Atria Institute of Technology

Department of Information Science and Engineering

Bengaluru-560024

ACADEMIC YEAR: 2021-2022
ODD SEMESTER NOTES

Semester	**:**	**7th Semester**
Subject Name	**:**	**Artificial Intelligence and Machine Learning**
Subject Code	**:**	**18CS71**
Faculty Name	**:**	**Dr .Jayasudha K**

Artificial Intelligence andMachine Learning (AI & ML)

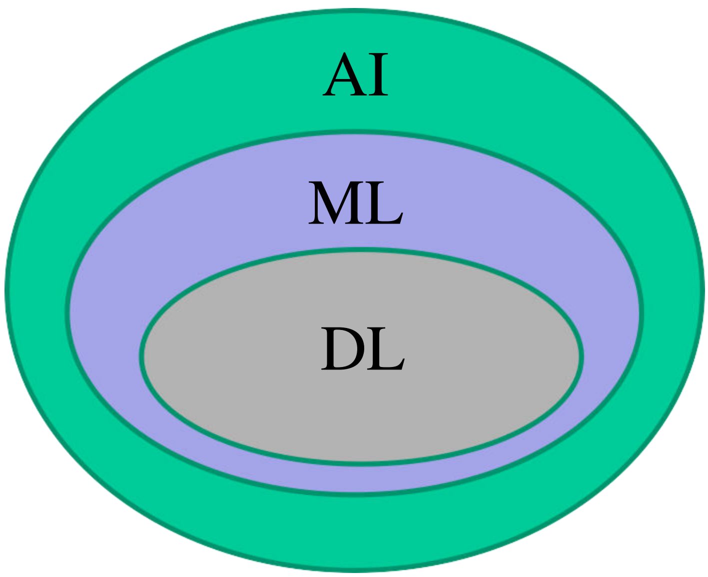
What is AI?

- Intelligence: "ability to learn, understand and think" (Oxford dictionary)
- Artificial Intelligence: is the study of how to make computers make things which at the moment people do better.
- Agent: Agents in AI sense the environment through sensors and act through actuators with properties like knowledge, belief, intention etc
- Logical Reasoning: It is a form of thinking in which premises and relations are used in rigorous mannerto infer conclusions.

- Examples: Speech recognition, Smell, Face, Object, Intuition, Inference, Learning new skills.

What is Machine Learning(ML)?

ML is a branch of AI, focuses on use of data and algorithms to imitate the way that humans learn gradually improving accuracy.



Task Domains of AI

- Mundane(repetitive) Tasks:
  - Perception
    - Vision
    - Speech
  - Natural Languages
    - Understanding
    - Generation
    - Translation
  - Common sense reasoning
  - Robot Control
- Formal Tasks
  - Games : chess, checkers etc
  - Mathematics: Geometry, logic,Proving properties of programs
- Expert Tasks:
  - Engineering (Design, Fault finding, Manufacturing planning)
  - Scientific Analysis
  - Medical Diagnosis
  - Financial Analysis

Branches of AI

- **Logical AI —** In general the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language
- **Search —** Artificial Intelligence programs often examine large numbers of possibilities – for example, moves in a chess game and inferences by a theorem proving program
- **Pattern Recognition —** When a program makes observations of some kind, it is often planned to compare what it sees with a pattern. For example: a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face

Branches of AI

- **Representation —** Usually languages of mathematical logic are used to represent the facts about the world.
- **Inference —** Others can be inferred from some facts. For example, when we hear of a bird, we infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin.
- **Common sense knowledge and Reasoning —** This is the area in which AI is farthest from the human level, in spite of the fact that it has been an active research area since the 1950s.

Branches of AI

- **Learning from experience** — There are some rules expressed in logic for learning. Programs can only learn what facts or behaviour their formalisms can represent.
- **Planning** — Planning starts with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal.
- **Epistemology** — This is a study of the kinds of knowledge that are required for solving problems in the world.
- **Ontology** — Ontology is the study of the kinds of things that exist. In AI, the programs and sentences deal with various kinds of objects and and what their
- basic properties are.

Branches of AI

- **Heuristics —** A heuristic is a way of trying to discover something or an idea embedded in a program. The term is used variously in AI. *Heuristic functions* are used in some approaches to search or to measure how far a node in a search tree seems to be from a goal. *Heuristic predicates* that compare two nodes in a search tree to see if one is better than the other, i.e.

  constitutes an advance toward the goal.

- **Genetic programming —** Genetic programming is an automated method for creating a workingcomputer program from a high-level problem statement of a problem. *Genetic programming* starts from a high-level statement of 'what needs to be done' and automatically creates a computer program

- ■ Machine vision
- ■ Speech understanding
- ■ Touch (*tactile* or *haptic*) sensation

❑ Robotics

❑ Natural Language Processing

- ■ Natural Language Understanding
- ■ Speech Understanding
- ■ Language Generation
- Machine TranslationPlanning
- Expert Systems Machine Learning Theorem Proving Symbolic Mathematics

Game Playing

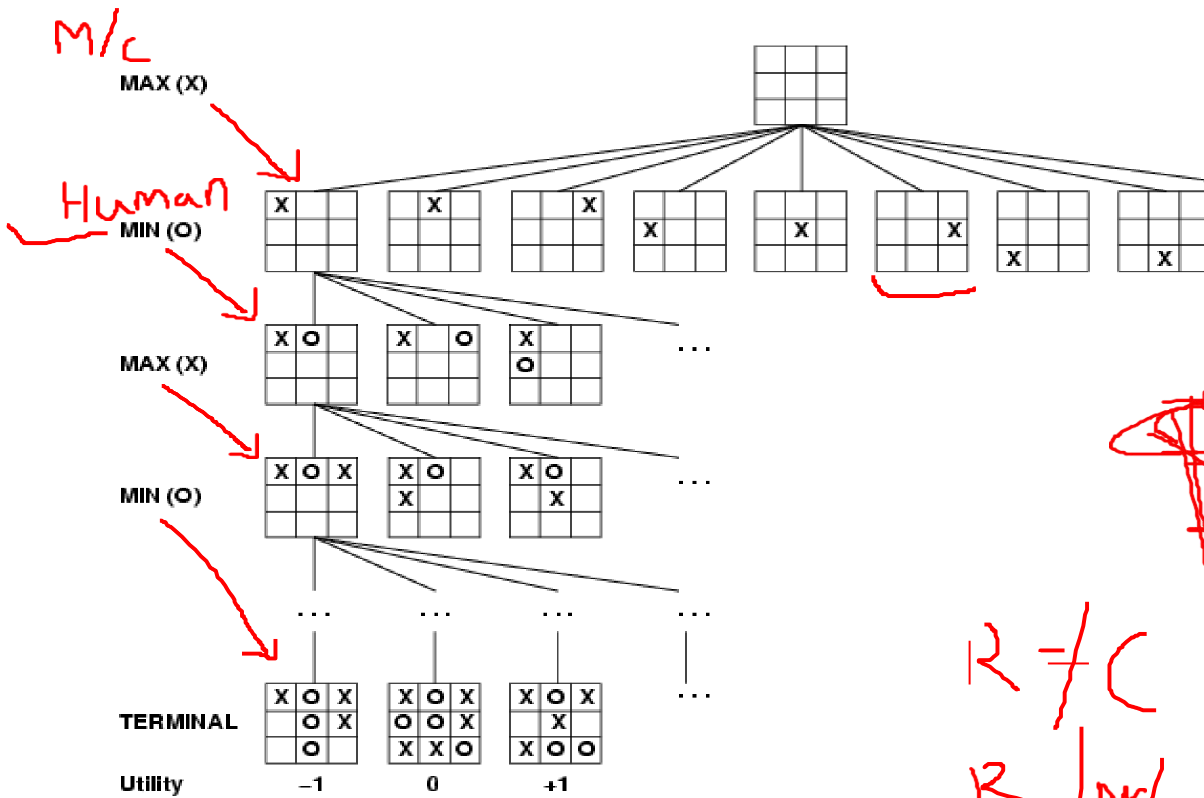
AI Technique

- Intelligence requires Knowledge
- Knowledge poseses less desirable properties such as:
  - Voluminous (very lengthy)
  - Hard to characterize accurately
  - Constantly changing
  - Differs from data that can be used
- AI technique is a method that exploits knowledge that should be represented in such a way that:
  - Knowledge captures generalization
  - It can be understood by people who must provide it
  - It can be easily modified to correct errors.
  - It can be used in variety of situations

The State of the Art

- Computer beats human in a chess game.
- Computer-human conversation using speechrecognition.
- Expert system controls a spacecraft.
- Robot can walk on stairs and hold a cup of water.
- Language translation for webpages.
- Home appliances use fuzzy logic.
- ……

Game tree (2-player, deterministic)



R ≠ C

R | M/C

H

Program-1: Tic-Tac-Toe

1.1 The first approach (simple)

The Tic-Tac-Toe game consists of a nine element vector called BOARD; it represents the numbers 1 to 9in three rows. 3^1 3^2 3^3 ...

1	2	3
4	5	6
7	8	9

An element contains the value 0 for blank, 1 for X and 2 for O. A MOVETABLE vector consists of 19,683elements (39) and is needed where each element is a nine element vector. The contents of the vector are especially chosen to help the algorithm.

Program-1: Tic-Tac-Toe

The algorithm makes moves by pursuing the following:

B X O
{ 1, 2 }
0,

1. View the vector as a ternary number.

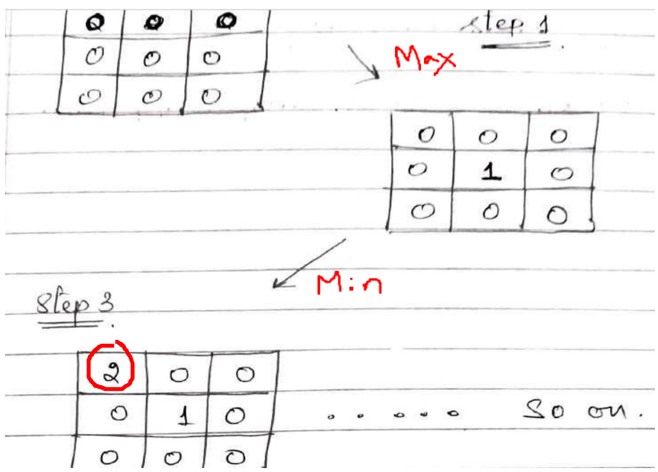
Convert it to a decimal number. 3^1 3^2 ... 3^9

2. Use the decimal number as an index in MOVETABLE

and access the vector.

3. Set BOARD to this vector indicating how the board looks after the move. This approach is capable in time but it has several disadvantages. It takes more space and requires stunning effort to calculate the decimal numbers. This method is specific to this game and cannot be completed.

Program-1: Tic-Tac-Toe



Program1:Tic-Tac-Toe

MOVETABLE

Index	Current position	Next position
0	000 000 000	000 010 000
1	.	
.	.	
.	.	
81	000 010 000	200 010 000
.		
.		
19,683		

Program1- Disadvantages

- It takes a lot of space to store the table that specifies the correct move to make from each b position.
- Someone will have to do a lot of work specifying all the entries in the movetable.
- It is very unlikely that all the required movetable entries can be determined and entered without errors.
- If we want to extend the game, say to three dimensions, we would have to start from scratch, and in this technique would no longer work at all, since 3^{27} board positions would have to be stored, overwhelming present computer memories.

Valu
0=bl
3=X
5=O

Program-2 (tic-tac-toe)

The structure of the data is as before but we use 2 for a blank, 3 for an X and 5 for an O. A variable called TURN indicates 1 for the first move and 9 for the last. The algorithm consists of three actions:

- MAKE2 which returns 5 if the centre square is blank; otherwise it
  returns any blank noncorner square, i.e. 2, 4, 6 or 8.

Win
alter

- POSSWIN (p) returns 0 if player p cannot win on the next move and otherwise returns the number of the square that gives a winning move.

R/c/D

- It checks each line using products 3*3*2 = 18 gives a win for X, 5*5*2=50 gives a win for O, and the winning move is the holder of the blank. GO (n) makes a move to square n setting BOARD[n] to 3 or 5.

R/c D

- This algorithm is more involved and takes longer but it is more efficient in storage which compensates for its longer time. It depends on the programmer's skill.

Program-2 (tic-tac-toe)

The algorithm has a built-in strategy for each move it may have to make. It make
moves if it is playing X, the even-numbered moves if it is playing O. The strategy for eac

Turn=1	Go(1) (upper left corner).
Turn=2	If Board[5] is blank, Go(5), else Go(1).
Turn=3	If Board[9] is blank, Go(9), else Go(3).
Turn=4	If Posswin(X) is not 0, then Go(Posswin(X)) [i.e., block opponent's w
Turn=5	If Posswin(X) is not 0 then Go(Posswin(X)) [i.e., win] else if Possv Go(Posswin(O)) [i.e., block win], else if Board[7] is blank, then [Here the program is trying to make a fork.]

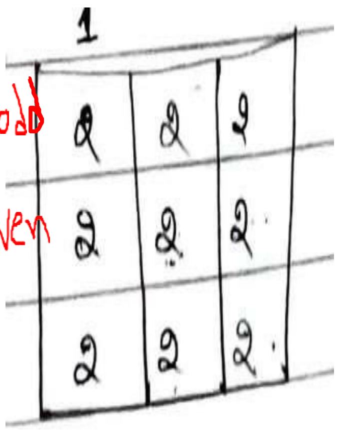
Program-2 (tic-tac-toe)

Turn=6	If Posswin(O) is not 0 then Go (Posswin(O)), else if Posswin(X) is not 0, then Go(Posswin(X)), else Go(Make2).
Turn=7	If Posswin(X) is not 0 then Go(Posswin(X)), else if Posswin(O) is not 0, then Go(Posswin(O)), else go anywhere that is blank.
Turn=8	If Posswin(O) is not 0 then Go(Posswin(O)), else if Posswin(X) is not 0, then Go(Posswin(X», else go anywhere that is blank.
Turn=9	Same as Turn=7.

