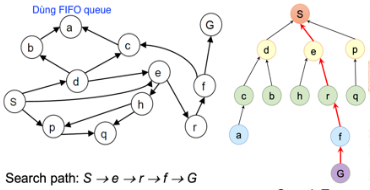
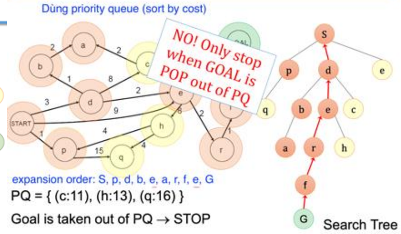
Breadth-first search: An example



Search path: $S \to e \to r \to f \to G$

Uniform-cost search: An example



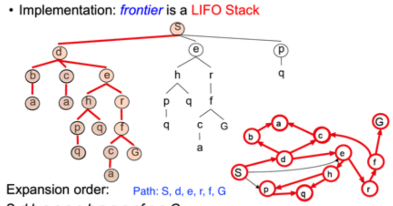
PQ = { (c:11), (h:13), (q:16) }

Goal is taken out of PQ → STOP

Search path: S → d → e → r → f → G, cost = 10

Depth-first search (DFS)

- Expand deepest unexpanded node
- Implementation: *frontier* is a LIFO Stack



Expansion order:
S,d,b,a,c,a,e,h,p,q,q,r,f,c,a,G

Comparison of BFS and DFS

	DFS	BFS
Space complexity	Linear space	Maybe the whole search space
Time complexity	Same, better on the average (many goals, no loops, and no infinite paths)	Same, better in worst-cases
In general	better if many goals, not many loops, and much better in terms of memory.	better if goal is not deep, infinite paths, many loops, or small search space

DFS in use

- The goal test is applied to each node when it is **generated** rather than when it is selected for expansion.
- Avoid repeated states by checking new states against those on the path from the root to the current node.

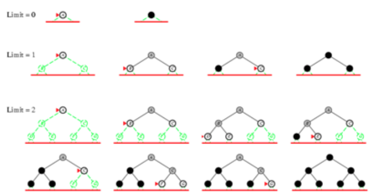
Iterative deepening search (IDS)

- General strategy, often used in combination with depth-first tree search to find the best depth limit

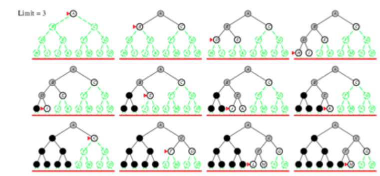
function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for *depth* = 0 **to** ∞ **do**
    result ← DEPTH-LIMITED-SEARCH(*problem, depth*)
    if *result* ≠ cutoff **then return** *result*

- Gradually increase the limit until a goal is found.
  - The depth limit reaches the depth d of the shallowest goal node.

Iterative deepening search (IDS)



Iterative deepening search (IDS)



A summary of uninformed search

- Comparison of uninformed algorithms (tree-search versions)

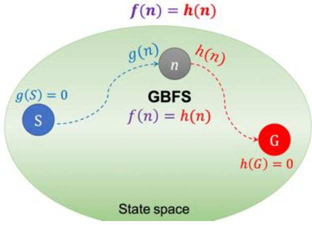
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Greedy best-first search

- Expand the node that appears to be closest to goal using

$$f(n) = h(n)$$

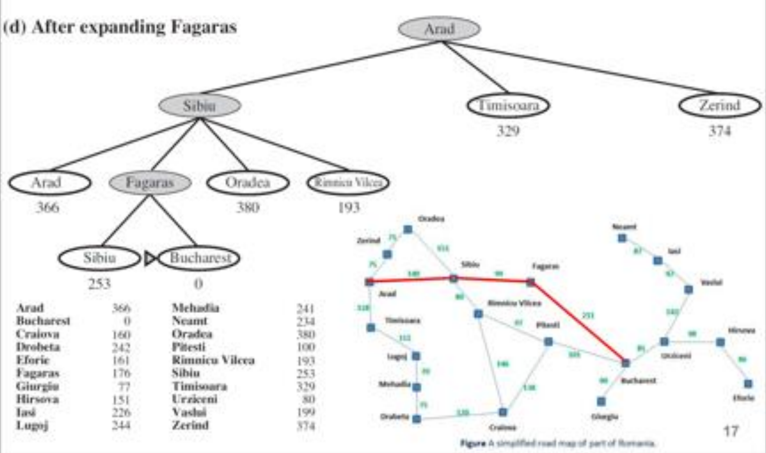


A* search

- The most widely known form of best-first search
- Use heuristic to guide search, but not only
- Avoid expanding paths that are already expensive
- Ensure to compute a path with minimum cost

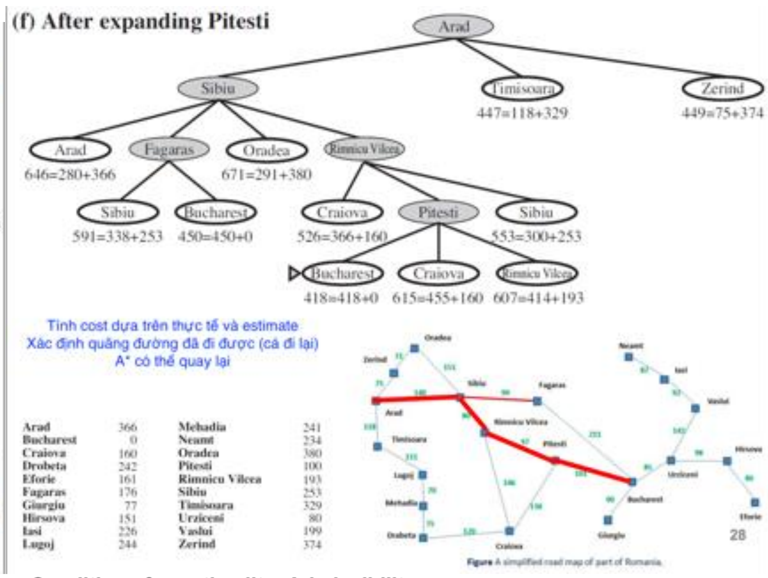
- Evaluate nodes by $f(n) = g(n) + h(n)$
  - where $g(n)$ is the cost to reach the node n and $h(n)$ is the cost to get from n to the goal
  - $f(n)$ = estimated cost of the cheapest solution through n

Greedy best-first search: An example



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure A simplified road map of part of Romania.

