{\partial Loss}{\partial a^L_J}$

BUT

$$\left[\frac{\partial Loss}{\partial a^{L-1}_K} = \sum_{J=\varnothing}^{n_L - 1} \frac{\partial z^L_J}{\partial a^{L-1}_K} \frac{\partial a^L_J}{\partial z^L_J} \frac{\partial Loss}{\partial a^L_J}\right]$$

So, in the end:

$$\left[\frac{\partial Loss}{\partial w^l_{JK}} = a^{l-1}_K \cdot \overbrace{g'(z^l_J)\,\frac{\partial Loss}{\partial a^l_J}}^{\text{FOR US IS } \Delta^l_J}\right]$$

$$\left[\begin{array}{l} \text{IS EQUAL TO } \left[2(y - a^L_J)\right] \\ \text{OR } \left[\sum_{J=\emptyset}^{n_{l+1}-1} w^{l+1}_{JK} \cdot g'(z^{l+1}_J)\,\frac{\partial Loss}{\partial a^{l+1}_J}\right] \end{array}\right]$$

→ ALL THE $\partial Loss$ COMPOSE THE $[\nabla Loss]$!

(WE USE GRADIENT DESCENT TO MINIMIZE THE LOSS)

• Gradient Descent with Backpropagation

Call $\tilde{X}$ the training set containing $[m]$ samples. So we have:

$[Loss \to Loss^{(x)}$ (loss for the sample $x \in \tilde{X}$)$]$ and,

ALL OF THOSE MATRIX(ES) REFERS TO THE SAMPLE X

(WEIGHT UPDATE): $\underline{\underline{w}}^l \leftarrow \underline{\underline{w}}^l - \alpha \times \frac{1}{m} \sum_x \underline{\Delta}^{l,x} (\underline{a}^{l-1,x})^T$

MORE SIMPLE → $\left[\underline{\underline{w}}^l \leftarrow \underline{\underline{w}}^l - \frac{\alpha}{m} \underline{\Delta}^l (\underline{a}^{l-1})^T\right]$ and BIAS UPDATE: $\left[\underline{w}_\emptyset^l \leftarrow \underline{w}_\emptyset^l - \frac{\alpha}{m} \underline{\Delta}^l \cdot \underset{\sim}{\vec{1}}\right]$

(COLUMN VECTOR OF ONES)

• Stochastic Gradient Descent

It's computationally faster split the training set T_s in subsets $\tilde{X}$ containing m' samples each

1. Select a random subset $\tilde{X}$ of m' samples from T_s
2. Train the network with $\tilde{X}$ to estimate gradient (average over $\tilde{X}$)
3. Repeat until there are not $\tilde{X}$s left

↳ $(1+2+3)$ = EPOCH of training → repeat epochs until convergence.

• Neuron Saturation

Since we start with random weights, it is possible that weighted inputs to neurons are very big or very small at some point, close to the "stable" regions of threshold-like function.
This implies that in such case the derivative is ALMOST ZERO ↳ [the modified error and the GRADIEND will be very small even if the output is not at all correct!]

Weights in hidden layers should be not initialized all to zero, in order to BREAK SIMMETRY,

draw them from INDIPENDEN GAUSSIAN distributions, with mean value $= \emptyset$ and standard deviation $= 1$, BUT...

But if a neuron has many inputs, the probability that it starts saturated is not negligible with this scheme → a STANDARD DEVIATION $= \frac{1}{\sqrt{n}}$ is better ($n = \#$ INPUTS)

- **CROSS-ENTROPY & LOGISTIC FUNCTION**

For an output layer with LOGISTIC NEURONS, we can design Loss to eliminate saturation

WE BUILD LOSS FOR HAVING THIS

↳ Since $\Delta_j^L = \mathcal{L}'(z_j^L)\frac{\partial Loss}{\partial a_j^L}$ We want $\frac{\partial loss}{\partial a_j^L} = \frac{a_j^L - y_j}{\mathcal{L}'(z_j^L)} = \frac{a_j^L - y_j}{\mathcal{L}(z_j^L)(1-\mathcal{L}(z_j^L))} = \frac{a_j^L - y_j}{a_j^L(1-a_j^L)}$