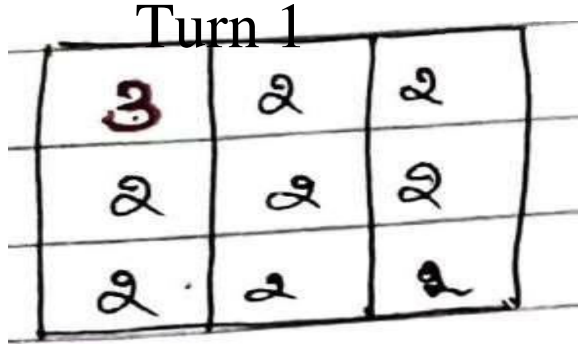
Program-2 (tic-tac-toe)

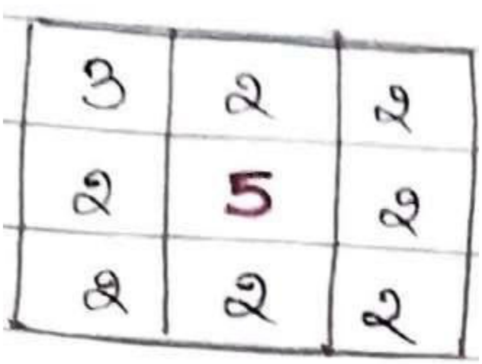
Initial

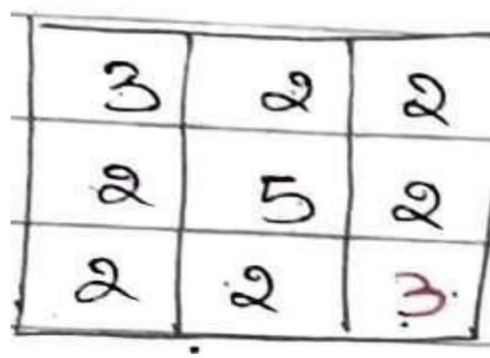
2 blank
3 - Max(M/c) odd
Min(H) Even

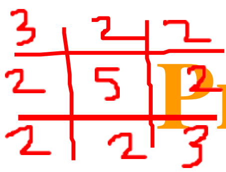


Turn 1







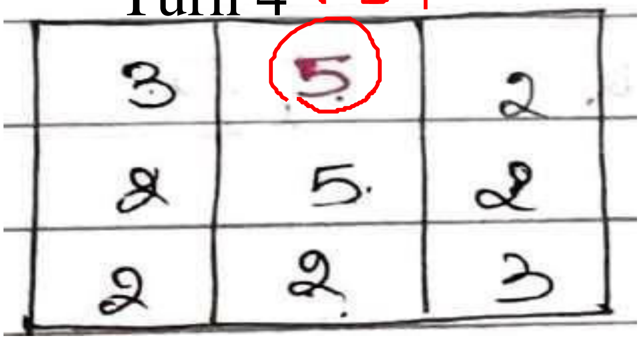


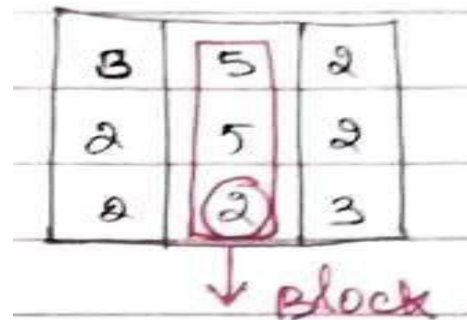
3×3× 2

Program-2 (tic-tac-toe)

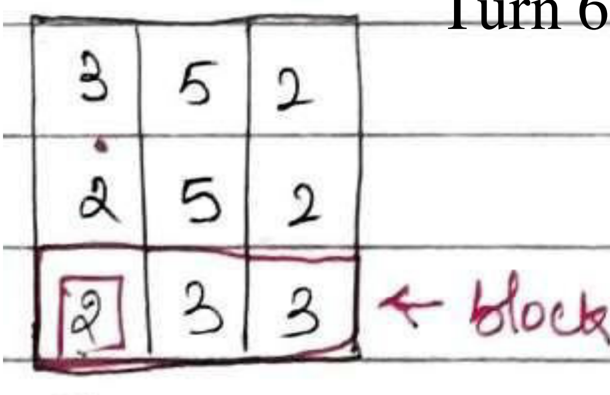
Min (5)

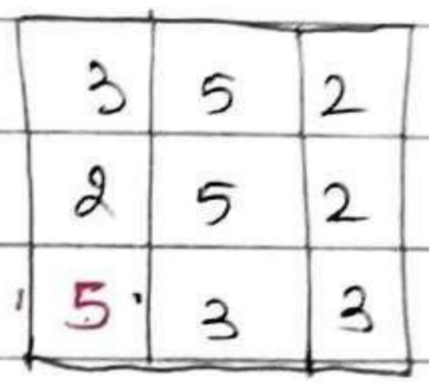
Turn 4





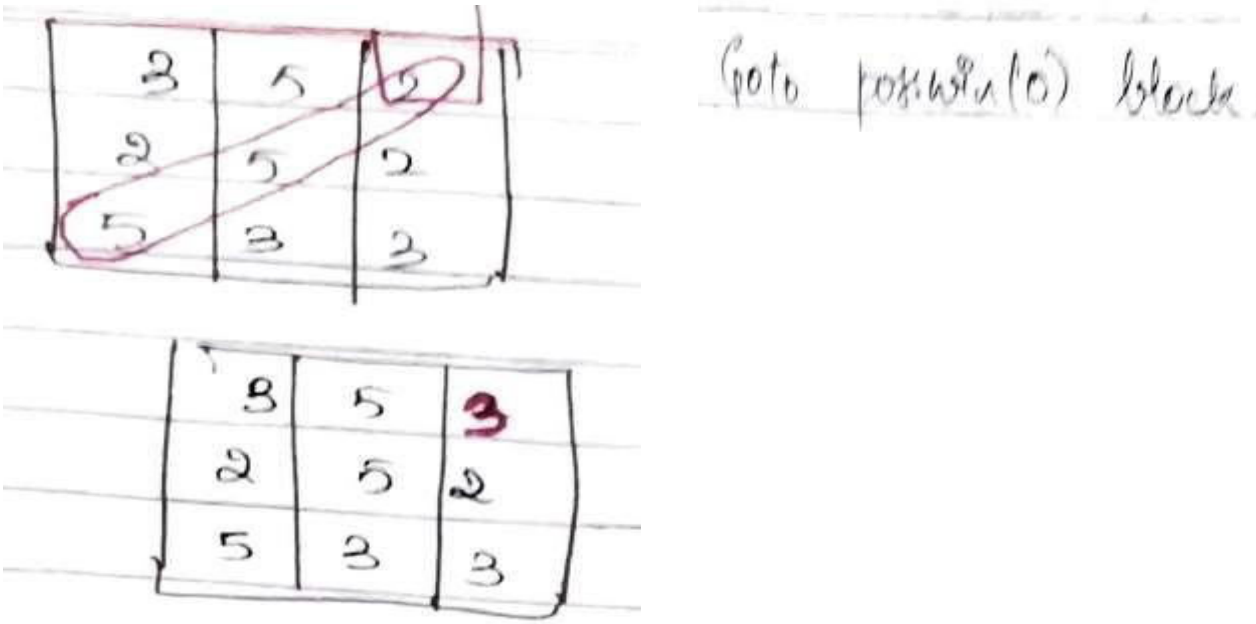
Turn 6

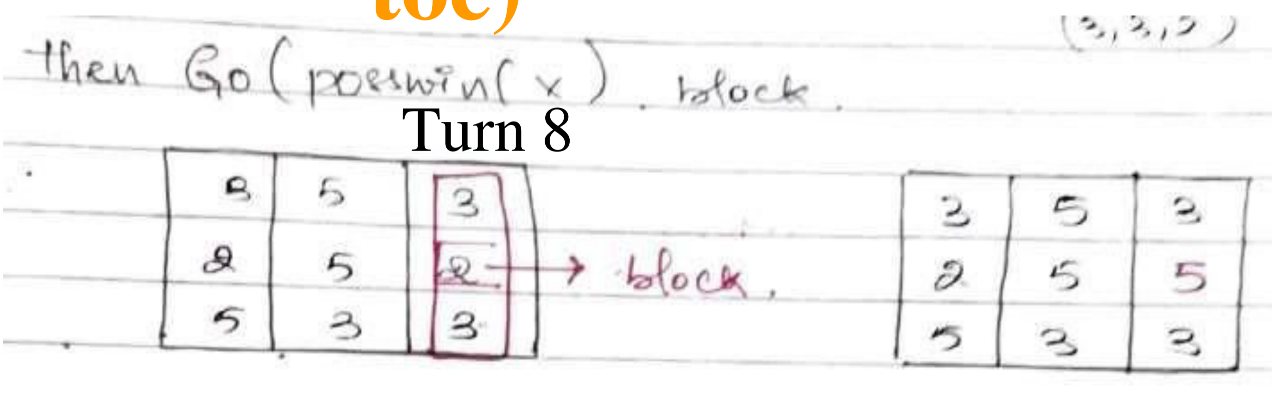




Program-2 (tic-tac-toe)

Turn 7





Turn 8

Program-2 (tic-tac-toe)

Turn 9

block it

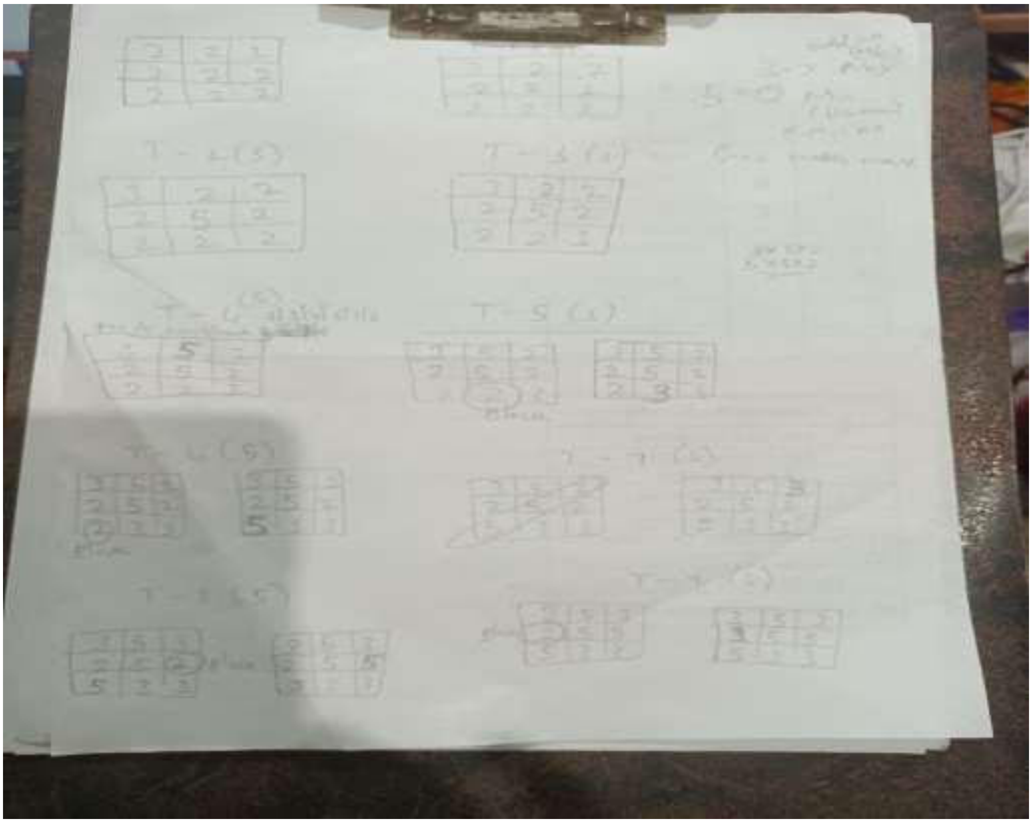
3	5	3
2	5	5
5	3	3

3	5	3
3	5	5
5	3	3

Comments

This program is not quite as efficient in terms of time as the first one since it has to check several conditions before making each move. But it is a lot more efficient in terms of space. It is also a lot easier to understand the program's strategy or to change the strategy if desired. But the total strategy has still been figured out in advance by the programmer. Any bugs in the programmer's tic-tac-toe playing skill will show up in the program's play. And we still cannot generalize any of the program's knowledge to a different domain, such as three-dimensional tic-tac-toe.

Program-2 tic tac toe



Program-3 tic tac toe(magic square)

8	3	4
1	5	9
6	7	2

The numbering of the board produces magic square: all rows, columns and diagonals sum up to 15. Here both human(uses brain) and machine(uses calculation) try to win the game by trying to make all rows or columns or diagonal elements to 15

Program-3 tic tac toe(magic square)

8
1
6

Human(mind)	Machine(brain)
	Move 1: choose 5
Move 2: choose 8	
	Move 3: choose 5,4
Move 4: choose 8,6	
	Move 5: works on calculation, first checks itself, if fails checks for opponents win by adding 2 elements 5+4=9 15-9=6 [machine cannot win] 8+6=14 15-14=1 [choose 1 by not allowing human to win] Choose 5,4,1
Move 6: choose 8,6,3	
	Move 7: 5,4,1 adds any two elements 5+1=6 15-6=9 So machine wins after choosing 9 **5**,4,**1**,**9**

Problem, problem spaces and search

Problem: problem can be caused for different reasons and can be solved in different ways. To solve a particular problem we need 4 things:

- Define problem precisely
- Analyze the problem
- Isolate and represent task knowledge
- Choose best solving technique

Define problem as state space search

Problem solving = searching for a goal state

state space is a set of legal positions, starting at initial state, using the set of rules to move from one state to another and attempting to end up in a goal state.

Methodology of state space approach

1. Represent problem in structured form using different states
2. Identify initial state
3. Identify goal state
4. Determine operator to for the changing state
5. Represent knowledge present in the problem in convenientform
6. Start from initial state and search a path to goal state

Production System

The procedure for getting a solution for AI problem canbe viewed as production system. Its components are:

- A set of rules: Left side determines applicability ofrule(pattern) and right side describes operation.
- Knowledge base: Contains information appropriate

  for a particular task.
- Control strategy: Specifies the order in which rules are implemented. First requirement is through motionand second requirement is should be systematic.
- A rule applier: production rule is shown below:

  if (conditin)

  then

  conseqence

  or

  action

Algorithm for Production System

1. Represent the initial state of the problem
2. If the present state is goal state then go to step5 else step3.
3. Choose one of the rules that satisfy the rules that satisfy the present state, apply it and change the stateto new state.
4. Go to step2
5. Print "Goal is reached" and indicate the search path from initial state to goal state.
6. stop

Classification of Production System

1. Forward Production system:

-moving from initial state to goal state

-where there are number of goal states and only one initial state, it is advantage to use forward production system.

2. Backward Production system:

-moving from goal state to initial state

-If there is only one goal state and many initial states, it is advantage to use backward productionsystem.

Categories of production systems(4)

	Monotonic	**Non-monotonic**
Partially commutative	Theorem proving	Robot navigation
Not partially commutative	Chemical synthesis	Bridge

Water Jug problem:

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one. Neit
markers on it. There is a pump that can be used to fill the jugs with water. How can you g
water into the 4-gallon jug?

The state space for this problem can be described as the set of ordered pairs of integer
0, 1, 2, 3, or 4 and y = 0, 1, 2, or 3; x represents the number of gallons of water in the
represents the quantity of water in the 3-gallon jug. The start state is (0, 0). The goal s
value of n (since the problem does not specify how many gallons need to be in the 3-ga

28

1	(x, y) if $x < 4$	$\rightarrow (4, y)$	Fill the 4-gallon ju
2	(x, y) if $y < 3$	$\rightarrow (x, 3)$	Fill the 3-gallon ju
3	(x, y) if $x > 0$	$\rightarrow (x - d, y)$	Pour some water o the 4-gallon jug
4	(x, y) if $y > 0$	$\rightarrow (x, y - d)$	Pour some water o the 3-gallon jug
5	(x, y) if $x > 0$	$\rightarrow (0, y)$	Empty the 4-gallo on the ground
6	(x, y) if $y > 0$	$\rightarrow (x, 0)$	Empty the 3-gallo on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow (4, y - (4 - x))$	Pour water from th 3-gallon jug int 4-gallon jug un 4-gallon jug is
8	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow (x - (3 - y), 3)$	Pour water from th 4-gallon jug int 3-gallon jug un 3-gallon jug is
9	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow (x + y, 0)$	Pour all the water from the 3-gall into the 4-gallo
10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow (0, x + y)$	Pour all the water from the 4-gall into the 3-gall
11	$(0, 2)$	$\rightarrow (2, 0)$	Pour the 2 gallons from the 3-gall into the 4-gall
12	$(2, y)$	$\rightarrow (0, y)$	Empty the 2 gallo the 4-gallon ju the ground

Fig. 2.3 *Production Rules for the Water Jug Problem*

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

Types of search algorithm

Uniformed search: will not have domain knowledge, operates in brute force way and no information about search space

Informed search: knows domain knowledge, find solution efficiently, operates heuristic way(guarantees good solutionnot best), can solve complex problems.