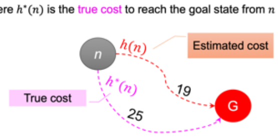


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure A simplified road map of part of Romania.

Conditions for optimality: Admissibility

- $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$
  - where $h^*(n)$ is the true cost to reach the goal state from n

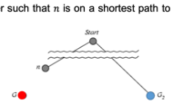


- Hence, $f(n)$ never overestimates the true cost of a solution along the current path through n.
  - $g(n)$ is the actual cost to reach n along the current path

Conditions for optimality: Admissibility

If $h(n)$ is admissible, A* using TREE-SEARCH is optimal

- Suppose some suboptimal goal G_2 has been generated and is in the frontier.
- Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.
- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
  $f(G_2) > f(G)$ (1)
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
  $f(n) \leq f(G)$ (2)
- From (1), (2): $f(G_2) > f(n)$ → A* will never select G_2 for expansion



Conditions for optimality: Consistency

If $h(n)$ is consistent, A* using GRAPH-SEARCH is optimal

- If $h(n)$ is consistent, the values of $f(n)$ along any path are non-decreasing.
  - Suppose n' is a successor of $n \to g(n') = g(n) + c(n, a, n')$
  - $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$
- Whenever A* selects a node n for expansion, the optimal path to that node has been found.
  - Proof by contradiction: There would have to be another frontier node n' on the optimal path from the start node to n (by the graph separation property)
  - f is nondecreasing along any path → $f(n') < f(n)$ → n' would have been selected first

Conditions for optimality: Admissibility

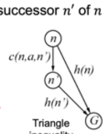
- $h(n)$ must be an admissible heuristic
  - Never overestimate the cost to reach the goal → optimistic
  - E.g., the straight-line distance h_{SLD}

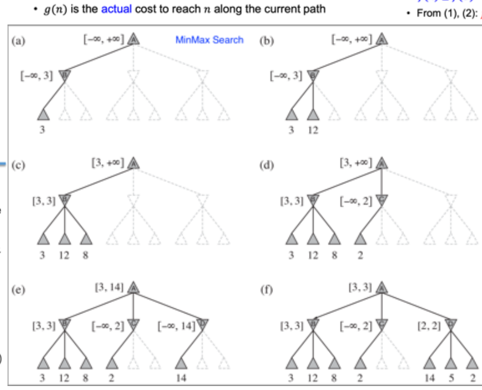
Conditions for optimality: Consistency

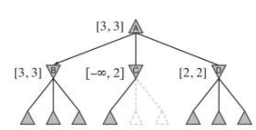
- Admissibility is insufficient for graph search.
  - The optimal path to a repeated state could be discard if it is not the first one selected.
- $h(n)$ is consistent if for every node n, every successor n' of n generated by any action a,

$$h(n) \leq c(n, a, n') + h(n')$$

- Every consistent heuristic is also admissible.

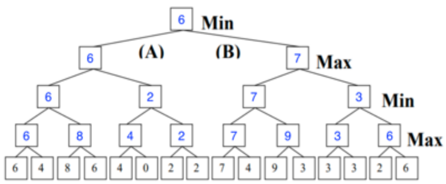






Another way to look at this is as a simplification of the formula for MINIMAX. Let the two unevaluated successors of node C have values x and y. Then the value of the root node is given by

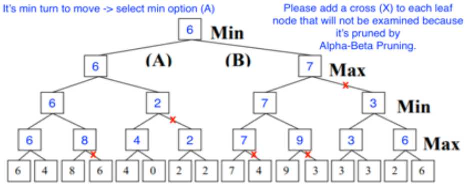
$$\begin{aligned}
\text{MINIMAX}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \quad \text{where } z = \min(2,x,y) \leq 2 \\
&= 3.
\end{aligned}$$



b. What's the best move for MIN? Answer (A or B): The best move for MIN is A.

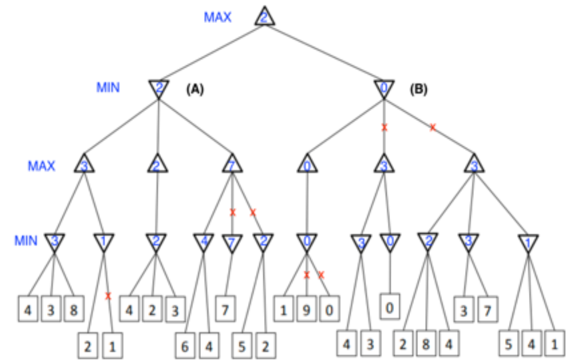
It's min turn to move -> select min option (A)

Please add a cross (X) to each leaf node that will not be examined because it's pruned by Alpha-Beta Pruning.



b. What's the best move for MAX? Answer (A or B): The best move for MAX is A.

c. Please add a cross (X) to each leaf node that will not be examined because it's pruned by Alpha-Beta Pruning. It's max turn to move -> select max value -> select A



Propositional logic: Semantics

- Each model specifies true/false for each proposition symbol.
  - E.g., $m_1 = \{P_{1,2} = false, P_{2,2} = false, P_{3,1} = true\}$, 8 possible models
- Rules for evaluating truth with respect to a model m

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

- Simple recursive process evaluates an arbitrary sentence.
  - E.g., $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (true \vee false) = true \wedge true = true$

Inference and Proofs

- **Proof**: A chain of conclusions leads to the desired goal
- Example sound rules of inference

Modus Ponens	Modus Tollens	AND-Introduction	AND-Elimination
$\alpha \Rightarrow \beta$, α ∴ β	$\alpha \Rightarrow \beta$, $\neg\beta$ ∴ $\neg\alpha$	α, β ∴ $\alpha \wedge \beta$	$\alpha \wedge \beta$ ∴ α

Inference rules: An example

KB
$P \wedge Q$
$P \Rightarrow R$
$Q \wedge R \Rightarrow S$

S?