$\mathcal{L}'(z_j^L) = \mathcal{L}(z_j^L)(1-\mathcal{L}(z_j^L))$

$\to \int d a_j^L \to Loss_j = -y \log(a_j^L) - (1-y_j)\log(1-a_j^L)$

So $\left[Loss = -\sum_j y \log(a_j^L) + (1-y_j)\log(1-a_j^L)\right]$ ← CROSS-ENTROPY LOSS FUNCTION

It is always NON NEGATIVE, it is CLOSE TO $\emptyset$ when a_j^L is close to y (Recall that y and a^L are vectors of numbers between $\emptyset$ and 1)

↳ So, by construction, $\left[\Delta_j^L = a_j^L - y_j\right]$ ← WE BUILT "LOSS" FOR HAVING THIS

It is generally a BETTER SOLUTION than L_2 if the OUTPUT LAYER has LOGISTIC NEURONS!

- **THE SOFTMAX FUNCTION**

If we want that the neurons of the output layers will return a PROBABILITY DISTRIBUTION that is not a binary outcome (logistic function) we have to use the SOFTMAX FUNCTION:

$$\left[S(\vec{z})_i = \frac{e^{z_i}}{\sum_k e^{z_k}}\right]$$

← [SOFTMAX ACTIVATION FUNCTION] (each neuron represent an outcome and its activation represent the probability of that outcome)

↳ It takes in account the weighted input of ALL the neurons in the layer
It has values between $\emptyset$ and 1, and the sum of the values along all the output layer is 1.

- CROSS ENTROPY

We can use again, in a slightly different way, the CROSS-ENTROPY:

$$\left[Loss = \sum_j y_j \log(a_j^L)\right]$$

$(S(\vec{z})_i)$ is similar to $(\mathcal{L}(z_j^L))$, in fact: $\left(\frac{\partial S(\vec{z})}{\partial z_j}\right)_i = \begin{cases} S(\vec{z})_j(1-S(\vec{z})_i) & \text{if } i = j \\ -S(\vec{z})_i S(\vec{z})_j & \text{otherwise}\end{cases} = S(\vec{z})_j(\delta_{ij} - S(\vec{z})_i)$

Knowing that, and remembering that $a_i^L = S(\vec{z}^L)_i$, we can compute the gradients:

$$\frac{\partial loss}{\partial w_{jk}^L} = \sum_i \frac{\partial loss}{\partial a_i^L}\frac{\partial a_i^L}{\partial w_{jk}^L} = \sum_i \frac{y_i}{a_i^L}\frac{\partial a_i^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{jk}^L} = \sum_i \frac{y_i}{a_i^L}\frac{\partial a_i^L}{\partial z_j^L} a_k^{L-1} = \sum_i \frac{y_i}{a_i^L} a_j^L(\delta_{ij} - a_i^L)\, a_k^{L-1}$$

[But $\vec{y}$ is $\emptyset$ $\forall i \neq j$ (null for each component representing a wrong answer)] !! So,

$$\frac{\partial loss}{\partial w_{jk}^L} = \frac{y_j}{a_j} a_j^L(1 - a_j^L)\, a_k^{L-1} = a_k^{L-1}\underbrace{(y_j - a_j^L)}_{\Delta_j^L}$$

???

Δ_j^L → OK NO SATURATION (AS WITH LOGISTIC FUNCTION)

• Regularisation in ANN

PG 12

Is helpful to use REGULARISATION to limit OVERFITTING (AS SEEN IN 4.2 LINEAR MODELS) but this time we can use COMBINATIONS of $L_1 + L_2$ REGULARIZATION → ELASTIC NET OF REGULARIZATION.

This can also be done by acting directly on the weights → DROPOUT REGULARIZATION

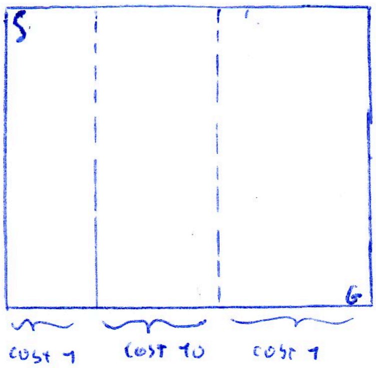
(As with all other methods, INCREASING THE SIZE of the training set, possibly artificially, reduces overfitting.)