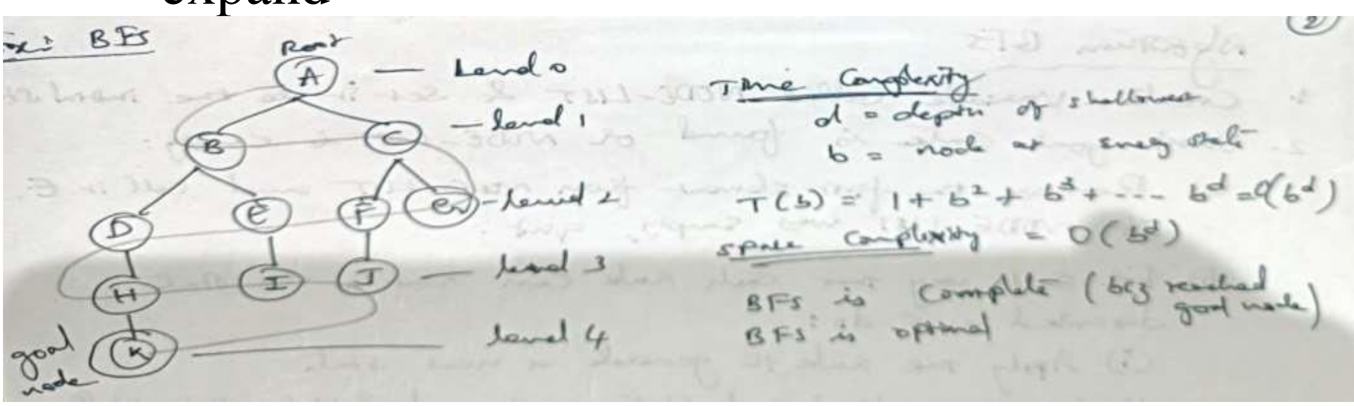
Uninformed(Blind search)	Informed(Heuristic)
BFS	Best fit
Uniform cost	A*
DFS	AO*
Depth limit	Problem reduction
Iterative deeping DFS	Hill climbing
Bidirectional search	

Breadth First Search

- Most common search strategy
- Searches breadth wise
- Searches from root and expands to all successors
- Implemented using FIFO(queue) data structure

Advantage: will provide solution

Disadvantage: requires lot of memory to expand



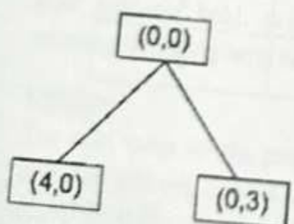
BFS Algorithm

Breadth First Search

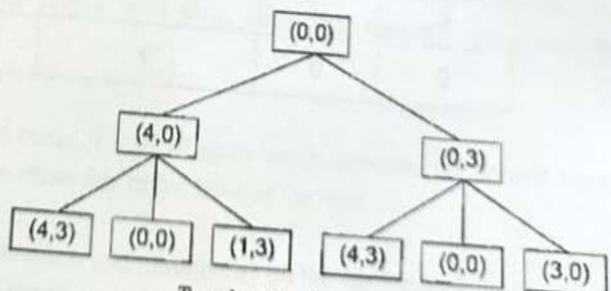
To solve the water jug problem systemically construct a tree with limited states as its root. Generate all the offspring and their successors from the root according to the rules until some rule produces a goal state. This process is called **Breadth-First Search.**

Algorithm:

1) Create a variable called NODE_LIST and set it to the initial state.
2) Until a goal state is found or NODE_LIST is empty do:
   a. Remove the first element from NODE_LIST and call it E. If NODE_LIST was empty quit.
   b. For each way that each rule can match the state described in E do:
      i. Apply the rule to generate a new state
      ii. If the new state is goal state, quit and return this state
      iii. Otherwise add the new state to the end of NODE_LIST



One Level of a Breadth-First Search Tree



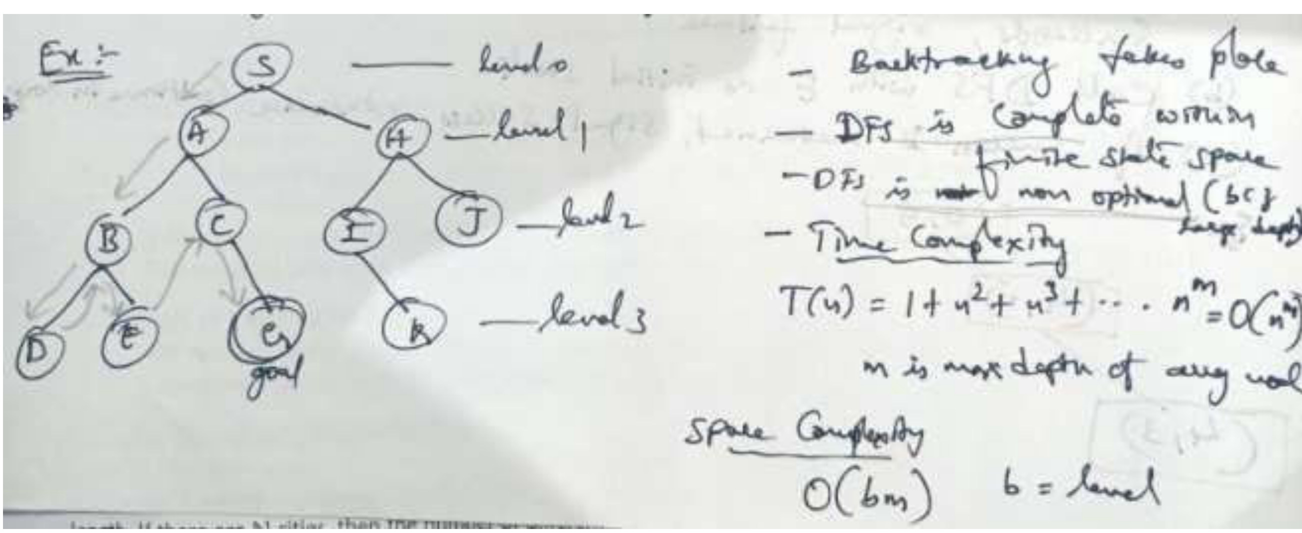
Two Levels of a Breadth-First Search Tree

The data stru

Depth First Search

- Recursive algorithm
- Starts with root and follows to its greatest depth
- Uses

LIFO
(stack) data
structure
Advantage:
requires less
memory
Disadvantage: no guarantee of finding solution and can goto infinite depth



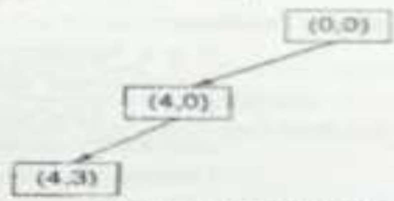
DFS Algorithm

Depth First Search

There is another way of dealing the Water Jug Problem. One should construct a single branched tree utility yields a solution or until a decision terminate when the path is reaching a dead end to the previous state. If the branch is larger than the pre-specified unit then backtracking occurs to the previous state so as to create another path. This is called **Chronological Backtracking** because the order in which steps are undone depends only on the temporal sequence in which the steps were originally made. This procedure is called **Depth-First Search.**

Algorithm:

1) If the initial state is the goal state, quit return success.
2) Otherwise, do the following until success or failure is signaled
   a. Generate a successor E of the initial state. If there are no more successors, signal failure
   b. Call Depth-First Search with E as the initial state
   c. If success is returned, signal success. Otherwise continue in this loop.



A Depth-First Search Tree

The data structure used in this algorithm is STACK.
Explanation of Algorithm:

- Initially put the **(0,0)** state in the stack.
- Apply production rules and generate the new state.
- If the new states are not a goal state, (not generated before and no expanded) then only add the state to top of the Stack.
- If already generated state is encountered then POP the top of stack elements and search in another direction.

Heuristic Search: uses various shortcuts in order to produce solutions that may not be optimal

- Heuristic search methods often known as weak methodsbecause they do not apply great deal of knowledge.

WEAK METHODS:

a) Generate and Test

b) Hill Climbing(simple, steepest and simulated Annealing)

c) Best First search

d) Problem reduction

e) Constraint satisfaction

f) Means-ends analysis

Generate and Test

Generate and Test

The generate-and-test strategy is the simplest of all the approaches. It consists of the fol
steps:

Algorithm:

1. Generate a possible solution. For some problems, this means generating a particula
in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endp
the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise return to step 1.

Example: searching a ball in a bowl

Hill Climbing(simple,steepest)

Algorithm: Simple Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   (a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   (b) Evaluate the new state.
      (i) If it is a goal state, then return it and quit.
      (ii) If it is not a goal state but it is better than the current state, then make it the current state.
      (iii) If it is not better than the current state, then continue in the loop.

Steepest Hill Climbing

Algorithm: Steepest-Ascent Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   (a) Let *SUCC* be a state such that any possible successor of the current state will be better than *SUCC*.
   (b) For each operator that applies to the current state do:
      (i) Apply the operator and generate a new state.
      (ii) Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to *SUCC*. If it is better, then set *SUCC* to this state. If it is not better, leave *SUCC* alone.
   (c) If the *SUCC* is better than current state, then set current state to *SUCC*.

Both basic and steepest-ascent hill climbing may fail to find a solution. Either algorithm may terminate not by finding a goal state but by getting to a state from which no better states can be generated. This will happen if the program has reached either a local maximum, a plateau, or a ridge.

Local maximum: a state better than all its neighbours

Plateau: a flat area where neighbouring states has the same value

Ridge: a area higher than surrounding areas.

Simulated Annealing

In simulated Annealing some hill down movements can bemade. In physical annealing:

-Physical substances are melted and gradually cooled untilsome solid state is reached.

-The goal is to produce a minimal energy state

-Annealing schedule: if temperature is lowered sufficientlyslowly, then goal will be attained

-The probability for a transition $P=e^{-\Delta E/KT}$

_ Δ E is positive energy level

_T is temperature

_K is boltzman constant

Simulated Annealing

Algorithm: Simulated Annealing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Initialize *BEST-SO-FAR* to the current state.
3. Initialize T according to the annealing schedule.
4. Loop until a solution is found or until there are no new operators left to be applied in the current state.
   (a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   (b) Evaluate the new state. Compute
   $$\Delta E = (\text{value of current}) - (\text{value of new state})$$
   - If the new state is a goal state, then return it and quit.
   - If it is not a goal state but is better than the current state, then make it the current state. Also set *BEST-SO-FAR* to this new state.
   - If it is not better than the current state, then make it the current state with probability p' as defined above. This step is usually implemented by invoking a random number generator to produce a number in the range [0,1]. If that number is less than p', then the move is accepted. Otherwise, do nothing.

   (c) Revise T as necessary according to the annealing schedule.
5. Return *BEST-SO-FAR*, as the answer.

Ex: current state p=0.45 and new state p'=0.36 if (p>p') move is rejected

Best First Search(Greedy search)

- It always selects the path that appears best at that moment
- It's a combination of DFS and BFS
- It uses heuristic function: $h(n)<h^*(n)$ and searches
- H(n)=heuristic cost
- H*(n)=estimated cost
- It is implemented by priority queue

Best First Search Algorithm

1. Place the starting node into the OPEN list
2. If the OPEN list is empty, stop and return failure
3. Remove node n from OPEN list that has lowest value of h(n) and place into the CLOSE list.
4. Expand node n and generate succors of node n
5. Check each successor of node n and find whether node is a goal node or not. If any successor node is a goal node, then return success and terminate else proceed to step 6.
6. For each successor node check if node has been in OPEN or CLOSE list. If it is not in both, then add to OPEN list.
7. Return to step2.

Advantage: more efficient than BFS and DFS
Disadvantage: can stuck in loop as dfs

A* search algorithm

- A* search algorithm finds shortest path through the search space using heuristic function h(n)
- It uses h(n) and cost to reach the node n from start state g(n)
- Provides optimal results faster

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

F(n): estimated cost

g(n): cost to reach node n from start state

h(n): cost to reach node n to goal state

A* algorithm steps

1. Place the starting node in the OPEN list
2. Check if open list is empty or not, if it is empty return failure and stop
3. Select node from open list, which has smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
4. Expand node n and generate all its successors and put n in CLOSE list

– For each successor n, check n in already in OPEN or CLOSED list

– If not compute evaluation function for

n' and place into
OPEN list.

A* algorithm contd.

5. Else if n' is already in OPEN and CLOSED then it shouldbe attached to the back pointer which reflects the lowest g(n') value

6. Return to step2

Advantage: best algorithm, optimal and complete, solvesvery complex problems

Disadvantage: not practical for very large scale problems.