No.	Sentences	Explanation
1	$P \wedge Q$	From KB
2	$P \Rightarrow R$	From KB
3	$Q \wedge R \Rightarrow S$	From KB
4	P	1 And-Elim
5	R	4,2 Modus Ponens
6	Q	1 And-Elim
7	$Q \wedge R$	5,6 And-Intro
8	S	3,7 Modus Ponens

Inference rules in Wumpus world

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

Proof: $\neg P_{1,2}$

- Bi-conditional elimination to R_2: R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- And-Elimination to R_6: R_7: $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$
- Logical equivalence for contrapositives: R_8: $\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$
- Modus Ponens with R_8 and the percept R_4: R_9: $\neg(P_{1,2} \vee P_{2,1})$
- De Morgan's rule: R_{10}: $\neg P_{1,2} \wedge \neg P_{2,1}$

Logical equivalence

- Two sentences, α and β, are logically equivalent if they are true in the same set of models.

$$\alpha \equiv \beta \ iff \ \alpha \models \beta \ and \ \beta \models \alpha$$

$$
\begin{array}{rcll}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) & \text{commutativity of } \wedge \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) & \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) & \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) & \text{associativity of } \vee \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) & \text{De Morgan} \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) & \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) & \text{distributivity of } \vee \text{ over } \wedge
\end{array}
$$

35

Conversion to CNF

1. Eliminate $\Leftrightarrow$: $\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. Eliminate $\Rightarrow$: $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
3. The operator $\neg$ appears only in literals: "move $\neg$ inwards"
   - $\neg\neg\alpha \equiv \alpha$ (double-negation elimination)
   - $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$ (De Morgan)
   - $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$ (De Morgan)
4. Apply the distributivity law to distribute $\vee$ over $\wedge$
   - $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$

Horn clauses and Definite clauses

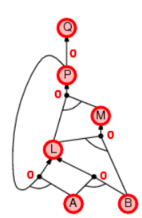
- Definite clause: a disjunction of literals of which exactly one is positive.
  - E.g., $\neg P \vee \neg Q \vee R$ is a definite clause, whereas $\neg P \vee Q \vee R$ is not.
- Horn clause: a disjunction of literals of which at most one is positive.
  - All definite clauses are Horn clauses
- Goal clause: clauses with no positive literals
- Horn clauses are closed under resolution
  - Resolving two Horn clauses will get back a Horn clause.

Backus normal form (BNF)

$$
\begin{array}{rcl}
CNFSentence & \to & Clause_1 \wedge \cdots \wedge Clause_n \\
Clause & \to & Literal_1 \vee \cdots \vee Literal_m \\
Literal & \to & Symbol \mid \neg Symbol \\
Symbol & \to & P \mid Q \mid R \mid \ldots \\
HornClauseForm & \to & DefiniteClauseForm \mid GoalClauseForm \\
DefiniteClauseForm & \to & (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow Symbol \\
GoalClauseForm & \to & (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow False
\end{array}
$$

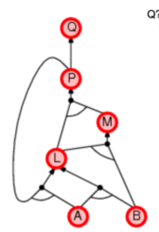
Forward chaining: An example

$$
\begin{array}{l}
P \Rightarrow Q \\
L \wedge M \Rightarrow P \\
B \wedge L \Rightarrow M \\
A \wedge P \Rightarrow L \\
A \wedge B \Rightarrow L \\
A \\
B
\end{array}
$$



Backward chaining: An example

$$
\begin{array}{l}
P \Rightarrow Q \\
L \wedge M \Rightarrow P \\
B \wedge L \Rightarrow M \\
A \wedge P \Rightarrow L \\
A \wedge B \Rightarrow L \\
A \\
B
\end{array}
$$



Q? ✓
P? ✓
L? ✓
A? ✓
B? ✓
M? ✓
L? ✓
B? ✓

Improvements in DPLL

- **Early termination:** A clause is true if any literal is true, and a sentence is false if any clause is false.
  - Avoid examination of entire subtrees in the search space
  - E.g., $(A \vee B) \wedge (A \vee C)$ is true if A is true, regardless B and C
- **Pure symbol heuristic:** A pure symbol always appears with the same "sign" in all clauses.
  - E.g., $(A \vee \neg B), (\neg B \vee \neg C), (A \vee C)$, A and B are pure, C is impure.
  - Make a pure symbol true → Doing so never make a clause false
- **Unit clause heuristic:** there is only one literal in the clause and thus this literal must be true
  - **Unit propagation:** if the model contains $B = true$ then $(\neg B \vee \neg C)$ simplifies to a unit clause $\neg C$ → C must be false (so that $\neg C$ is true) → A must be true (so that $A \vee C$ is true)

The DPLL procedure

Davis–Putnam–Logemann–Loveland (DPLL)

```
function DPLL-SATISFIABLE?(s) returns true or false
   inputs: s, a sentence in propositional logic
   clauses ← the set of clauses in the CNF representation of s
   symbols ← a list of the proposition symbols in s
   return DPLL(clauses, symbols,{ })
```

```
function DPLL(clauses, symbols, model) returns true or false
  if every clause in clauses is true in model then return true      } 1. Early
  if some clause in clauses is false in model then return false     } Termination
  P, value ←(2) FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ←(3) FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
     DPLL(clauses, rest, model ∪ {P=false}))
```

Quantifiers: Universal quantification

- Expressions of general rules   ∀<variables> <sentence>
  - E.g., "All kings are persons.": $\forall x\ King(x) \Rightarrow Person(x)$
  - E.g., "Students of FIT are smart.": $\forall x\ Student(x, FIT) \Rightarrow Smart(x)$

∀x P is true in a model m iff P is true with x being each possible object in the model.

- It is equivalent to the conjunction of instantiations of P.