Artin Exercises

- Exam 17-18 - Ex-1 : A^*

Robot moving in 12x10 tiles, with random walls. Moves: ↑, ↓, →, ←,

Start: (1,1) Goal (12,10) Cost of passing = { cols 4,5,6,7 ⇒ (10) ; cols 1,2,3,8,9,10, 11,12 ⇒ (1) }



Q1 Assuming that exists a path to the goal, can the robot forced to move over more than 4 of the tiles that have cost 10?

Yes, if there is not a full row free from column 4 to column 8 you cannot have a straight line between the "cost 10" zone

Q2 Give an admissible heuristic for the robot to find a cost-optimal path in this problem, not trivial and take in account the fact that cols 4→7 cost is 10.

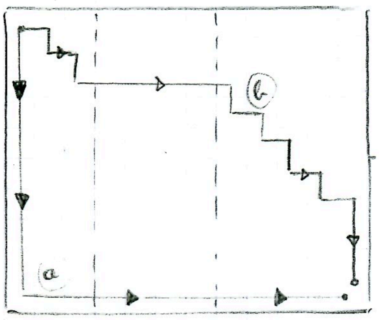
$h(\underline{x}) = manhattan(\underline{x}, \underline{GOAL}) = \Delta X + \Delta Y$ → I have to take account of the cost

↳ $h(\underline{x}) = man(\underline{x}, \underline{GOAL}) + col(\underline{x})$ with

$col(\underline{x}) = \begin{cases} (10-1)\cdot 4 & \text{if } X < 4 \\ \varnothing & \text{if } X > 7 \\ (10-1)\cdot(7-X) & \text{if } X \in [4,7] \end{cases}$

cause 1 is counted already in manhattan

Is it admissible?



Yes, because it never overestimates the cost of a $\underline{x}$.

$m(x,y) + col(x)$

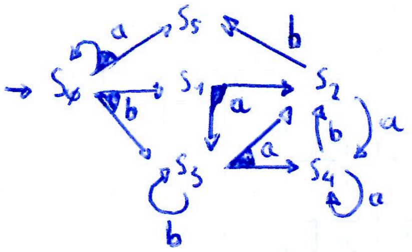
if $x < 4$, $9 * 4$

if $x > 7$ 0

if $x \in [4,7]$ $(7-x)*9$

- Exam '17/18 - Ex 2 : Non-determinism

Q3 Does there exist a winning strategy to go from $s_\emptyset$ to s_5 in the following hypergraph?



$W = s_5$ $\qquad$ $R = \begin{array}{cccccc} \emptyset & 1 & 2 & 3 & 4 & 5 \\ \end{array}$ $[F, F, F, F, F, T]$

↓ pop s_5

$R[s_\emptyset] = F \qquad R[s_2] = T \Rightarrow strat[s_2] = b \Rightarrow W = s_2$

↓ pop s_2

$R[s_1] = F \qquad R[s_4] = T \Rightarrow strat[s_4] = b \Rightarrow W = s_4$

↓ pop s_4

$R[s_3] = T \Rightarrow strat[s_3] = a \Rightarrow W = s_3$

↓ pop s_3

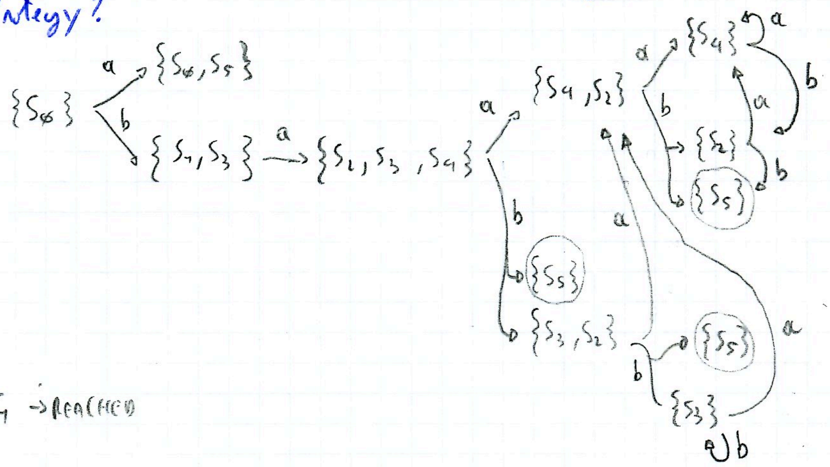
$R[s_1] = T \Rightarrow strat[s_1] = a \qquad R[s_\emptyset] = T \Rightarrow strat[s_\emptyset] = b$

WINNING STRATEGY:

$$\begin{bmatrix} s_\emptyset \to b \\ s_1 \to a \\ s_2 \to b \\ s_3 \to a \\ s_4 \to b \\ s_5 \mid \bot \end{bmatrix}$$

Q4 Suppose now the system is partially observable: $[O(s_\emptyset) = O(s_5) = o_1]$ and $[O(s_1) = O(s_2) = O(s_3) = O(s_4) = o_2]$
Does exist a winning strategy?