AO* search algorithm (AND-OR)

Means Ends Analysis

Algorithm: Means-Ends Analysis (CURRENT, GOAL)

1. Compare *CURRENT* to *GOAL*. If there are no differences between them then return.
2. Otherwise, select the most important difference and reduce it by doing the following until success or failure is signaled:
   (a) Select an as yet untried operator *O* that is applicable to the current difference. If there are no such operators, then signal failure.
   (b) Attempt to apply *O* to *CURRENT*. Generate descriptions of two states: *O-START*, a state in which *O*'s preconditions are satisfied and *O-RESULT*, the state that would result if *O* were applied in *O-START*.
   (c) If
   (FIRST-PART ← MEA(CURRENT, O-START))
   and
   (LAST-PART ← MEMO-RESULT, GOAL))
   are successful, *then* signal success and return the result of concatenating
   FIRST-PART, O, and *LAST-PART.*

Constraint satisfaction problem (CSP)

Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this, first set *OPEN* to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until *OPEN* is empty:
   (a) Select an object *OB* from *OPEN*. Strengthen as much as possible the set of constraints that apply to *OB*.
   (b) If this set is different from the set that was assigned the last time *OB* was examined or if this is the first time *OB* has been examined, then add to *OPEN* all objects that share any constraints with *OB*.
   (c) Remove *OB* from *OPEN*.
2. If the union of the constraints discovered above defines a solution, then quit and report the solution.
3. If the union of the constraints discovered above defines a contradiction, then return failure.
4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:
   (a) Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.
   (b) Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

Module-2

AI part and ML part

Knowledge Representation Issues

Representation and Mappings:

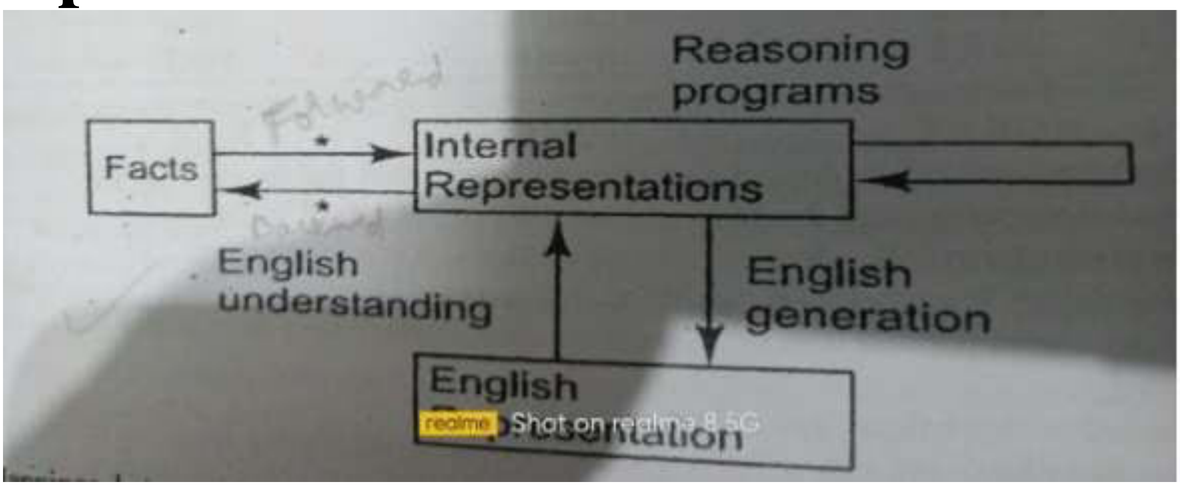
To solve complex problems in AI we need large amounts of knowledge and mechanisms for manipulating that knowledge. Different ways of representing the knowledge:

-Facts(truths)

-Representation of facts

-Structuring both(knowledge level(facts), symbol level(representation of facts)

Mappings between facts and Representation



Forward representation: mapping from facts torepresentation

Backward representation: mapping from representationto facts.

Mapping functions from English sentences torepresentation and back to sentences.

Knowledge representation schemes

There are 4 types of knowledge representationschemes:

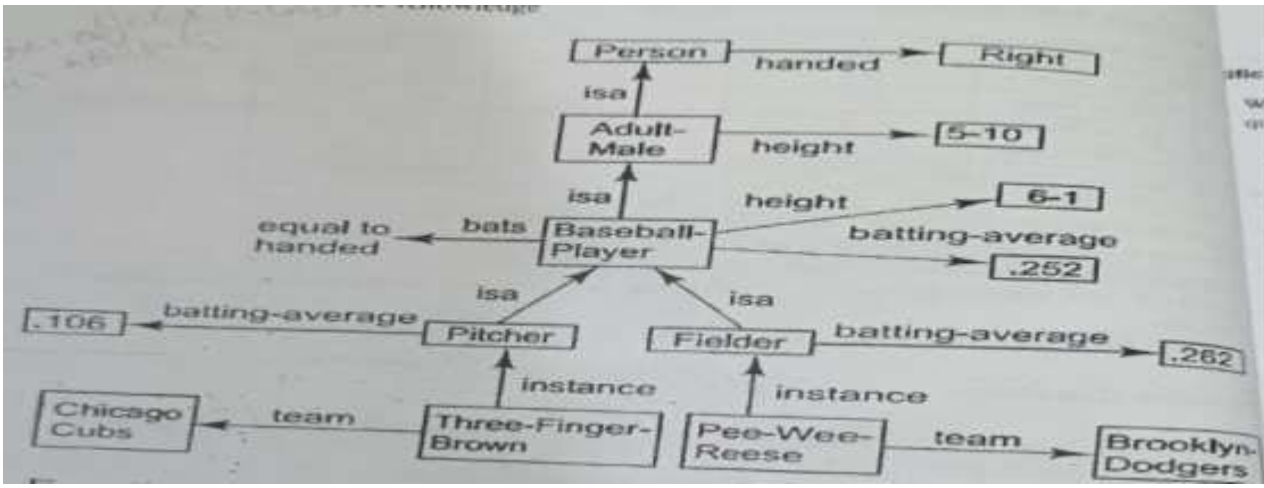
- Relational
- Inheritable
- Inferential
- Declarative

Relational knowledge

Player	Height	Weight	Bats-Throws
Hank Aaron	6-0	180	Right-Right
Willie Mays	5-10	170	Right-Right
Babe Ruth	6-2	215	Left-Left
Ted Williams	6-3	205	Left-Right
player_info('hank aaron', '6-0', 180,right-right).			

- Made up of objects with attributes and values
- Associates elements from one domain toanother
- Mapping of elements among differentdomains is possible

Inheritable Knowledge



Objects must be organized into classes andclasses must be arranged in generalizationhierarchy.

- Elements inherit attributes from their parents

Inferential knowledge

Example: 1. Tommy is a dog

dog (Tommy)

2. All dogs are animals

$\forall x$ dog $(x) \longrightarrow$ animal (x)

3. All animals either live on land or in water

$\forall x$ animal $(x) \longrightarrow$ live $(x$, land$)$ v live $(x$, water$)$

- It is a powerful form of inference.
- Sometimes traditional logic is necessary todescribe inferences
- It is used to generate new knowledge fromgiven knowledge

Declarative/procedural knowledge

✓ **Example:** Procedural Knowledge as Rules

If: Internal marks is minimum of **12 out of 20** and external marks is **35% of 80**, i.e., **28**, leads to **40% of 100** marks

Then: Result of the subject is **pass – E grade**

- These are represented as small programs thatknow how to do specific programs
- Commonly used technique in this is production rules.

*Issues in knowledge representation

a) Important attributes
b) Relationships among attributes
c) Choosing the granularity of representation
d) Representing sets of objects
e) Finding the right structures as needed

a) Important attributes

There are 2 attributes that are basic and common and occur in almost every problemdomain. They are:

- Is-A
- Instance

b) Relationships among attributes

There are 4 important relationships that existamong attributes. They are:

- Inverses
- Existence in an IS-A hierarchy
- Techniques for reasoning about values
- Single valued attributes

Inverses

For **example,** the assertion:

team (Pee-Wee-Reese, Brooklyn-Dodgers)

The second approach is to use attributes that focus on a single entity but to use the
one the inverse of the other. In this approach, we would represent the team inform
two attributes:

One associated with **Pee Wee Reese:**

team=Brooklyn-Dodgers

One associated with **Brooklyn Dodgers:**

team-members=Pee-Wee-Reese,...

This is the approach that is taken in **semantic net** and **frame-based** systems.

IS-A hierarchy of attributes

For example: the attribute height is actually a specialization of more general attribute called physical-size which is in turn a specialization of physical-attribute.

Techniques for reasoning about values

- Reasoning system must reason about values ithas not been given explicitly.
- Example1: the age of a person cannot begreater than age of their parents
- Example2: height must be measured in a unitof length

Single valued attributes

Example: a baseball player can, at any one time,have only a single height and be a member of only one team.

c) Choosing the granularity of representation

- It is necessary to answer the question – **At what level of detail should the Knowle represented?** Should there be a **small number of low-level ones** or should there be a **number** covering a range of granularities?

✓ **Example 1:** Suppose we are interested in the following **fact:**

John spotted Sue.
We could represent this as
spotted(agent(John).
object(Sue))

Such a representation would make it easy to answer questions such as: **Who spotted Sue**
But now suppose we want to know:

Did John see Sue?

The obvious answer is **yes,** but given only the one fact we have, we cannot discove answer, we could, add other facts, such as

$$\textbf{spotted}(x, y) \rightarrow \textbf{saw}(x, y)$$

We could then **infer the answer** to the question.

An **alternative solution** to this problem is to represent the fact **spotting** is really a s type of **seeing** explicitly in the representation of the fact. we might write as:

saw(agent(John),
object(Sue),
timespan(briefly))

d) Representing set of objects

There are 2 ways to represent a set and its elements.

- Extensional definition: list the members

Ex: set of sun's planets on which people live is {earth}

- Intensional definition: true or false

Ex: {x:sun_planet(x) ∧ human_inhabited(x)}

e) Finding right structures as needed

Finding the Right Structures as Needed

- ✓ Here, is to find right structure for accessing relevant parts of knowledge.
- ✓ **For example,** suppose we have a **script (a description of a class of events in contexts, participants, and subevents)** that describes the typical sequence of ev restaurant. This script would enable us to take a text such as

  John went to Steak and Ale last night. He ordered a large rare steak, paid his left.

  And answer **"yes"** to the question, **Did John eat dinner last night?**

- ✓ In order to have access to the right structure for describing a particular situati necessary to solve all of the following problems.
  - How to perform an **initial selection** of the most appropriate structure.
  - How to fill in appropriate **details** from the current situation.
  - How to find a **better structure** if the one chosen initially turns out not to be app
  - What to do if **none** of the available structure is **appropriate.**
  - When to **create** and **remember** a new structure.

Predicate Logic

Predicate logic is used to represent knowledge.

- Logic is a language for reasoning, a collectionof of rules.
- Predicate is a truth assignment given for a particular statement which is either true orfalse. Logic symbols used in predicate logicare:

$\forall$ — *For all*
$\exists$ — *There exists*
$\rightarrow$ — *Implies*
$\neg$ — *Not*
$\vee$ — *OR*
$\wedge$ — *AND*

Demorgan’s laws in Predicate Logic

DE Morgan’s Laws in Predicate logic

$\neg(\neg a) = a$

$\neg(a \vee b) = \neg a \wedge \neg b$

$\neg(a \wedge b) = \neg a \vee \neg b$

Predicate logic contd.

a) The balls color is red: color(ball,red)
b) Rohan likes bananas: likes(rohan, bananas)
c) Raju likes rani: likes(raju,rani)
d) Raju likes everyone
e) Someone likes someone
f) Someone likes everyone
g) Everyone likes someone
h) Everyone is liked by someone
i) Someone is liked by everyone
j) Nobody likes everyone
k) Every gardener likes sun:
l) All purple mushrooms are poisonous
m) Everyone loves everyone

Representing facts with predicate logic

1) Marcus is a noun and man is predicate or marcus is an instance of class

Representing facts with Predicate Logic

1) Marcus was a man $man(Marcus)$
2) Marcus was a Pompeian $pompeian(Marcus)$
3) All Pompeians were Romans $\forall x : pompeian(x) \rightarrow roman(x)$
4) Caeser was a ruler. $ruler(Ceaser)$
5) All romans were either loyal to caeser or hated him.

$$\forall x : roman(x) \rightarrow loyalto(x, caeser) \vee hate(x, caeser)$$

6) Everyone loyal to someone. $\forall x, \exists y : loyalto(x, y)$
7) People only try to assassinate rulers they are not loyal to.

$$\forall x, \forall y: Person(x) \wedge Ruler(y) \wedge try\_assassinate(x, y) \rightarrow \neg Loyal\_to(x, y)$$

8) Marcus try to assassinate Ceaser $try\_assacinate(Marcus, Ceaser)$

Q. Prove that Marcus is not loyal to Ceaser by backward substitution

4. $\neg Loyal\_to(Marcus, Ceaser)$

$\uparrow$

5. $Person(Marcus) \wedge Ruler(Ceaser) \wedge Try\_assacinate(Marcus, Ceaser)$

6. $\uparrow$

7. $Person(Marcus) \wedge Ruler(Ceaser)$

8. $\uparrow$

9. $Person(Marcus)$

Q. Prove that Marcus is not loyal to Ceaser

hat Marcus is not loyal to Ceaser by backward substitution

4. $\neg Loyal\_to(Marcus, Ceaser)$

$\uparrow$

5. $Person(Marcus) \wedge Ruler(Ceaser) \wedge Try\_assacinate(Marcus, Ceaser)$

6. $\uparrow$

7. $Person(Marcus) \wedge Ruler(Ceaser)$

8. $\uparrow$

9. $Person(Marcus)$

Computable Functions and Predicates

It would be extremely inefficient to store explicitly a large number of statements, so to

1. Marcus was a Man => $Man(Marcus)$
2. Marcus was a Pompeian => $Pompeian(Marcus)$
3. Marcus born in 40 AD => $Born(Marcus, 40)$
4. All men are mortal => $\forall x: Men(x) \rightarrow Mortal(x)$
5. All Pompeians died when the volcano was erupted in 79 AD.
   $Erupted(volcano, 79) \wedge (\forall x: pompeian(x) \rightarrow died(x, 79))$
6. No mortal lives longer than 150 years
   $\forall x, \forall t1, \forall t2: Mortal(x) \wedge Born(x, t1) \wedge Greater\_then(t2 - t1, 150) \rightarrow died(x, t2)$
7. It is now 1991 => $Now = 1991$
8. Alive means not deal
   $\forall x, \forall t: (alive(x,t) \rightarrow \neg Dead(x,t)) \wedge \neg Dead(x,t) \rightarrow alive(x,t))$
9. If someone dies then he is dead at all later times
   $\forall x, \forall t1, \forall t2: Died(x, t1)\ Greater\_then(t2, t1) \rightarrow Dead\ (x, t2)$

compute easily we need computable predicates.

Q. Prove that Marcus is dead

$\neg$ alive (Marcus, now)
↑ (9, substitution)
Dead(Marcus, now)
↑ (10, substitution)
Died(Marcus, t1) $\wedge$ gt(now,t1)
↑ (5, substitution)
Pompeian (Marcus) $\wedge$ gt(now,79)
↑ (2)
gt(now,79)
↑ (8, substitute equals)
gt(1991,79)
↑ (Compute gt)
Nil

Unification example

Unification

making expressions look identical
we need to do substitutions

Ex: $P[x, f(y)]$ $P[a, f(g(z))]$ first arg =
sec arg = fu

if x is replaced with a
and " y " " " $g(z)$

Unification $[a/x, g(z)/y]$

Conditions

(1) Predicate symbols should be same P
(2) No. of args should be identical
(3) Unification fails, when there are 2 similar
in same Expr.

The Unification Algorithm

1. Initial predicate symbols must match.
2. For each pair of predicate arguments:

- Different constants cannot match
- A variable may be replaced by a constant
- A variable may be replaced by another variable
- A variable may be replaced by a function as longas it does not contain an instance of the variable
- When attempting to match 2 literals, all substitutions must be made to the entire literal

Unification Algorithm

Algorithm: Unify(L1, L2)

1. If *L1* or *L2* are both variables or constants, then:
   (a) If *L1* and *L2* are identical, then return NIL.
   (b) Else if *L1* is a variable, then if *L1* occurs in *L2* then return {FAIL}, else return (*L2/L1*).
   (c) Else if *L2* is a variable then if *L2* occurs in *L1* then return {FAIL}, else return (*L1/L2*).
   (d) Else return {FAIL}.
2. If the initial predicate symbols in *L1* and *L2* are not identical, then return {FAIL}.
3. If *L1* and *L2* have a different number of arguments, then return {FAIL}.
4. Set *SUBST* to NIL. (At the end of this procedure, *SUBST* will contain all the substitutions used to u *L1* and *L2*.)
5. For $i \leftarrow 1$ to number of arguments in *L1*:
   (a) Call Unify with the *i*th argument of *L1* and the *i*th argument of *L2*, putting result in *S*.
   (b) If *S* contains FAIL then return {FAIL}.
   (c) If *S* is not equal to NIL then:
      (i) Apply *S* to the remainder of both *L1* and *L2*.
      (ii) *SUBST* : = APPEND(S, *SUBST*).
6. Return *SUBST*.