Student(Lan, FIT) ⇒ Smart(Lan)
∧ Student(Tuan, FIT) ⇒ Smart(Tuan)
∧ Student(Long, FIT) ⇒ Smart(Long)
∧ …

Quantifiers: Existential quantification

- Expressions of "some cases"   ∃<variables> <sentence>
  - E.g., "Some students of FIT are smart." $\exists x\ Student(x, FIT) \wedge Smart(x)$

∃x P is true in a model m iff P is true with x being some possible object in the model.

- It is equivalent to the disjunction of instantiations of P.

Student(Lan, FIT) ∧ Smart(Lan)
∨ Student(Tuan, FIT) ∧ Smart(Tuan)
∨ Student(Long, FIT) ∧ Smart(Long)
∨ …

Quantifiers: A common mistake to avoid

- Typically, ⇒ is the main connective with ∀
  - The conclusion of the rule just for those objects for whom the premise is true
  - It says nothing at all about individuals for whom the premise is false.
- **Common mistake:** using ∧ as the main connective with ∀
  - $\forall x\ Student(x, FIT) \wedge Smart(x)$
  - It means "Everyone is a student of FIT and Everyone is smart."

FOL definite clause: An example

- Consider the following problem

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$\exists\ x\ Owns(Nono, x) \wedge Missile(x)$ → $Owns(Nono, M_1)$, $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$

$Enemy(Nono, America)$

64

CNF for First-order logic

- First-order resolution requires that sentences be in CNF.
- For example, the sentence

$\forall x\ American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

becomes, in CNF,

$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z) \vee Criminal(x)$

- *Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.*
  - The CNF sentence will be unsatisfiable just when the original sentence is unsatisfiable → perform proofs by contraction.

Quantifiers: A common mistake to avoid

- Typically, ∧ is the main connective with ∃
- **Common mistake:** using ⇒ as the main connective with ∃
  - $\exists x\ Student(x, FIT) \Rightarrow Smart(x)$
  - It is true even with anyone who is not at FIT.

Conversion to CNF

Everyone who loves all animals is loved by someone.

$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

1. Eliminate implications

$\forall x\ [\neg\forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

2. Move $\neg$ inwards: $\neg\forall x\ p \equiv \exists x \neg p$, $\neg\exists x\ p \equiv \forall x\ \neg p$

$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)]$

$\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

Conversion to CNF

Everyone who loves all animals is loved by someone.

$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

3. Standardize variables: each quantifier uses a different one

$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$

4. Skolemize: remove existential quantifiers by elimination
   - Simple case: translate $\exists x\ P(x)$ into $P(A)$, where A is a new constant.
     - However, $\forall x[Animal(A) \wedge \neg Loves(x,A)] \vee [Loves(B,x)]$ has an entirely different meaning.
   - The arguments of the **Skolem function** are all universally quantified variables in whose scope the existential quantifier appears.

$\forall x\ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$

Conversion to CNF

Everyone who loves all animals is loved by someone.

$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

5. Drop universal quantifiers

$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$

6. Distribute $\vee$ over $\wedge$

$[Animal(F(x)) \vee Loves(G(x),x)] \wedge$
$[\neg Loves(x,F(x)) \vee Loves(G(x),x)]$

Resolution: Another example

Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by no one.
Jack loves all animals.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
B. $\forall x\ [\exists z\ Animal(z) \wedge Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$
C. $\forall x\ Animal(x) \Rightarrow Loves(Jack,x)$
D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
E. $Cat(Tuna)$
F. $\forall x\ Cat(x) \Rightarrow Animal(x)$
G. $\neg Kills(Curiosity,Tuna)$

Resolution: Another example

Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by no one.
Jack loves all animals.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

A. $Animal(F(x)) \vee Loves(G(x),x)$
$\neg Loves(x,F(x)) \vee Loves(G(x),x)$
B. $\neg Loves(y,x) \vee \neg Animal(z) \vee \neg Kills(x,z)$
C. $\neg Animal(x) \vee \neg Loves(Jack,x)$
D. $Kills(Jack,Tuna) \vee Kills(Curiousity,Tuna)$
E. $Cat(Tuna)$
F. $\neg Cat(x) \vee Animal(x)$
G. $\neg Kills(Curiosity,Tuna)$

Full joint distribution with BN

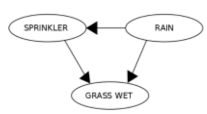
- An entry in the joint distribution is the probability of a variable assignment, such as $\mathbf{P}(X_1 = x_1 \wedge \cdots \wedge X_n = x_n)$.

$$\mathbf{P}(x_1, \ldots, x_n) = \prod_{i=1}^{n} \mathbf{P}(X_i \mid parent(X_i))$$

  - where $parent(X_i)$ denotes the values of $Parent(X_i)$ that appear in $x_1, \ldots, x_n$.
- Thus, it is the product of the appropriate elements of the CPTs in the Bayesian network.
- A Bayesian network can be used to answer any query, by summing all the relevant joint entries.

The wet grass example

RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99



RAIN T	RAIN F
0.2	0.8

SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

G = Grass wet (True/False)
S = Sprinkler turned on (True/False)
R = Raining (True/False)

The wet grass example

- What is the probability that it is raining, given the grass is wet?

$$P(R=T|G=T) = \frac{P(G=T,R=T)}{P(G=T)} = \frac{\sum_{S\in\{T,F\}} P(G=T,S,R=T)}{\sum_{S,R\in\{T,F\}} P(G=T,S,R)}$$

- Using the expansion for the joint probability function $P(G,S,R)$ and the conditional probabilities from the CPTs stated in the diagram

$$P(G=T,S=T,R=T) = P(G=T \mid S=T,R=T)\,P(S=T \mid R=T)P(R=T)$$
$$= 0.99 \times 0.01 \times 0.2 = 0.00198$$

- The numerical results (subscripted by the associated variable values) are

$$P(R=T \mid G=T) = \frac{0.00198_{TTT} + 0.1584_{TFT}}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0.0_{TFF}}$$
$$= \frac{891}{24\ldots} \approx 35.77\%$$

Inference by enumeration

- A query can be answered by computing sums of products of conditional probabilities from the Bayesian network.

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \mathbf{P}(X,\mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X,\mathbf{e},\mathbf{y})$$

  - where α stands for the constant denominator term, which is usually simplified during calculation.