RESPECTIVE BELIEF HYPERGRAPH:



WINNING STRATEGY (example):

- $o_1 \to (b) \to o_2 \to (a) \to o_2 \to (b)$ → o_1 → REACHED
  - ↳ $o_2 \to (b)$ → o_1 → REACHED
    - ↳ $o_2 \to (a) \to o_2 \to (b)$ → o_1 → REACHED
      - ↳ $o_2 \to (b) \to o_1$ → REACHED

↳ an incorditional one is $[o_1 \to (b) \to o_2 \to (a) \to o_2 \to (a) \to o_2 \to (a) \to o_2 \to (b) \to o_2 \to (b) \to o_1]$

- Exam '17-'18 - Ex 3 : Markov Models

ARTIN EXES (2)

	1	2	3
1		-0.07	-1
2		-0.07	+2
3	-0.07	-0.07	

FINAL

10% of going LEFT instead of going in the desired direction (if goes into wall stays there)

Q_5 : Compute the first 3 iterations of the VALUE ALGORITHM for the MDP. Assume $\gamma = 1$. Detail the computation for tile (2,2)

$$\begin{pmatrix} \uparrow \Rightarrow \uparrow 90\% + \leftarrow 10\% \\ \leftarrow \Rightarrow \leftarrow 90\% + \downarrow 10\% \\ \downarrow \Rightarrow \downarrow 90\% + \rightarrow 10\% \\ \rightarrow \Rightarrow \rightarrow 90\% + \uparrow 10\% \end{pmatrix}$$

$$U_{i+1}(s)\, R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s', a, s)\, U(s')$$

$$U_\emptyset : (s_i) = \emptyset \quad \forall s_i$$

$$U_1(2,2) = -0.07 + \left[\uparrow 0.9(\emptyset) + 0.1(\emptyset),\ \rightarrow 0.9(\emptyset) + 0.1(\emptyset),\ \downarrow 0.9(\emptyset) + 0.1(\emptyset),\ \leftarrow 0.9(\emptyset) + 0.1(\emptyset) \right] = -0.07$$

$$U_1(s_i) = R(s_i), \forall s_i$$

$$U_2(2,2) = -0.07 + \left[\uparrow 0.9(-0.07) + 0.1(-0.07),\ \rightarrow 0.9(2) + 0.1(-0.07),\ \uparrow 0.9(-0.07) + 0.1(-0.07),\ \leftarrow 0.9(-0.07) + 0.1(-0.07) \right] = 1{,}723$$

$U_2(1,1) = -0.14$ strat $[1,1] = \rightarrow$; $U_2(2,1) = -0.14$ strat $[1,2] = \uparrow$

$U_2(2,3) = -0.14$ strat $[2,3] = \downarrow$; $U_2(3,2) = 2$; $U_2(3,3) = -1$

↳ STRAT : $(1,1) \rightarrow \overset{U_2}{[-0.14]}$ / $(1,2) \uparrow [-0.14]$ / $(2,2) \rightarrow [1.723]$ / $(3,2) \perp [2]$ / $(2,3) \downarrow [-0.14]$ / $(3,3) \perp [-1]$

Q_6 Suppose STRAT : $\rightarrow \ \forall s_i$, $P(\text{SQUEAK IN } (J, I)) = 10 \cdot (J + I)\%$

$$X = \begin{bmatrix} (3,1) \\ (3,2) \\ (2,2) \\ (1,2) \\ (2,3) \\ (1,3) \end{bmatrix} \quad P_\emptyset = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ \emptyset \\ \emptyset \end{bmatrix}$$

We observe (Squeak, Squeak, Not Squeak) What is the most probably position of the robot.