Unification Resolution

Consider $P(x, g(x)) = P(z, y)$

Soln (i) $P(z, y)$: Unifies with $[x/z, g(x)/y]$

(ii) $P(z, g(z))$: Unifies with $[x/z$ or $z/x]$

(iii) $P($Prime, $f($Prime$)$: does not Unifies

[f and g does not match

Representing knowledge using Rules

Procedural V/s Declarative

Procedural knowledge

Knowledge is embedded in knowledge itself

- Answers "what can you do"?
- Demonstrated using nouns
- Relies on action words or verbs
- Ability to carry out actionsto complete a task

Procedural V/s Declarative contd.

Example

1. Man(marcus)
2. Man(ceaser)
3. $\forall x$: man(x)$\rightarrow$person(x)
4. Person(cleopatra)

Statements 1,2,3 are procedural and 4 isdeclarative

Forward & Backward Reasoning

Forward Reasoning

Reasoning forward from initialstate

- Build a tree of move sequences with initialconfiguration
- Generate next level of tree whose left side rules match the root node
- Generate next level considering previous level whose left sides match
- Continue until the goal state is generated

Forward and Backward Chaining

Forward chaining Rule Systems

- Want to be directed by

  incoming data
- Rules of RHS assertions aredumped into the state andthe process repeats
- Matching is more complex

  than backward chaining

- Example: sense heat near your hand and take away

Logic Programming

Logic Programming

- Logic Programming is a programming language paradigm in which logical a are viewed as programs.
- There are several logic programming systems in use today, the most popular is PROLOG.
- A PROLOG program is described as a series of logical assertions, each of w Horn clause.
- A Horn clause is a clause that has at most one positive literal. Thus p, ¬p ∨ are all Horn clauses.

Programs written in pure PROLOG are composed only of Horn Clauses.

Difference between logic and PROLOG representation

LOGIC

- Variables are explicitly quantified
- Explicit symbols for AND($\wedge$) and OR($\vee$) are used
- P implies q is written as $p \rightarrow q$

Example of logic and PROLOG representation

Example:

$\forall x : pet(x) \wedge small(x) \rightarrow apartmentpet(x)$
$\forall x : cat(x) \vee dog(x) \rightarrow pet(x)$
$\forall x : poodle(x) \rightarrow dog(x) \wedge small(x)$
$poodle(fluffy)$

A Representation in Logic

```
apartmentpet(X) :- pet(X), small(X).
pet(X) :- cat(X).
pet(X) :- dog(X).
dog(X) :- poodle (X) .
small(X) :- poodle(X).
poodle(fluffy).
```

A Representation in PROLOG

Example of Horn and PROLOG

$\forall x : \forall y : cat(x) \wedge fish(y) \rightarrow likes-to-eat(x,y)$
$\forall x : calico(x) \rightarrow cat(x)$
$\forall x : tuna(x) \rightarrow fish(x)$
$tuna(Charlie)$
$tuna(Herb)$
$calico(Puss)$

(a) Convert these wff's into Horn clauses.
(b) Convert the Horn clauses into a PROLOG program.
(c) Write a PROLOG query corresponding to the question, "What does Puss like to eat?" and show how it will be answered by your program.

(a) Horn clauses:

1. $\neg cat(x) \vee \neg fish(y) \vee likes\text{-}to\text{-}eat(x, y)$
2. $\neg calico(x) \vee cat(x)$
3. $\neg tuna(x) \vee fish(x)$
4. $tuna(Charlie)$
5. $tuna(Herb)$
6. $calico(Puss)$

(b) PROLOG program:

```
likestoeat(X,Y) :- cat(X), fish(Y).
cat(X) :- calico(X).
fish(X) :- tuna(X).
tuna(charlie).
tuna(herb).
calico(puss).
```

$P \rightarrow q$ AND
$q :- P$

Matching

- The process of search to solve problem beginswith appropriate rules to generate new states
- Hence there should be a matching betweencurrent state and preconditioned rules. They are

-Indexing

-Matching with variables

-complex and appropriate matching

-conflict resolution

Indexing

One way to select applicable rules is to do **simple search** through all the rules. But there are 2 problems with this:

-it will be necessary to use large no of ruleswould be **inefficient**

-it is **not always** immediately obvious whether arule is satisfied by particular state

Approximate matching

Approximate Matching:

Rules should be applied if their preconditions approximately match to the current situation

Eg: Speech understanding program

Rules: A description of a physical waveform to phones

Physical Signal: difference in the way individuals speak, result of background noise.

Matching with variables

One efficient many-many match algorithm RETE (many rules are matched against many elements). This gains efficiency from 3 major sources:

- ✓ Temporal nature of data: rules do not alter the state description completely
- ✓ Structural similarity in rules:

$$
\left.
\begin{array}{l}
mammal(x) \wedge \\
feline(x) \wedge \\
carnivorous(x) \wedge \\
has-spots(x)
\end{array}
\right. \rightarrow jaguar(x)
$$

$$
\left.
\begin{array}{l}
mammal(x) \wedge \\
feline(x) \wedge \\
carnivorous(x) \wedge \\
has-stripes(x)
\end{array}
\right. \rightarrow tiger(x)
$$

- ✓ Persistence of variable binding consistency:

$$Son(x, y) \wedge son(y, z) \rightarrow grandparent\ (x, z)$$

$$\begin{array}{l} son(x, y) \wedge \\ son(y, z) \end{array} \quad \rightarrow grandparent(x, z)$$

Conflict resolution

... result of background noise.

Conflict Resolution:

When several rules matched at once such a situation is called conflict resolution. There approaches to the problem of conflict resolution in production system.

1. Preference based on rule match:
    a. Physical order of rules in which they are presented to the system
    b. Priority is given to rules in the order in which they appear
2. Preference based on the objects match:
    a. Considers importance of objects that are matched
    b. Considers the position of the match able objects in terms of Long Term Mer (LTM) & Short Term Memory(STM)
    LTM: Stores a set of rules
    STM (Working Memory): Serves as storage area for the facts deduced by rule long term memory
3. Preference based on the Action:
    a. One way to do is find all the rules temporarily and examine the results of ea Using a Heuristic function that can evaluate each of the resulting states compa the merits of the result and then select the preferred one.

Concept Learning(Machine learning)

Machine learning is a type of AI allows software applications to become more accurate at predicting outcomes without being explicitly programmed.

There are 3 types of machine learning:

-Supervised: Task driven(predict next value)

-Unsupervised: Data driven(identify clusters)

-Reinforcement: Learn from mistakes

Concept Learning

Concept learning can be formulated as a problem of searching through a predefinedspace of potential hypothesis (statement of prediction)for the hypothesis that best fits the training examples.

A concept learning task

- Consider the example task of learning the target concept "
days on which my friend Aldo enjoys his favorite water sport"

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- Table describes a set of examples , each represented by a set of
attributes.
- The Enjoy sport indicates whether or not Aldo enjoys his favorite
water sport on this day.
- The task is to learn to predict the value of Enjoy sport for an
arbitrary day
- '?' indicate any value is acceptable

and ‘Φ’ indicate no value is acceptable.

Concept Learning as Search

The goal of this search is to find the hypothesis that best fits the training examples. The designer of the learning algorithm implicitly defines the space of all hypothesis.

-Find S: Finding a maximally specific hypothesis algorithm

-Version space and the candidate elimination algorithm

Find – S Algorithm

Objective	To find most specific hypothesis in set of hypotheses, which is consistent with positive training example.
Dataset	Tennis data set: This data set contains the set of examples days on which playing of tennis is possible or not, based on attributes *Sky, AirTemp, Humidity, Wind, Water and Forecast*.
ML Algorithm	Supervised Learning-FIND-S Algorithm
Description	The FIND-S Algorithm is probably one of the simplest machine learning algorithms.

Algorithm:

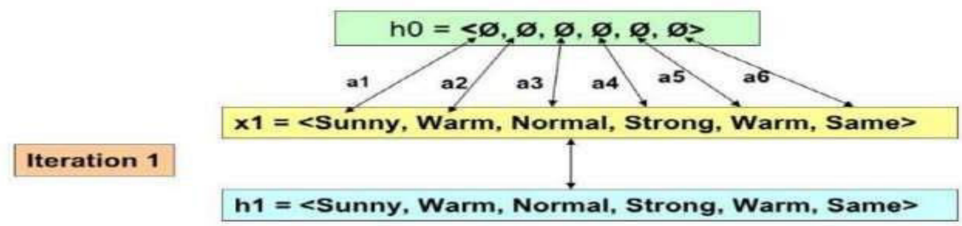
1. Initialize **h** to the most specific hypothesis in **H**
2. **For** each positive training instance **x**
    - **For** each attribute constraint $\mathbf{a_i}$ in **h**

        If the constraint $\mathbf{a_i}$ in **h** is satisfied by **x**
        Then do nothing
        Else replace $\mathbf{a_i}$ in **h** by the next more general constraint that is satisfied by **x**
3. Output hypothesis **h**

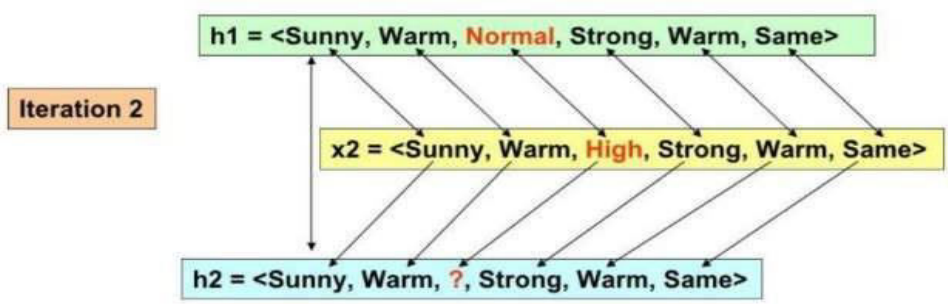
Find-S Algorithm Illustration

Step 1: Find-S

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

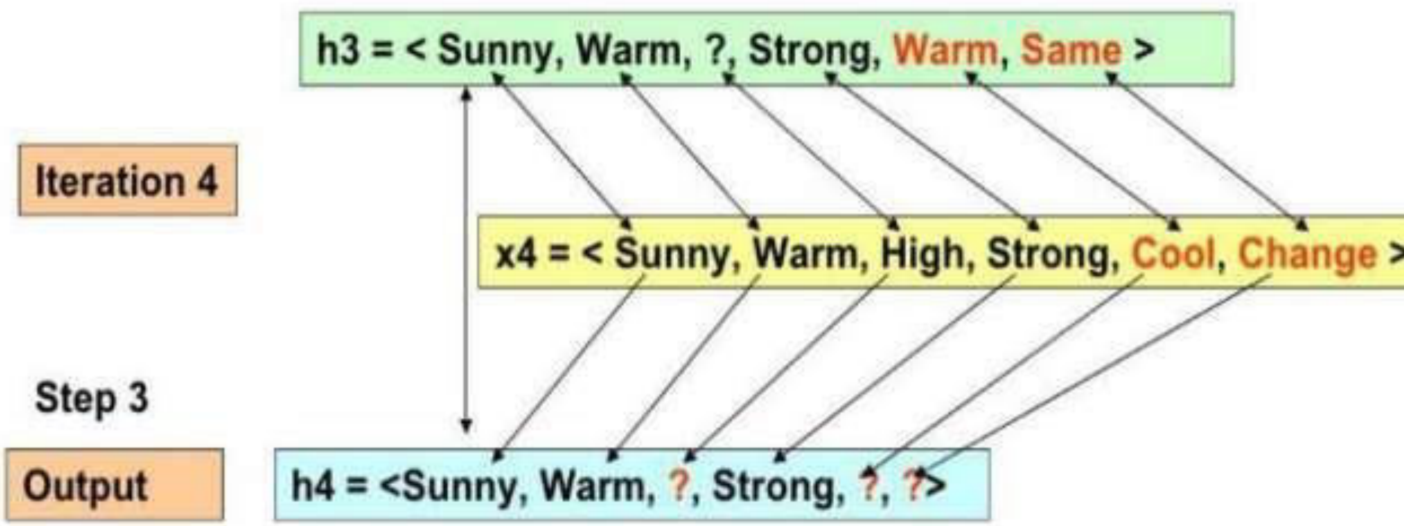


Step 2: Find-S



Find-S Algorithm cont.

Iteration 4 of Step 3: Find-S



Candidate Elimination Algorithm

Objective	To find most specific hypothesis in set of hypotheses, which is consistent with positive and negative training example.
Dataset	Tennis data set: This data set contains the set of examples days on which playing of tennis is possible or not, based on attributes *Sky, AirTemp, Humidity, Wind, Water and Forecast*. The dataset has 14 instances
ML Algorithm	Supervised Learning- Candidate-Elimination Algorithm
Description	The Candidate-Elimination Algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Candidate Elimination Algorithm

Algorithm:

G ← maximally general hypotheses in **H**

S ← maximally specific hypotheses in **H**

For each training example d = < x, c(x) >

Case 1: If d is a positive example

Remove from **G** any hypothesis that is inconsistent with **d**

For each hypothesis **s** in **S** that is not consistent with **d**

- Remove s from **S**.
- Add to **S** all minimal generalizations **h** of **s** such that
  - **h** consistent with **d**
  - Some member of **G** is more general than **h**
- Remove from **S** any hypothesis that is more general than another hypothesis in **S**

Case 2: If d is a negative example

Remove from **S** any hypothesis that is inconsistent with **d**

For each hypothesis **g** in **G** that is not consistent with **d**

- Remove **g** from **G**.
- Add to **G** all minimal specializations **h** of **g** such that
  - **h** consistent with **d**
  - Some member of **S** is more specific than **h**
- Remove from **G** any hypothesis that is less general than another hypothesis in **G**

Candidate Elimination Algorithm

Candidate Elimination Algorithm (works for +ve and –ve examples)

1. Specific hypothesis(S) - Φ
2. General hypothesis(G) - ?
3. Version space (contradiction)

Step1: Initialize G and S as most

general and specific hypothesis

Step2: for each example E

If E is positive (+ve):

Make specific hypothesis more general (works like Find S) Else

Make general hypothesis more specific

Candidate Elimination Algorithm

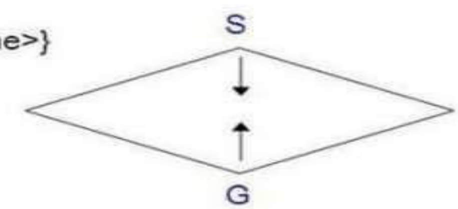
Illustration

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

S_0 = {<∅, ∅, ∅, ∅, ∅, ∅>}
G_0 = {<?, ?, ?, ?, ?, ?>}

S_1 = {<Sunny, Warm, Normal, Strong, Warm, Same>}
G_1 = {<?, ?, ?, ?, ?, ?>}

S_2 = {<Sunny, Warm, ?, Strong, Warm, Same>}
G_2 = {<?, ?, ?, ?, ?, ?>}



Trace1 :

Training examples:

1 . *<Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Yes*

2 . *<Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes*

Candidate Elimination Algorithm contd.