Inference by enumeration

- Consider the following query

$$P(Burglary \mid JohnCalls = true, MaryCalls = true)$$

- The hidden variables are $Earthquake$ and $Alarm$.
- Using initial letters for the variables, we have

$$P(B \mid j,m) = \alpha\,\mathbf{P}(B,j,m) = \alpha \sum_e \sum_a \mathbf{P}(B,j,m,e,a)$$

- For simplicity, we do this for $Burglary = true$.

$$P(b \mid j,m) = \alpha \sum_e \sum_a P(b)\,P(e)\,P(a \mid b,e)P(j \mid a)\,P(m \mid a)$$

- **Complexity:** $O(n2^n)$ for a network of n Boolean variables

Variable elimination: Factorization

- Consider the burglary network. We evaluate the following

$$\mathbf{P}(B \mid j,m) = \alpha \underbrace{\mathbf{P}(B)}_{\mathbf{f}_1(B)} \sum_e \underbrace{P(e)}_{\mathbf{f}_2(E)} \sum_a \underbrace{\mathbf{P}(a \mid B,e)}_{\mathbf{f}_3(A,B,E)} \underbrace{P(j \mid a)}_{\mathbf{f}_4(A)} \underbrace{P(m \mid a)}_{\mathbf{f}_5(A)}$$

factor

where
  - $\mathbf{f}_4(A) = \begin{pmatrix} P(j \mid a) \\ P(j \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0.90 \\ 0.05 \end{pmatrix}$, $\mathbf{f}_5(A) = \begin{pmatrix} P(m \mid a) \\ P(m \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0.70 \\ 0.01 \end{pmatrix}$
  - $\mathbf{f}_3(A,B,E)$ is a 2×2×2 matrix
- In terms of factors, the query expression is written as

$$\mathbf{P}(B \mid j,m) = \alpha\,\mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \sum_a \mathbf{f}_3(A,B,E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A)$$

  - where × is the pointwise product operation.

Variable elimination: Factorization

- First, we sum out A from the product of $\mathbf{f}_3$, $\mathbf{f}_4$, and $\mathbf{f}_5$.

$$\mathbf{f}_6(B,E) = \sum_a \mathbf{f}_3(A,B,E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A)$$
$$= \mathbf{f}_3(a,B,E) \times \mathbf{f}_4(a) \times \mathbf{f}_5(a) + \mathbf{f}_3(\neg a,B,E) \times \mathbf{f}_4(\neg a) \times \mathbf{f}_5(\neg a)$$

Now we are left with $\mathbf{P}(B \mid j,m) = \alpha\,\mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B,E)$

- Next, we sum out E from the product of $\mathbf{f}_2$ and $\mathbf{f}_6$

$$\mathbf{f}_7(B) = \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B,E) = \mathbf{f}_2(e) \times \mathbf{f}_6(B,e) + \mathbf{f}_2(\neg e) \times \mathbf{f}_6(B,\neg e)$$

- The following expression can be evaluated by taking pointwise product and normalizing the result.

$$\mathbf{P}(B \mid j,m) = \alpha\,\mathbf{f}_1(B) \times \mathbf{f}_7(B)$$

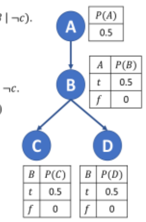
Variable elimination: An example

- Consider the following network. Calculate $P(B \mid \neg c)$.
- $\mathbf{P}(B \mid \neg c) = \alpha \underbrace{P(\neg c \mid B)}_{\mathbf{f}_1(B)} \times \sum_a \underbrace{P(B \mid a)}_{\mathbf{f}_2(A,B)} \times \underbrace{P(a)}_{\mathbf{f}_3(A)}$

factor

- Irrelevant variable: D. Observed variable: $C = \neg c$.
- Sum out A to have $\boldsymbol{f}_4(B) = \sum_a P(B \mid a) \times P(a)$
- Join $\boldsymbol{f}_1$ and $\boldsymbol{f}_4$: $\boldsymbol{f}_5(B,\neg c) = \boldsymbol{f}_1(B) \times \boldsymbol{f}_4(B)$
- Finally, we have $\mathbf{P}(B \mid \neg c) = \alpha\,\boldsymbol{f}_5(B,\neg c)$
- Assume that $B = b$. Normalize for B:

$$\mathbf{P}(b \mid \neg c) = \frac{\boldsymbol{f}_5(b,\neg c)}{\boldsymbol{f}_5(b,c) + \boldsymbol{f}_5(\neg b,\neg c)}$$



$P(A)$
0.5

A	$P(B)$
t	0.5
f	0

B	$P(C)$
t	0.5
f	0

B	$P(D)$
t	0.5
f	0

ID3 Decision tree algorithm

1. The remaining examples are **all** positive (or **all** negative), → DONE, it is possible to answer Yes or No.
   - E.g., in Figure (b), None and Some branches
2. There are **some** positive and **some** negative examples → choose the **best** attribute to split them
   - E.g., in Figure (b), Hungry is used to split the remaining examples
3. No examples left at a branch → return a default value.
   - No example has been observed for a combination of attribute values
   - The default value is calculated from the plurality classification of all the examples that were used in constructing the node's parent.
   - These are passed along in the variable parent examples
4. No attributes left but both positive and negative examples → return the plurality classification of remaining ones.
   - Examples of the same description, but different classifications
   - Usually an error or noise in the data, nondeterministic domain, or no observation of an attribute that would distinguish the examples.