$$\begin{array}{c|cccccc} & (3,1) & (3,2) & (3,2) & (1,2) & (2,3) & (1,3) \\ \hline (3,1) & 0.1 & 0.9 & \emptyset & \emptyset & \emptyset & \emptyset \\ (3,2) & \emptyset & 0.9 & 0.1 & \emptyset & \emptyset & \emptyset \\ (2,2) & \emptyset & \emptyset & \emptyset & 0.1 & 0.9 & \emptyset \\ (1,2) & \emptyset & \emptyset & \emptyset & 0.1 & \emptyset & 0.9 \\ (2,3) & \emptyset & \emptyset & \emptyset & \emptyset & 1 & \emptyset \\ (1,3) & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & 1 \end{array}$$

$$O_{SQUEAK} = \begin{bmatrix} 40\% & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ & 50\% & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & 40\% & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & 30\% & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset\% & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset\% \end{bmatrix}$$

$O_{1:t} = (S, S, NS)$

$$l_{1:1} = O_S T^T l_\emptyset = \begin{bmatrix} 0.01 \\ 0.225 \\ 0.01 \\ 0.0150 \\ 0 \\ 0 \end{bmatrix} \qquad l_{1:2} = O_S T^T l_{1:1} = \begin{bmatrix} 0.0004 \\ 0.1058 \\ 0.0090 \\ 0.0008 \\ \emptyset \\ \emptyset \end{bmatrix} \qquad l_{1:3} = \underbrace{O_{NS}}_{(I_{6x6} - O_S)} T^T l_{1:2} = \begin{bmatrix} 0.0000 \\ 0.0478 \\ 0.0063 \\ 0.0007 \\ 0.0081 \\ 0.0007 \end{bmatrix}$$

$$F_{1:1} = \frac{l_{1:1}}{sum(l_{1:1})} = \begin{bmatrix} 3.8462\% \\ 86.5385\% \\ 3.8462\% \\ 5.7692\% \\ 0\% \\ 0\% \end{bmatrix} \qquad F_{1:2} = \begin{bmatrix} 0.3451\% \\ 91.2425\% \\ 7.7653\% \\ 0.6471\% \\ \emptyset\% \\ \emptyset\% \end{bmatrix} \qquad F_{1:3} = \begin{bmatrix} 0.0377\% \\ 75.1132\% \\ 9.9774\% \\ 1.0732\% \\ 12.7371\% \\ 1.0614\% \end{bmatrix}$$

(WITH MATLAB)

$\approx 75\%$ that robot is in $(3,2)$
$\approx 13\%$ in $(2,3)$
$\approx 10\%$ in $(2,2)$

- Exam '17-'18 - Ex 4 : Supervised Learning

x_1	0	1	0	1	1	0	1	1	0	0
x_2	1	1	0	1	0	0	0	1	1	0
x_3	1	1	1	0	0	1	1	1	0	0
$f(x)$	1	1	0	1	1	0	1	1	0	0

Q7 What is the IG for x_1 and x_2? Which one should be chosen as the root for a decision tree for approximating f?

$$IG(A) = H_B\left(\frac{p}{p+n}\right) - R(A) \qquad R(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} H_B\left(\frac{p_k}{p_k+n_k}\right) \qquad H_B(q) = -\left[q \log_2(q) + (1-q)\log_2(1-q)\right]$$

x_1) $\begin{array}{cccccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{array}$ f $\rightarrow$ $p_\emptyset + n_\emptyset = 5$ $\quad p_\emptyset = 1$; $p_1 + n_1 = 5$ $\quad p_1 = 5$

$$H_B\left(\frac{p}{p+n}\right) = H_B(3/5) = -\left[\frac{3}{5}\log_2\left(\frac{3}{5}\right) + \frac{2}{5}\log_2\left(\frac{2}{5}\right)\right] = 0.97$$

$$H_{B\emptyset}\left(\frac{1}{5}\right) = -\left[\frac{1}{5}\log_2\left(\frac{1}{5}\right) + \frac{4}{5}\log\left(\frac{4}{5}\right)\right] = 0.72 \qquad H_{B1}\left(\frac{5}{5}\right) = \emptyset$$

$$R(x_1) = \frac{1}{2}\cdot 0.72 + \frac{1}{2}\cdot\emptyset = 0.36 \qquad IG(x_1) = 0.97 - 0.36 = 0.61$$

x_2) $\begin{array}{cccccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{array}$ f $\rightarrow$ $p_0 + n_0 = 5$ $\quad p_\emptyset = 2$; $p_1 + n_1 = 5$ $\quad p_1 = 4$