CANDIDATE-ELIMINATION Trace 1. S_0 and G_0 are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the S boundary to become more general, as in the FIND-S algorithm. They have no effect on the G boundary.

S_2: {<*Sunny, Warm, ?, Strong, Warm, Same*>}

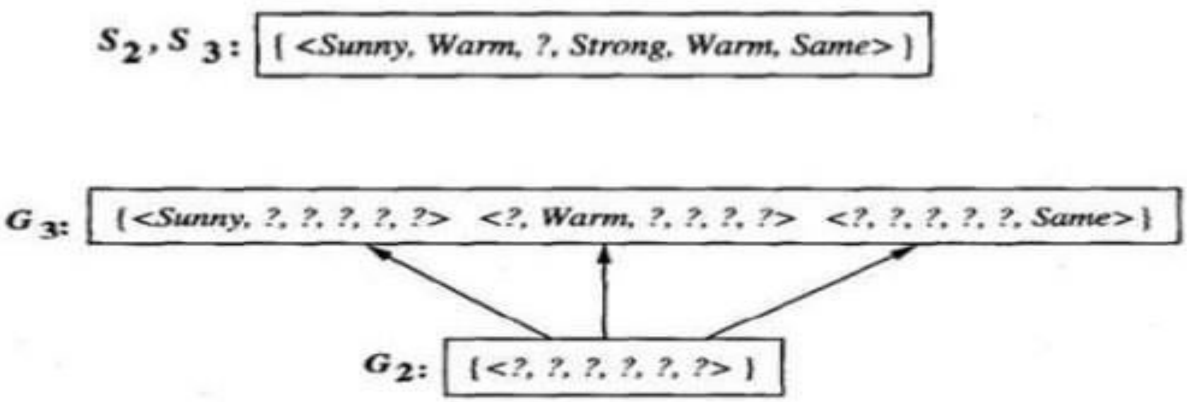
G_0, G_1, G_2: {<?, ?, ?, ?, ?, ?>}

Training examples:

1. <*Sunny, Warm, Normal, Strong, Warm, Same*>, *Enjoy Sport* = *Yes*
2. <*Sunny, Warm, High, Strong, Warm, Same*>, *Enjoy Sport* = *Yes*

Candidate Elimination Algorithm contd.

Trace 2:



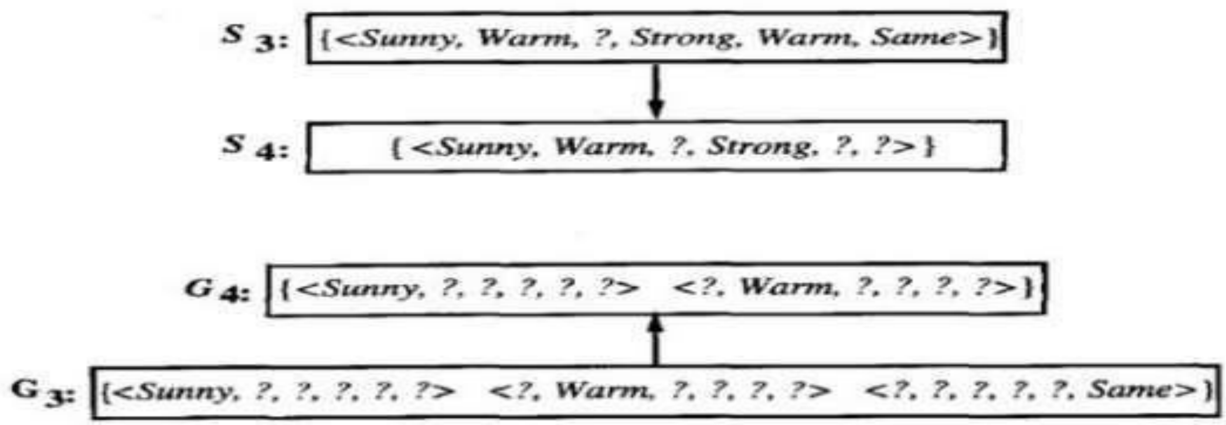
Training Example:

3. *<Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No*

CANDIDATE-ELIMINATION Trace 2. Training example 3 is a negative example that forces the boundary to be specialized to G_3. Note several alternative maximally general hypotheses are incluc in G_3.

Candidate Elimination Algorithm contd.

Trace3 :



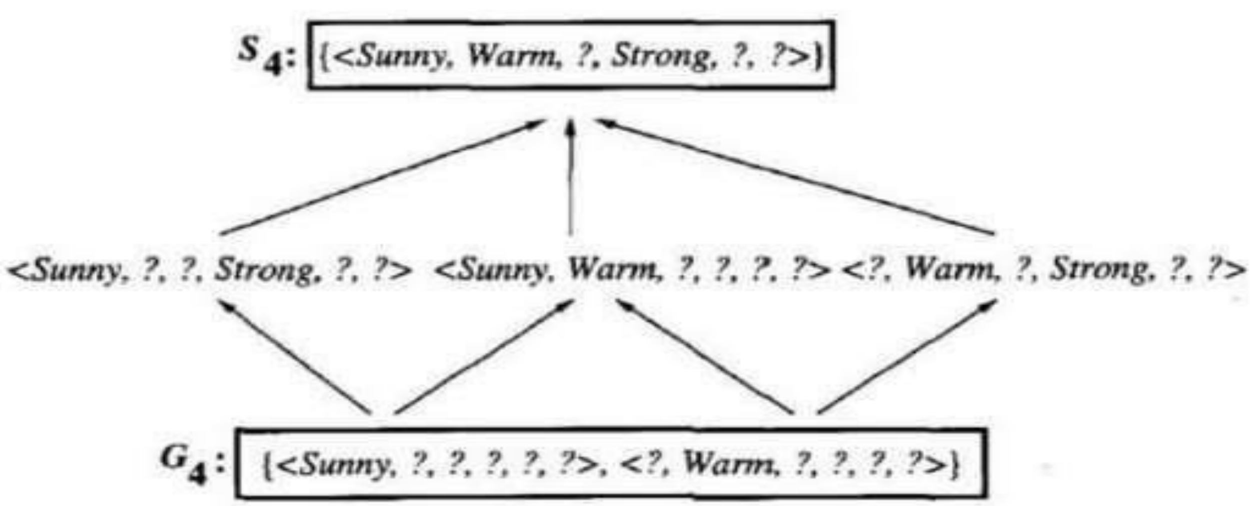
Training Example:

4. *<Sunny, Warm, High, Strong, Cool, Change>, EnjoySport = Yes*

CANDIDATE-ELIMINATION Trace 3. The positive training example generalizes the S bounda
S_3 to S_4. One member of G_3 must also be deleted, because it is no longer more general tha
boundary.

Candidate Elimination Algorithm contd.

Final Version Space:



The final version space for the *EnjoySport* concept learning problem and training examples des earlier.

Inductive Bias

Remarks on CE and VS algorithms:

1. Will the CE algorithm gives us correcthypothesis?

2. What training example should the learner request next?

Inductive learning: From examples we derive rules (feeding examples to machines)

Deductive learning: Already existing rules areapplied to our examples

Biased and Unbiased Hypothesis Space

Biased Hypothesis space

Does not consider all types of training examplesSolution: include all hypothesis

Example:

sunny^warm^normal^strong^cool∨change=yesUnbiased Hypothesis space

Providing a hypothesis capable of representing set of allexamples

Possible instances: 3X2X2X2X2X2=96

Target concepts: 2^96 (huge and practically not possible)

Idea of Inductive Bias

- The learner generalizes beyond the observed training examples to infer new examples.
- “>” : Inductively inferred from
- Example: x>y: y is inductively inferred fromx(predefined in the system)

The futility of Bias-free learning:

- Learning algorithm: L
- Training data: Dc={x,c(x)}
- New instance=Xi
- Represented as L(xi,Dc)
- **(Dc^xi) > L(xi,Dc)** (L is inferred from existing system)

MODULE-3

ARTIFICIAL NEURAL NETWORK

Chapters: 4.1 – 4.6

TOPICS INCLUDED

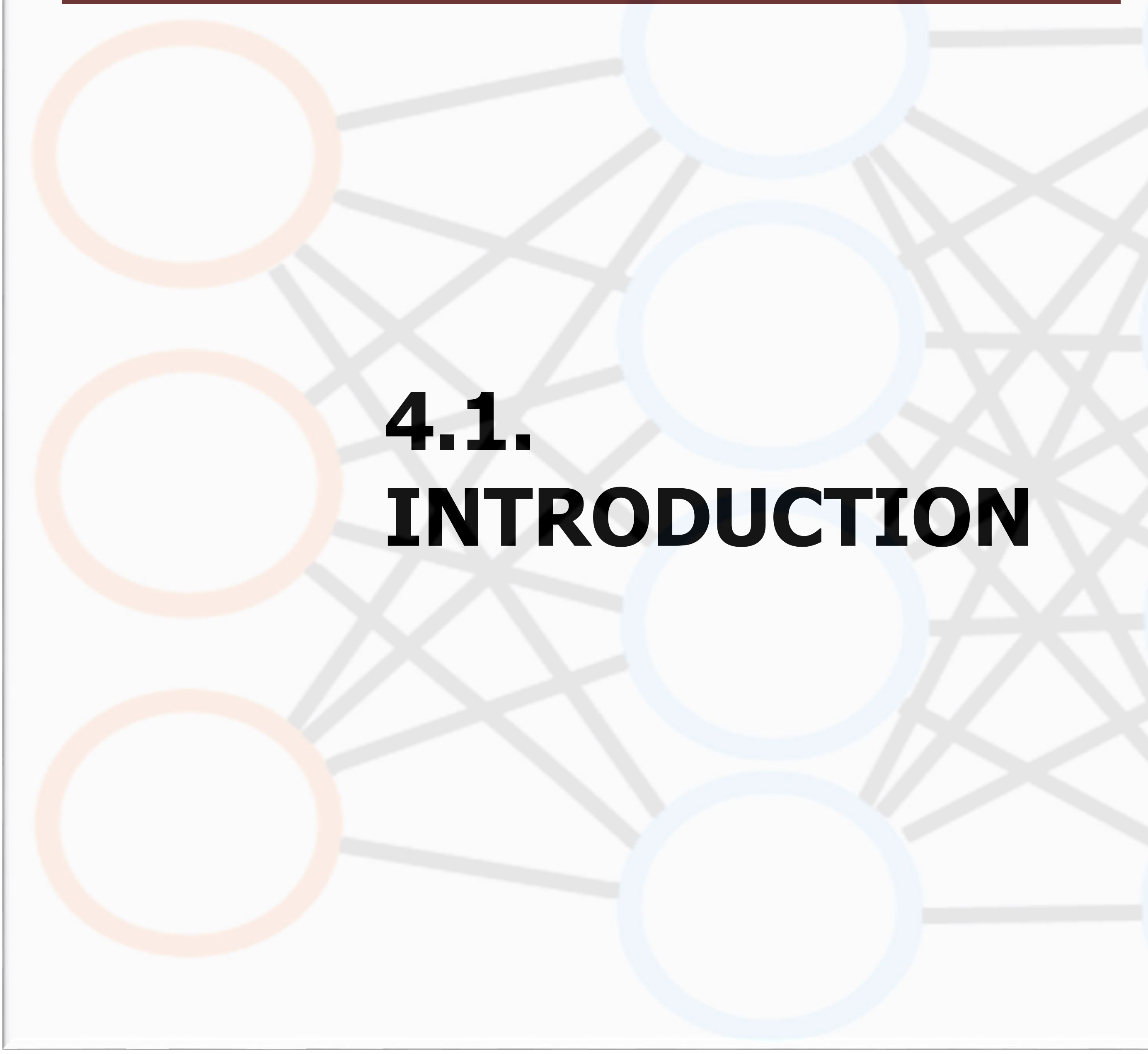
4.1. Introduction

4.2. Neural Network Representation

4.3. Appropriate Problems

4.4. Perceptrons

4.5. Backpropagation Algorithm


4.1.
INTRODUCTION

What are ANNs?

- ***ANN is an information-processing model that works by taking up the data from sensors as input, apply conventional methods (activation functions) to it and finally produce appropriate results (solutions) out of it.***

- To develop a computational device for modelling just like the brain, to perform various tasks such as

  pattern-matching and classification, optimization function, approximation, & data clustering.

- ANN consists of 3 layers: ***input layer, hidden layer and output layer.***

- Input Layer: Consists of Input neurons, represented as ***X1, X2,……..,Xn***

- Hidden Layer: Consists of hidden neurons, represented as ***Z1, Z2, Zk***

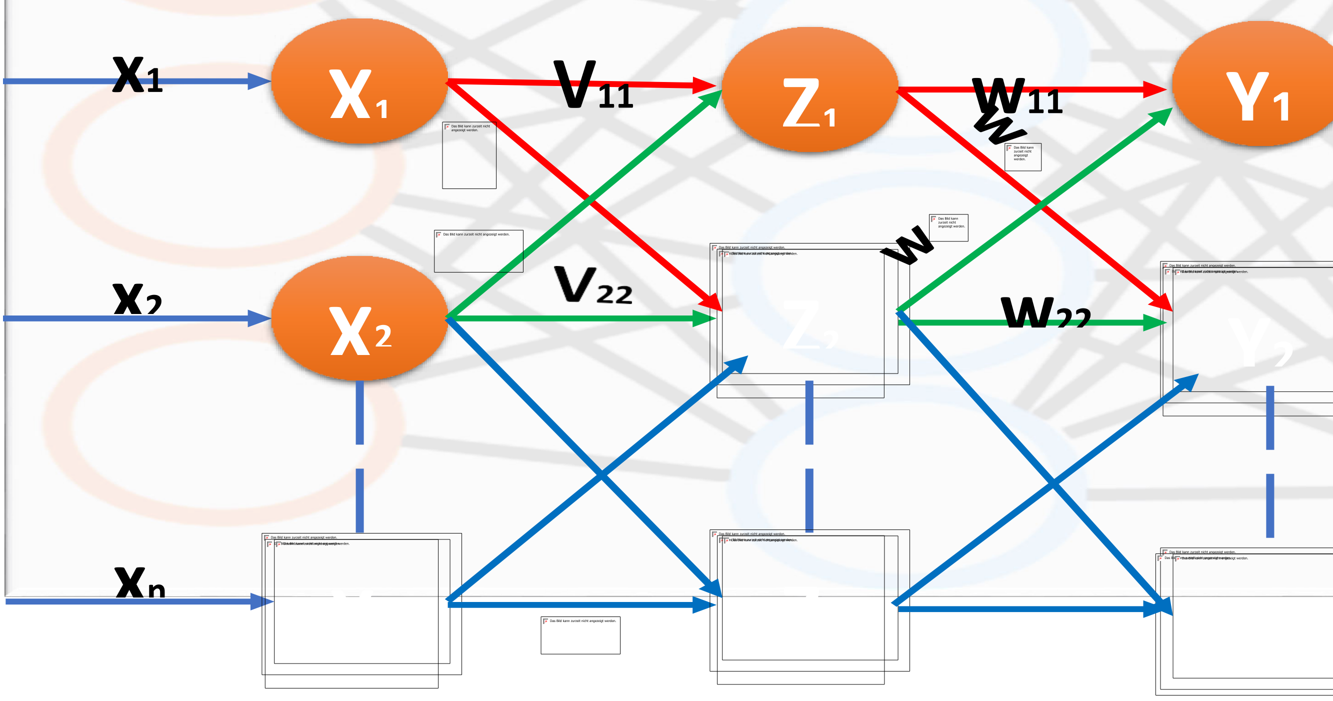
- Output Layer: Consists of output neurons, represented as ***Y1,Y2,……Ym***

- ***Neural network learning*** methods provide a robust approach to approximating ***real-valued, discrete***

  valued, and vector-valued target functions.