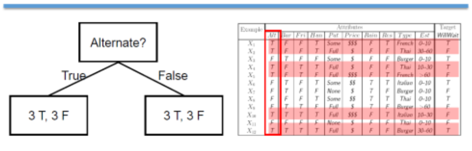
A purity measure with entropy

- **Entropy** is a measure of the uncertainty of a random variable V with values v_k. *An indicator of how messy your data is*

$$H(V) = \sum_k P(v_k)\log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k)\log_2 P(v_k)$$

  - v_k is a class in V (e.g., yes/no in binary classification)
  - $P(v_k)$ is the proportion of the number of elements in class v_k to the number of elements in V

ID3 Decision tree: An example

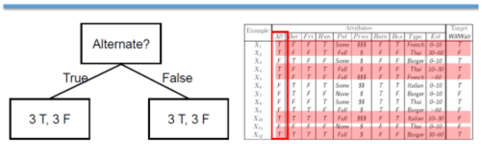


- Calculate Average Entropy of attribute Alternate

$$AE_{Alternate} = P(Alt = T) \times H(Alt = T) + P(Alt = F) \times H(Alt = F)$$

$$AE_{Alternate} = \frac{6}{12}\left[-\left(\frac{3}{6}\log_2\frac{3}{6}\right) - \left(\frac{3}{6}\log_2\frac{3}{6}\right)\right] + \frac{6}{12}\left[-\left(\frac{3}{6}\log_2\frac{3}{6}\right) - \left(\frac{3}{6}\log_2\frac{3}{6}\right)\right] = 1$$

ID3 Decision tree: An example



- **Information Gain** is the difference in entropy from before to after the set S is split on the selected attribute.

$$IG(Alternate, S) = H(S) - AE_{Alternate} = 1 - 1 = 0$$

You are stranded on a deserted island. Mushrooms of various types grow widely all over the island, but no other food is anywhere to be found. Some of the mushrooms have been determined as poisonous and others as not (determined by your former companions' trial and error). You are the only one remaining on the island. You have the following data to consider:

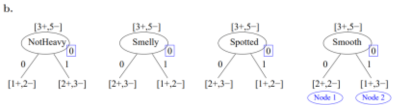
Example	NotHeavy	Smelly	Spotted	Smooth	Edible
A	1	0	0	0	1
B	1	0	1	0	1
C	0	1	0	1	1
D	0	0	0	1	0
E	1	1	1	0	0
F	1	0	1	1	0
G	1	0	0	1	0
H	0	1	0	0	0
U	0	1	1	1	?
V	1	1	0	1	?
W	1	1	0	0	?

You know whether or not mushrooms A through H are poisonous, but you do not know about U through W.

a.

$$H_{Edible} = H[3+,5-] \overset{def.}{=} -\frac{3}{8}\cdot\log_2\frac{3}{8} - \frac{5}{8}\cdot\log_2\frac{5}{8} = \frac{3}{8}\cdot\log_2\frac{8}{3} + \frac{5}{8}\cdot\log_2\frac{8}{5}$$
$$= \frac{3}{8}\cdot 3 - \frac{3}{8}\cdot\log_2 3 + \frac{5}{8}\cdot 3 - \frac{5}{8}\cdot\log_2 5 = 3 - \frac{3}{8}\cdot\log_2 3 - \frac{5}{8}\cdot\log_2 5$$
$$\approx 0.9544$$

b.



c.

$$H_{0/Smooth} \overset{def.}{=} \frac{4}{8}H[2+,2-] + \frac{4}{8}H[1+,3-] = \frac{1}{2}\cdot 1 + \frac{1}{2}\left(\frac{1}{4}\log_2\frac{4}{1} + \frac{3}{4}\log_2\frac{4}{3}\right)$$
$$= \frac{1}{2} + \frac{1}{2}\left(\frac{1}{4}\cdot 2 + \frac{3}{4}\cdot 2 - \frac{3}{4}\log_2 3\right) = \frac{1}{2} + \frac{1}{2}\left(2 - \frac{3}{4}\log_2 3\right)$$
$$= \frac{1}{2} + 1 - \frac{3}{8}\log_2 3 = \frac{3}{2} - \frac{3}{8}\log_2 3 \approx 0.9056$$

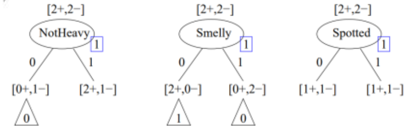
$$IG_{0/Smooth} \overset{def.}{=} H_{Edible} - H_{0/Smooth}$$
$$= 0.9544 - 0.9056 = 0.0488$$

d.

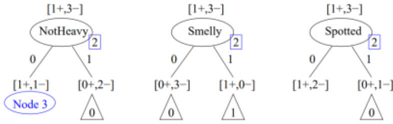
$$H_{0/NotHeavy} \overset{def.}{=} \frac{3}{8}H[1+,2-] + \frac{5}{8}H[2+,3-]$$
$$= \frac{3}{8}\left(\frac{1}{3}\log_2\frac{3}{1} + \frac{2}{3}\log_2\frac{3}{2}\right) + \frac{5}{8}\left(\frac{2}{5}\log_2\frac{5}{2} + \frac{3}{5}\log_2\frac{5}{3}\right)$$
$$= \frac{3}{8}\left(\frac{1}{3}\log_2 3 + \frac{2}{3}\log_2 3 - \frac{2}{3}\cdot 1\right) + \frac{5}{8}\left(\frac{2}{5}\log_2 5 - \frac{2}{5}\cdot 1 + \frac{3}{5}\log_2 5 - \frac{3}{5}\log_2 3\right)$$
$$= \frac{3}{8}\left(\log_2 3 - \frac{2}{3}\right) + \frac{5}{8}\left(\log_2 5 - \frac{3}{5}\log_2 3 - \frac{2}{5}\right)$$
$$= \frac{3}{8}\log_2 3 - \frac{2}{8} + \frac{5}{8}\log_2 5 - \frac{3}{8}\log_2 3 - \frac{2}{8}$$
$$= \frac{5}{8}\log_2 5 - \frac{4}{8} \approx 0.9512$$

$$\Rightarrow IG_{0/NotHeavy} \overset{def.}{=} H_{Edible} - H_{0/NotHeavy} = 0.9544 - 0.9512 = 0.0032,$$
$$IG_{0/NotHeavy} = IG_{0/Smelly} = IG_{0/Spotted} = 0.0032 < IG_{0/Smooth} = 0.0488$$