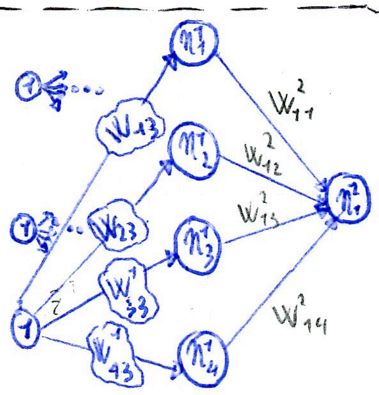
$$H_{B\emptyset}\left(\frac{2}{5}\right) = -\left[\frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right] = 0.97 \qquad H_{B\emptyset}\left(\frac{4}{5}\right) = 0.72$$

$$R(x_2) = \frac{1}{2}\,0.97 + \frac{1}{2}\,0.72 = 0.845 \qquad IG(x_2) = 0.125$$

$\boxed{IG(x_1) > IG(x_2)} \Rightarrow$ ROOT NODE WILL TEST x_1 (MORE INFORMATION ON x_1)

ARTIN EXES (3)

Q8 Now we want to approx as an ANN with a single hidden layer, $l=1$ of 4 neurons.
We use cross entropy + logistic function. Biases initially $\emptyset$.
$W^l_{JK} = (K+(1-l)\cdot J)\cdot 0.1$. Compute the prediction for $\overset{x_1\,x_2\,x_3}{(0,1,1)}$ and the error w.r.t. $y=1$.



WEIGHTS ARE IN THE CONNECTIONS! BEFORE z^1_J

$$\underbrace{\underline{z}^1 = \begin{bmatrix} z^1_1 \\ z^1_2 \\ z^1_3 \\ z^1_4 \end{bmatrix}}_{\underline{z}^1} = \underbrace{\begin{bmatrix} W^1_{11} & W^1_{12} & W^1_{13} \\ W^1_{21} & W^1_{22} & W^1_{23} \\ W^1_{31} & W^1_{32} & W^1_{33} \\ W^1_{41} & W^1_{42} & W^1_{43} \end{bmatrix}}_{\underline{W}^1} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\underline{a}^\emptyset} + \underbrace{\begin{bmatrix} W^1_{10} \\ W^1_{20} \\ W^1_{30} \\ W^1_{40} \end{bmatrix}}_{\text{BIAS}} [1] \;!$$

$$z^1 = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} \emptyset \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} [1] = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

$$\underline{W}^2 = \begin{bmatrix} W^2_{11} & W^2_{12} & W^2_{13} & W^2_{14} \end{bmatrix} = \begin{bmatrix} \emptyset & 0.1 & 0.2 & 0.3 \end{bmatrix}$$

$$g(0.5) = \frac{1}{1+e^{-(0.5)}} = 0.622 \quad \sim \quad \underline{a}^1 = \begin{bmatrix} 0.622 \\ 0.622 \\ 0.622 \\ 0.622 \end{bmatrix}$$

REMEMBER TO PASS THE $\underline{z}$ THROUGH THE SIGMOID FUNCTION TO GET $\underline{a}$!

$$\underline{z}^2 = \underline{W}^2 \cdot \underline{a}^1 + \underbrace{[W^2_{10}]}_{(\emptyset - 1)0.1 = -0.1}[1] = 0.622(0.1+0.2+0.3) - 0.1 = 0.3732 - 0.1 = 0.2732$$

$$a^2 = g(z^2) = \frac{1}{1+e^{-(0.2732)}} = 0.5679 \qquad \Delta^L = a^L - y = 0.5679 - 1 = -0.4321$$

$$\left(\underline{\Delta}^l = (\underline{W}^l)^T \underline{\Delta}^l \circ g'(\underline{z}^{l-1}) \;;\; g'(\underline{z}^{l-1}) = g(\underline{z}^{l-1})\left(1 - g(\underline{z}^{l-1})\right).\right)$$

$$\Delta^1 = \begin{bmatrix} 0 \\ 0.1 \\ 0.2 \\ 0.3 \end{bmatrix} [-0.4321] \circ \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} = \begin{bmatrix} \emptyset \\ 0.025 \\ 0.05 \\ 0.075 \end{bmatrix} [-0.4321] = \begin{bmatrix} 0 \\ -0.0108 \\ -0.0216 \\ -0.0324 \end{bmatrix} = \underline{\Delta}^1$$