4.1.INTRODUCTION

- **Below figure shows the general representation of an ANN with one hidden layer at least.**



1. https://youtu.be/_aCCsRCw78g: Introduction: Neuroanatomy VideoLab - Brain Dissections (6:05 Secs)
2. https://youtu.be/1aplTvEQ6ew: Expressive Aphasia - Sarah Scott - Teenage Stroke

Survivor

3. https://youtu.be/PHQhCiVLRpE: Creating Virtual Humans: The Future of AI
4. https://youtu.be/eAwqB9W-HQ4: Baby X world showcase comingto TEDxAuckland 2013
5. https://youtu.be/yzFW4-dvFDA

4.1. INTRODUCTION

- This model, the net input is clarified as:

$$y_{in} = x_1 w_1 + x_2 w_2 + - - - - - + x_n w_n = \sigma_{i=}^{n}$$

Where **i** represents the **i**th processing element. The activation function is applied over it to calculate the output. The weight represents the strength of synapse connecting the input and output neurons.

A positive weight corresponds to an **excitatory synapse**, and a **negative weight** corresponds to an **inhibitory synapse.**

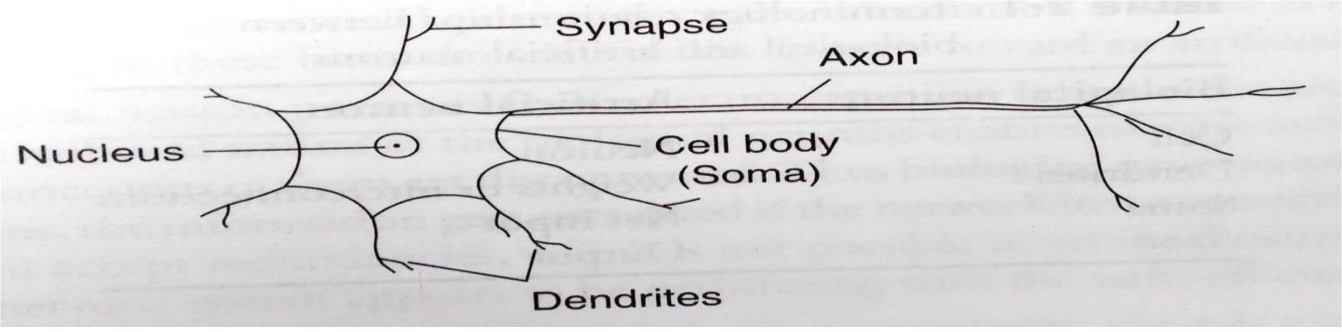
Some of the applications of ANN are: Face recognition, Visual Interpretation, Speech recognition, and learning robot control strategies.

4.2. BIOLOGICAL MOTIVATION

- The study of artificial neural networks (ANNs) has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons.
- Informally, artificial neural networks are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output (which may become the input to many other units).

4.2. BIOLOGICAL MOTIVATION

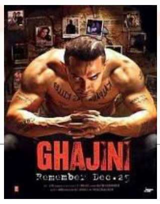
- Human brain consists of a huge number of neurons, approximately 10^{11}, Withnumerous interconnections.
- A schematic diagram of a biological neuron is shown in Fig:



4.2. BIOLOGICAL MOTIVATION

- A comparison could be made between biological neurons andArtificial neurons on the basis of the following criteria:

Speed	
Processing	**ANN**- processing is faster than brain simultaneously. But, in general, th
Size and Complexity	The total no of neurons in the brain is ab The size and complexity of a **biologica**
Storage Capacity	**BN**- stores the info in its interconnectio



4.2. BIOLOGICAL MOTIVATION

Das Bild kann zurzeit nicht angezeigt werden.
Das Bild kann zurzeit nicht angezeigt werden.

Tolerance	**BN**- possesses fault tolerant capability, **ANN** has no fault tolerance. T information even when the inter In case of AN, the information gets corrupted if the network interconne Biological neurons can accept redundancies, which is not possible in AN
Control Mechanism	The control mechanism of ANN is very simple compared to that of a BN cannot be.

4.3. APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

- ***Artificial Neural Network Learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data,such as inputs from cameras and microphones.***

Automatic Vehicle

ALVINN

In Neural Network

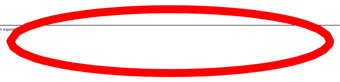


Fig: 1

4.3. APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

- ALVINN: A neural network learning based steering autonomous vehicle.
- The ALVINN system uses Back-Propagation Algorithm, to
  - Learn
  - Steer, an autonomous vehicle. (Shown in Fig: 1)
- Driving at the speed up to 70 MPH (113 KMPH)
- Fig: 2 shows, how the image of a fed forward to ***4 hidden units, connected to 30 output units.***
- Network outputs encode the commanded steering direction.
- Fig: 3 shows, weight values for one of the hidden units in the network.
- The 30 x 32 weights into the hidden unit are displayed in the large matrix, with white blocks indicating positive weights, and black indicating negative weights.

4.3. APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

- The weights from this hidden unit to the 30 output units are depicted by the smaller rectangular block. (Above, Fig: 3).
- As can be seen from these output weights, activation of this particular hidden unit encourages a turn towards the left.

4.3. APPROPRIATE PROBLEMS FORNEURAL NETWORK LEARNING

- The back-propagation algorithm or ANN is appropriate for problems with the following characteristics: (Explain with ALVINN example)

1. **Instances are represented by many attribute-value pairs (pixel values)**
2. **The target function output may be discrete-valued, real-valued, or a vector of severalreal-or discrete-valued attributes.** (vector of 30 attributes→ steering direction)
3. **The training examples may contain errors.** (robust to training data)(mostly in beginning stage)
4. **Long training times are acceptable.** (Training times can range from a few seconds to many hours)
5. **Fast evaluation of the learned target function may be required.**

(ALVINN applies its neural network several times per second to continually update its steering command as the vehicle drives forward.)

6.

The ability of humans to understand the learned target function is not important.

Das Bild kann zurzeit nicht angezeigt werden.

What are the appropriate proble

PERCEPTRONS

- A basic type of ANN system based on a unit called a ***perceptron.***
- Illustrated in below figure:

Wh

Fig: A Perceptron

PERCEPTRONS

- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.
- More precisely, given inputs ***x1*** through ***xn,*** the output ***o(x1, . . . , x,)***
  computed by the perceptron is:

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 \\ -1 & \text{otherwise} \end{cases}$$

4.4.PERCEPTRONS

- To simplify notation, we imagine the constant input x0 = 1, allowing us to write as:

$$\sum_{i=0}^{n}$$

- Or, in vector form as:

$$\vec{w} \cdot \vec{x} > 0.$$

- However, for perceptron function representation we write as:

PERCEPTRONS

- Learning a perceptron involves choosing values for the weights w0,…….wn.
- Therefore, the space ***H*** *candidate hypotheses* considered in perceptron learning is the set of all possible real-valued weight vectors. (except imaginary numbers)

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

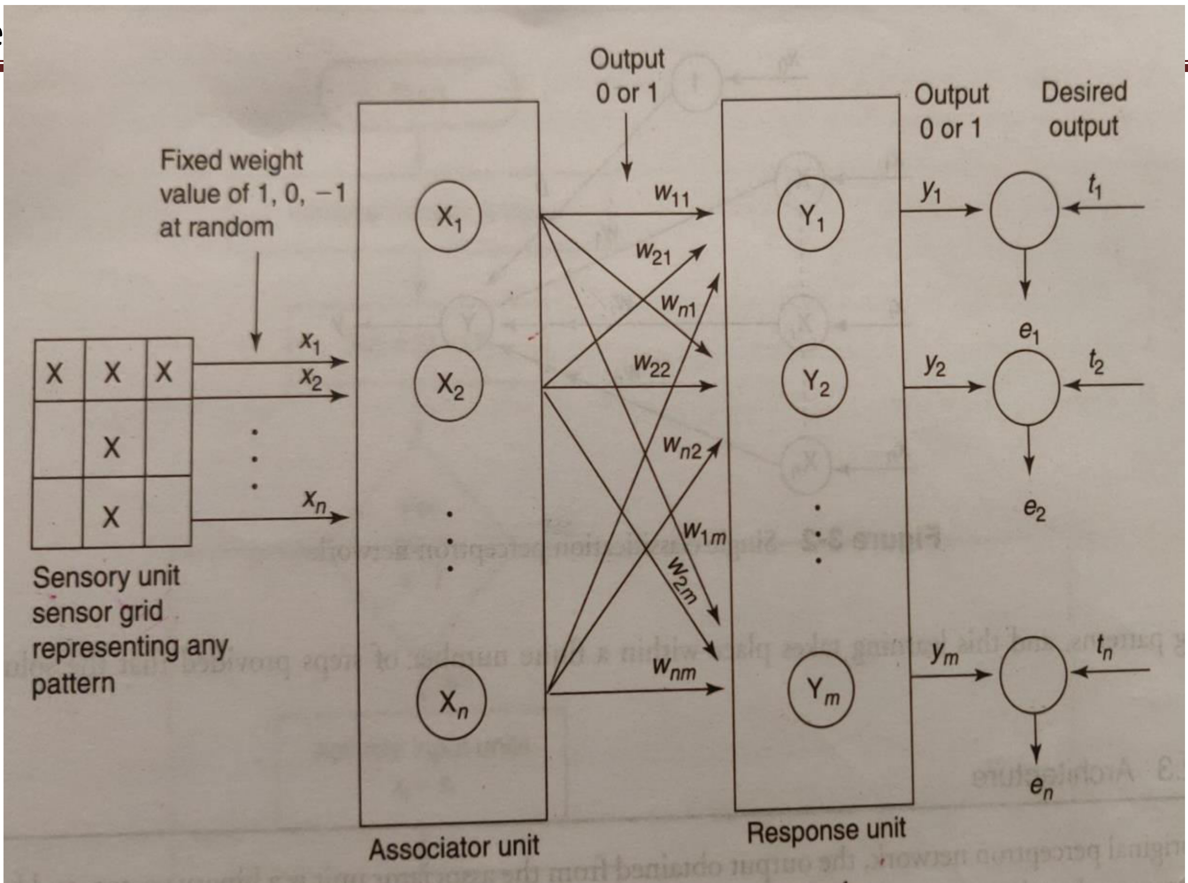


Figure 3-1 Original perceptron network.

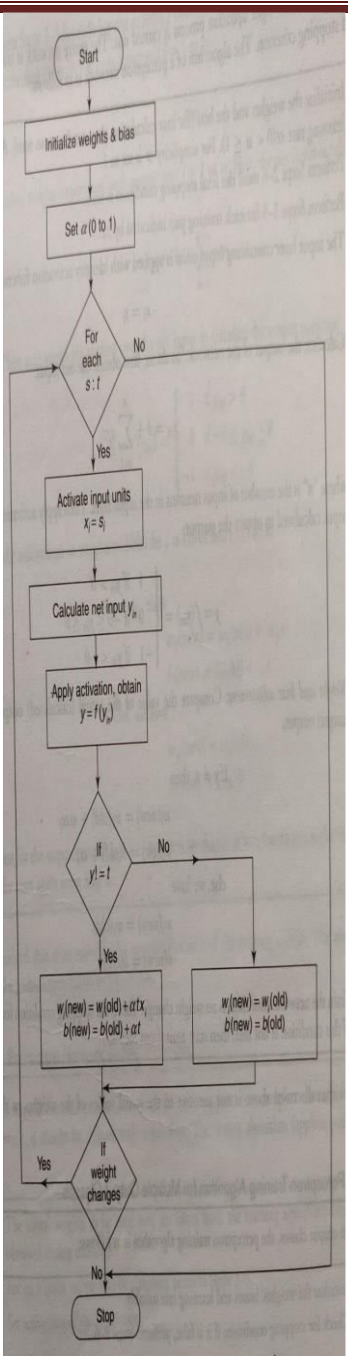


Figure 3-3 Flowchart for perceptron network with single output.

REPRESENTATIONAL POWER OF PERCEPTRONS

- We can view the perceptrons as representing a hyperplane decision surface in the n-dimensional space of instances (i.e. points).
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side,
-

The decision surface represented by a two-input perceptron.

Li

n

e

a

rl

y

s

e

p

a

r

a

b

le

a

n

d

n

o

t

li

n

e

a

rl

y

s

e

p

a

r

a

b

le

P

o

si

ti

v

e

e

x

a

m

p

le

s

→

+

,

N

e

g

a

ti

v

e

e

x

a

m

p

le

s

→

-

X

1

,

x

2

→

P

e

r

c

e

p

tr

o

n

I

n

p

u

t

REPRESENTATIONAL POWER OF PERCEPTRONS

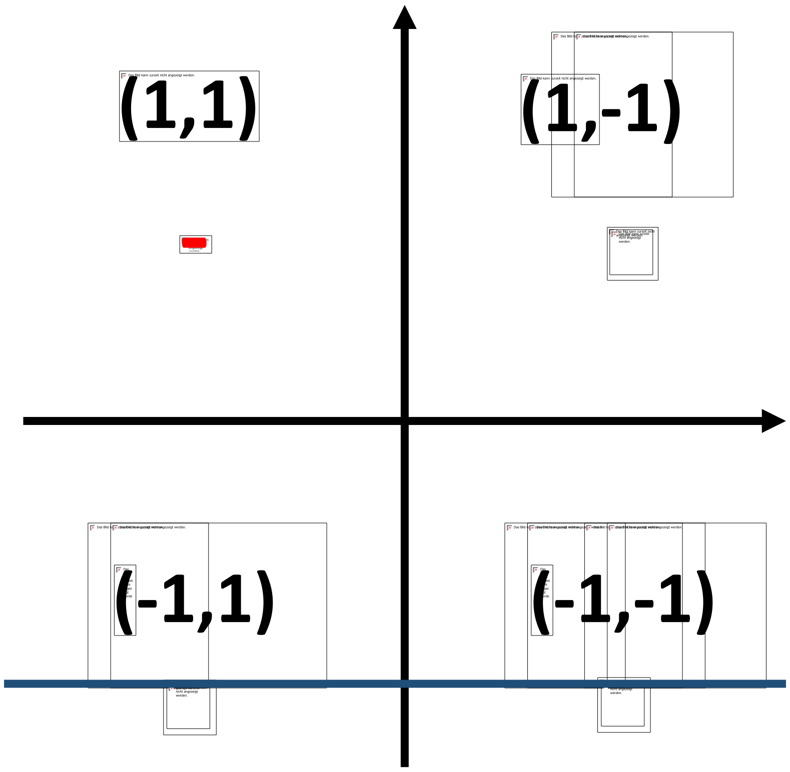
- A single perceptron can be used to represent many Boolean functions.
- For ex: **AND** & **OR** function

Discuss, how a single perceptron can used to represent the Boolean Functions such as AND,OR.