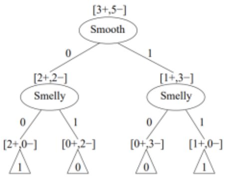
Node 1: Smooth = 0



Node 2: Smooth = 1



The resulting ID3 Tree



IF (*Smooth* = 0 AND *Smelly* = 0) OR (*Smooth* = 1 AND *Smelly* = 1)
THEN *Edibile*;
ELSE ¬*Edible*;

Classification of *test instances*:

U	$Smooth = 1,\ Smelly = 1 \Rightarrow Edible = 1$
V	$Smooth = 1,\ Smelly = 1 \Rightarrow Edible = 1$
W	$Smooth = 0,\ Smelly = 1 \Rightarrow Edible = 0$

- **Foundation:** Based on **Bayes' Theorem**

Likelihood · Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

Posterior Probability · Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- Total Probability Theorem: $P(B) = \sum_{i=1}^{M} P(B|A_i)P(A_i)$
- Let $\mathbf{X}$ be a data sample ("evidence") with unknown class label and H be a hypothesis that $\mathbf{X}$ belongs to class C
- **Bayes' Theorem:** $P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$
- Classification is to determine $P(H \mid \mathbf{X})$, the probability that the hypothesis H holds given the observed data sample $\mathbf{X}$.

- $P(H)$ (prior probability): the initial probability
  - E.g., $\mathbf{X}$ will buy computer, regardless of age, income, …
- $P(\mathbf{X})$: the probability that sample data is observed
  - E.g., $\mathbf{X}$ is 31..40 and has a medium income, regardless of the buying
- $P(\mathbf{X} \mid H)$ (likelihood): the probability of observing the sample $\mathbf{X}$, given that the hypothesis holds
  - E.g., given that $\mathbf{X}$ will buy computer, the probability that $\mathbf{X}$ is 31..40 and has a medium income
- $P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$ (posterior probability)
  - E.g., given that $\mathbf{X}$ is 31..40 and has a medium income, the probability that $\mathbf{X}$ will buy computer

- Informally, $P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$ can be viewed as
  $posteriori = likelihood * prior / evidence$
- $\mathbf{X}$ belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|\mathbf{X})$ for all the k classes

- **Practical difficulty**
  - Require initial knowledge of many probabilities
  - Significant computational cost involved

Classification with Bayes' Theorem

- Let D be a training set of tuples and associated class labels
- Each tuple is represented by a n-attribute $\mathbf{X} = (x_1, x_2, \dots, x_n)$
- Suppose there are m classes $C_1, C_2, \dots, C_m$
- Classification is to derive the maximum posteriori $P(C_i|\mathbf{X})$ from **Bayes' theorem**

$$P(C_i \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid C_i)P(C_i)}{P(\mathbf{X})}$$

- $P(X)$ is constant for all classes, only $P(\mathbf{X} \mid C_i)P(C_i)$ needs to be maximized.

Naïve Bayesian classification

- Class-conditional independence: There are no dependence relationships **among the attributes**
- The **naïve Bayesian classification** formula is written as

$$P(\mathbf{X} \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) = P(x_1 \mid C_i) \times P(x_2 \mid C_i) \times \cdots \times P(x_n \mid C_i)$$

  - A_k is categorical: $P(x_k \mid C_i)$ is the number of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
  - A_k is continuous: $P(x_k \mid C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$ with the Gaussian distribution $g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- Count class distributions only → computation cost reduced

Naïve Bayesian classification: An example

P(buys_computer = "yes")	9/14
P(buys_computer = "no")	5/14

	buys_computer = "yes"	buys_computer = "no"
age = "<=30"	2/9	3/5
age = "31…40"	4/9	0/5
age = ">40"	3/9	2/5
income = "low"	3/9	1/5
income = "medium"	4/9	2/5
income = "high"	2/9	2/5
student = "yes"	6/9	1/5
student = "no"	3/9	4/5
credit_rating = "fair"	6/9	2/5
credit_rating = "excellent"	3/9	3/5

age	income	student	credit_rating	buys_computer
<=30	medium	yes	fair	?

- $P(\mathbf{X}|C_i)$
  - $P(\mathbf{X} \mid buys\_computer = \text{"yes"}) = 2/9 * 4/9 * 6/9 * 6/9 = 0.044$
  - $P(\mathbf{X} \mid buys\_computer = \text{"no"}) = 3/5 * 2/5 * 1/5 * 2/5 = 0.019$
- $P(\mathbf{X}|C_i) * P(C_i)$
  - $P(\mathbf{X} \mid buys\_computer = \text{"yes"}) * P(buys\_computer = \text{"yes"}) = 0.028$
  - $P(\mathbf{X} \mid buys\_computer = \text{"no"}) * P(buys\_computer = \text{"no"}) = 0.007$
- $P(C_i \mid \mathbf{X})$
  - $P(buys\_computer = \text{"yes"} \mid \mathbf{X}) = 0.8$
  - $P(buys\_computer = \text{"no"} \mid \mathbf{X}) = 0.2$

Therefore, X belongs to class ("buys_computer = yes")

Avoiding the zero-probability issue

- **Laplacian correction** (or Laplacian estimator)

$$P(C_i) = \frac{|C_i| + 1}{|D| + m} \qquad P(x_k \mid C_i) = \frac{|x_k \cup C_i| + 1}{|C_i| + r}$$

  - where m is the number of classes, $|x_k \cup C_i|$ denotes the number of tuples contains both $A_k = x_k$ and C_i, and r is the number of values of attribute A_k
- The "corrected" probability estimates are close to their "uncorrected" counterparts

Naïve Bayesian classification: An example

P(buys_computer = "yes")	10/16
P(buys_computer = "no")	6/16

	buys_computer = "yes"	buys_computer = "no"
age = "<=30"	3/12	4/8
age = "31…40"	5/12	1/8
age = ">40"	4/12	3/8
income = "low"	4/12	2/8
income = "medium"	5/12	3/8
income = "high"	3/12	3/8
student = "yes"	7/11	2/7
student = "no"	4/11	5/7
credit_rating = "fair"	7/11	3/7
credit_rating = "excellent"	4/11	4/7

Naïve Bayesian classification: An example

age	income	student	credit_rating	buys_computer
31..40	medium	yes	fair	?

- $P(\mathbf{X}|C_i)$
  - $P(\mathbf{X} \mid buys\_computer = \text{"yes"}) = 5/12 * 5/12 * 7/11 * 7/11 = 0.070$
  - $P(\mathbf{X} \mid buys\_computer = \text{"no"}) = 1/8 * 3/8 * 2/7 * 3/7 = 0.006$
- $P(\mathbf{X}|C_i) * P(C_i)$
  - $P(\mathbf{X} \mid buys\_computer = \text{"yes"}) * P(buys\_computer = \text{"yes"}) = 0.044$
  - $P(\mathbf{X} \mid buys\_computer = \text{"no"}) * P(buys\_computer = \text{"no"}) = 0.002$
- $P(C_i \mid \mathbf{X})$
  - $P(buys\_computer = \text{"yes"} \mid \mathbf{X}) = 0.953$
  - $P(buys\_computer = \text{"no"} \mid \mathbf{X}) = 0.047$

Therefore, X belongs to class ("buys_computer = yes")

Perceptron learning rule

- Step 1 – Initialization: Initial weights $w_1, w_2, \dots, w_n$ and threshold θ are randomly assigned to small numbers (usually in $[-0.5, 0.5]$, but not restricted to).
- Step 2 – Activation: At iteration p, apply the p^{th} example, which has inputs $x_1(p), x_2(p), \dots, x_n(p)$ and desired output $Y_d(p)$, and calculate the actual output

$$Y(p) = \sigma\left(\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x \ge 0 \\ 0 & \text{if } x < 0 \end{cases}$$

where n is the number of perceptron inputs and $step$ is the activation function

- Step 3 – Weight training
  - Update the weights w_i: $w_i(p+1) = w_i(p) + \Delta w_i(p)$
    where $\Delta w_i(p)$ is the weight correction at iteration p
  - The **delta rule** determines how to adjust the weights: $\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$
    where α is the learning rate $(0 < \alpha < 1)$ and $e(p) = Y_d(p) - Y(p)$
- Step 4 – Iteration: Increase iteration p by one, go back to Step 2 and repeat the process until convergence.

Back-propagation learning rule

- Step 1 – Initialization: Initial weights and thresholds are assigned to random numbers.
  - The numbers may be uniformly distributed in the range $\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right)$ (Haykin, 1999), where F_i is the total number of inputs of neuron
  - The weight initialization is done on a neuron-by-neuron basis
- Step 2 – Activation: At iteration p, apply the p^{th} example, which has inputs $x_1(p), x_2(p), \dots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p), \dots, y_{d,l}(p)$.
  - (a) Calculate the actual output, from n inputs, of neuron j in the hidden layer.

$$y_j(p) = \sigma\left(\sum_{i=1}^{n} x_i(p) w_{ij}(p) - \theta_j\right) \qquad \sigma(x) = \frac{1}{1 + e^{-x}}$$

  - (b) Calculate the actual output, from k inputs, of neuron m in the hidden layer.

$$y_k(p) = \sigma\left(\sum_{j=1}^{m} y_j(p) w_{jk}(p) - \theta_k\right)$$

Back-propagation learning rule

- Step 3 – Weight training: Update the weights in the back-propagation network and propagate backward the errors associated with output neurons.
  - (a) Calculate the error gradient for neuron k in the output layer

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times \underbrace{[y_{d,k}(p) - y_k(p)]}_{e_k(p)}$$

    Calculate the weight corrections: $\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$

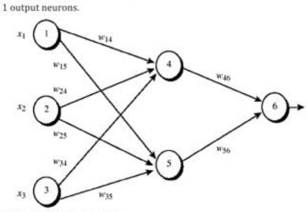
    Update the weights at the output neurons: $w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$
  - (b) Calculate the error gradient for neuron j in the hidden layer

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) w_{jk}(p)$$

    Calculate the weight corrections: $\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p)$

    Update the weights at the hidden neurons: $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$
- Step 4: Iteration: Increase iteration p by one, go back to Step 2 and repeat the process until the selected error criterion is satisfied.

Problem. Consider the following neuron network, which includes 3 input neurons, 2 hidden neurons and 1 output neurons.



Initial input, weight and bias values are

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

The expected output value is 1. The learning rate is 0.9
Knowing that the actual output at some neuron j is calculated as follows.

$$y_j(p) = \text{sigmoid}\left[\sum_{i=1}^{n} x_i(p) \times w_{ij}(p) + \theta_j\right]$$

$$y_j(p) = \text{sigmoid}\left[\sum_{i=1}^{n} x_i(p) \times w_{ij}(p) + \theta_j\right]$$

where n is the number of inputs of neuron j, w_{ij} is the corresponding link from a neuron i in the previous layer to neuron j, and θ_j is the bias at neuron j.

Present all calculations required to perform the backpropagation once (i.e., one forward pass and one backward pass) on the given neural network in the following cases

a) Ignore all biases *(precision to 3 decimal places).*

Ignore all biases – Forward

Output at neuron 4	0.426
Output at neuron 5	0.475
Output at neuron 6	0.527

Ignore all biases – Backward

Error gradient at neuron 6	0.118
Error gradient at neuron 5	-0.006

Error gradient at neuron 4	-0.009
Update w46	-0.255
Update w56	-0.150
Update w14	0.192
Update w15	-0.305
Update w24	0.400
Update w25	0.100
Update w34	-0.508
Update w35	0.195

b) Consider all biases such that each bias is treated as a neuron and thus it will be also updated *(precision to 3 decimal places).*

(Consider all biases – Forward

Output at neuron 4	0.332
Output at neuron 5	0.525
Output at neuron 6	0.552

Consider all biases – Backward

Error gradient at neuron 6	0.111
Error gradient at neuron 5	-0.006
Error gradient at neuron 4	-0.007
Update w46	-0.267
Update w56	-0.148
Update w14	0.193
Update w15	-0.305
Update w24	0.400
Update w25	0.100
Update w34	-0.507
Update w35	0.195
Update bias 6	-0.407
Update bias 5	0.195
Update bias 4	0.200