(Problem Solving)

1	1	1	1	0	0
1	0	1	0	2	1
0	1	1	0	2	1
0	0	1	0	1	1

Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	0	1	1	2	1	0	0	0	1	1	1
0	1	1	1	2	1	0	0	0	1	1	[illegible]
0	0	1	−1	1	1	0	0	−1	1	1	0



THE PERCEPTRON TRAININGRULE

- How to learn the weights for a single perceptron?
- Here the precise learning problem is **to determine a weight vector** that causes the perceptron to produce the correct **± 1 output** for each of the given training examples.
- Consider two: **the perceptron rule** and **the delta rule** (a variant of the LMS rule used for learning evaluation functions).
- **Why:** They are important to ANNs because they provide the basis for learning networks of many units.

THE PERCEPTRON TRAININGRULE

One way to learn an acceptable weight vector is to:

1. Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.

2. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.

3. Weights are modified at each step according to the ***perceptron training rule,*** which updates the weight ***wi*** associated with input ***xi*** according to the rule: $w_i \leftarrow w_i + \Delta w_i$

4.4.2.THE PERCEPTRON TRAININGRULE

Where: ***wi = η* (t – o) xi**

- **So, why should this update rule converge towards successful weight values?**
- Consider a specific case, where a perceptron correctly classifies training examples. So in this case, the error ***(t – o) = 0 → making Δwi = 0.***

Here no weights are updated

- Consider another case, where a perceptron outputs a **-1, when t = +1.** Now, in order to make a perceptron output a **+1,** instead of **-1,** the weights must be altered to increase the value of $m \cdot \boldsymbol{x}$.

THE PERCEPTRON TRAININGRULE

- If **xi > 0,** then increasing **wi** will bring the perceptron closer to correctly classifying
  this example.
- **Now, training will increase wi, as η and xi are all positive.**
- Ex: if **xi = 0.8, η = 0.1, t = 1, and o = -1, then the weight update will be:**

$$\Delta wi = \eta\,(t - o)\,xi$$
$$\rightarrow \Delta wi = 0.1\,(1 + 1)\,0.8$$
$$\rightarrow \Delta wi = 0.16$$

- On the other hand: if **xi = 0.8, η = 0.1, t = -1, and o = 1, then the weight update will be:**

$$\Delta wi = \eta\,(t - o)\,xi$$
$$\rightarrow \Delta wi = 0.1\,(-1 + 1)\,0.8$$
$$\rightarrow \Delta wi = 0$$

GRADIENT DESCENT RULE ANDDELTA RULE

- Although the **perceptron training rule** finds a successful weight vector when the training examples are linearly separable, **it can fail to converge** if the examples are **not linearly separable.**
- **A second training rule, called the delta rule overcomes the difficulty faced by perceptron training rule.**
- **If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.**
- The key idea behind the delta rule is to use ***gradient-descent*** to search the hypothesis space of possible weight **vectors to find**

the weights that best fit the training examples.

GRADIENT DESCENT RULE ANDDELTA RULE

- This rule is important because gradient descent provides the basis for the Back Propagation algorithm, which can learn networks with many interconnected units.
- It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypotheses.
- The delta training rule is best understood by considering the task of training an ***unthresholded*** perceptron; that is, a ***linear unit*** for which the output ***o*** is given by:

$$\boldsymbol{o} = m \cdot \boldsymbol{x} \qquad (\vec{\boldsymbol{x}})$$

GRADIENT DESCENT RULE ANDDELTA RULE

- In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.
- Although there are many ways to define this error, one common measure that will turn out to be especially convenient is

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d$$

- Where D is the set of training examples, t_d is the target output for training example *d*, and o_d is the output of the linear unit for training example d.
- By this definition, $E(\vec{w})$ is simply half the squared difference between the target output t_d and the linear unit output o_d, summed over all training examples.
- Here we characterize E as a function of $\vec{w}$, because the linear unit output o depends on this weight vector.

GRADIENT DESCENT RULE ANDDELTA RULE

- In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.
- Although there are many ways to define this error, one common measure that will turn out to be especially convenient is

$$E(\vec{w}) \equiv \frac{1}{2} \; (t_{d} \ldots d \in$$

- Where D is the set of training examples, t_d is the target output for training example

d, and o_d is the output of the linear unit for training example d.

- By this definition, $E(\vec{w})$ is simply half the squared difference between the target output t_d and
  the linear unit output o_d, summed over all training examples.
- Here we characterize E as a function of $\vec{w}$, because the linear unit output o depends on this weight vector.

MODULE -4

BAYESIAN LEARNING

CONTENT

- Introduction
- Bayes theorem
- Bayes theorem and concept learning
- Maximum likelihood and Least Squared Error Hypothesis (ML&LS)
- Maximum likelihood Hypotheses for predicting probabilities
- Minimum Description Length Principle (MDL)
- Naive Bayes classifier
- Bayesian belief networks (BBN)
- EM algorithm

INTRODUCTION

Bayesian reasoning provides a probabilistic approach to inference. These are governed by probabilistic distributions and optimal decisions can be made by reasoning.

Bayesian learning methods are relevant to study of machine learning for two different reasons.

- First, Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practicalapproaches to certain types of learning problems
- The second reason is that they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities.

Features of Bayesian Learning Methods

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting (1) a prior probability for each candidate hypothesis, and (2) a probability distribution over observed data for each possible hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

Practical difficulty in applying Bayesian methods

- One practical difficulty in applying Bayesian methods is that they typically require initial knowledge of many probabilities. When these probabilities are not known in advance they are often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions.

- A second practical difficulty is the significant computational cost required to determine the Bayes optimal hypothesis in the general case. In certain specialized situations, this computational cost can be significantly reduced.

BAYESTHEOREM

Bayes theorem gives the probability of an event based on prior knowledge of conditions. P(A/B)=[P(B/A).P(A)] / P(B)

P(A/B)= hypothesis;

P(B/A)=likelihood;

P(A)=prior;

P(B)=marginal Proof of Bayes theorem:

P(A/B)=P(A∩B)/P(B). So

P(A∩B)=P(A/B).P(B) -----------------------------------

P(B/A)=P(B∩A)/P(A). So

P(B∩A)=P(B/A).P(A) -----------------------------------

LHS are equal therefore RHS are also equal

P(A/B).P(B)=P(B/A).P(A)

Hence P(A/B)=[P(B/A).P(A)] / P(B)

Terms:

A=hypothesis; B=given data;

P(A/B)=Finding probability of hypothesis when probability of training example is given.

P(B/A)=Finding probability of given data when provided with probability of hypothesis that is true.

BAYESTHEOREM

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Notations

- P(h) *prior probability of* h, reflects any background knowledge about the chancethat h is correct
- P(D) *prior probability of* D, probability that D will be observed
- P(D|h) probability of observing D given a world in which h holds
- P(h|D) *posterior probability of* h, reflects confidence that h holds after D has beenobserved

Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability ***P(h|D),*** from the prior probability ***P(h),*** together with ***P(D)*** and ***P(D(h).***

Bayes Theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

P(h|D) increases with ***P(h)*** and with ***P(D|h)*** according to Bayes theorem.

P(h|D) decreases as ***P(D)*** increases, because the more probable it is that ***D*** will be
observed independent of ***h,*** the less evidence ***D*** provides in support of ***h.***

Example on Bayes Theorem

Q. What is the probability that person that person has disease dengue with neck pain.

Solun:

Given:

80% of time dengue causes neck pain:p(a/b)=0.8

P(dengue-b)=1/30,000: p(b)=1/30,000

P(neck pain-a)=0.2: p(a)=0.02

a =

probability that person has neck pai**n

b = person has dengue

$$P(b/a)=[p(a/b)\ p(b)] / p(a)$$
$$=[0.8 * 1/30{,}000] / 0.02$$
$$=0.00133$$

Maximum a Posteriori (MAP) Hypothesis

- In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis ***h*** ∈***H*** given the observed data ***D***. Any such maximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.
- Bayes theorem to calculate the posterior probability of each candidate hypothesis is h_{MAP}
  is a MAP hypothesis provided

$$
\begin{aligned}
h_{MAP} &= \underset{h \in H}{argmax}\ P(h|D) \\
&= \underset{h \in H}{argmax}\ \frac{P(D|h)P(h)}{P(D)} \\
&= \underset{h \in H}{argmax}\ P(D|h)P(h)
\end{aligned}
$$

- P(D) can be dropped, because it is a constant independent of h

Maximum Likelihood (ML) Hypothesis

In some cases, it is assumed that every hypothesis in H is equally probable a priori
($P(h_i)$ = $P(h_j)$ for all ***h_i*** and ***h_j*** in H).
In this case the below equation can be simplified and need only consider the term
P(D|h) to find the most probable hypothesis.

$$h_{MAP} = \underset{h \in H}{argmax}\ P(D|h)P(h)$$

the equation can be simplified

$$h_{ML} = \underset{h \in H}{argmax}\ P(D|h)$$

P(D|h) is often called the ***likelihood*** of the data ***D*** given ***h,*** and any hypothesis that
maximizes ***P(D|h)*** is called a ***maximum likelihood*** (ML) hypothesis

Example on MAP

Consider a medical diagnosis problem in which there are two alternative hypotheses

(1) The patient has a particular form of cancer (denoted by ***cancer***)

(2) The patient does not (denoted by ¬ ***cancer***)

A patient takes a lab test and the result comes back positive. The test results a correct positive result in only 98% of cases in which the disease is actually present and a correct negative result in only 97% in which the disease is not present. Further more 0.008 of the entire population have this cancer. Determine whether the patient has a cancer or not using MAP hypothesis.

Solution:

The available data is from a particular laboratory with two possible outcomes: + (positive) and

- (negative)

$$P(cancer) = .008 \qquad P(\neg cancer) = 0.992$$

$$P(\oplus|cancer) = .98 \qquad P(\ominus|cancer) = .02$$

$$P(\oplus|\neg cancer) = .03 \qquad P(\ominus|\neg cancer) = .97$$

- Suppose a new patient is observed for whom the lab test returns a positive (+) result.
- We diagnose the patient as not having cancer because the negative probability is more.

$$P(\oplus|cancer)P(cancer) = (.98).008 = .0078$$
$$P(\oplus|\neg cancer)P(\neg cancer) = (.03).992 = .0298$$
$$\Rightarrow h_{MAP} = \neg cancer$$

BAYESTHEOREM AND CONCEPTLEARNING

What is the relationship between Bayes theorem and the problem of concept learning?

Since Bayes theorem provides a principled way to calculate the posterior probabilityof each hypothesis given the training data p(h/D), and can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable.

Brute-Force Bayes Concept Learning

We can design a straightforward concept learning algorithm to output the maximuma posteriori hypothesis, based on Bayes theorem, as follows:

Brute-Force MAP LEARNING algorithm

1. For each hypothesis h in H calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \underset{h \in H}{argmax}\, P(h|D)$$

In order specify a learning problem for the **BRUTE-FORCE MAP LEARNING** algorithm we must specify what values are to be used for ***P(h)*** and for ***P(D|h) ?***

Lets choose ***P(h)*** and for ***P(D|h)*** to be consistent with the following assumptions:

- The training data ***D*** is noise free (i.e., $d_i = c(x_i)$)
- The target concept ***c*** is contained in the hypothesis space H
- We have no a priori reason to believe that any hypothesis is more probable than any other.

What values should we specify for ***P(h)?***

- Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis ***h*** in ***H.***
- Assume the target concept is contained in

$$P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

H and require that these priorprobabilities sum to 1.

What choice shall we make for ***P(D|h)***?

- ***P(D|h)*** is the probability of observing the target values D = ***(d_1 . . .d_m)*** for the fixed set of instances ***(x_1 . . . x_m),*** given a world in which hypothesis ***h*** holds
- Since we assume noise-free training data, the probability of observing classification ***d_i*** given ***h*** is just ***1 (consistent)*** if ***d_i = h(x_i)*** and 0(inconsistent) if ***d_i # h(x_i).*** Therefore,

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in \\ 0 & \text{otherwise} \end{cases}$$

Given these choices for ***P(h)*** and for ***P(D|h)*** we now have a fully-defined problem for the above **BRUTE-FORCE MAP LEARNING** algorithm.

In a first step, we have to determine the probabilities for P(h|D)

h is **inconsistent** with training data D

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$$

h is **consistent** with training data D

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} = \frac{1}{|VS_{H,D}|}$$

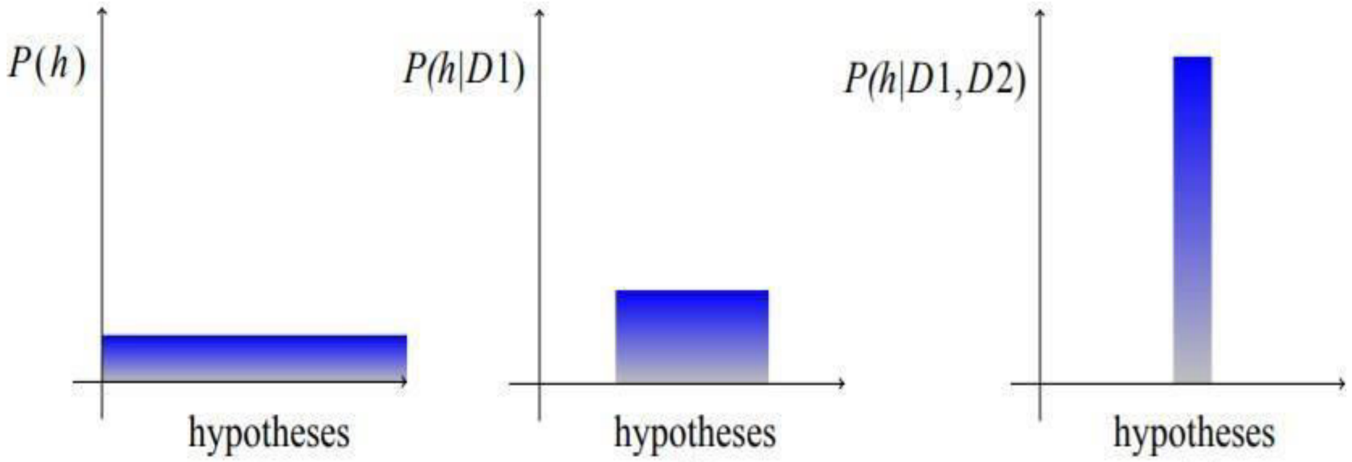
To summarize, Bayes theorem implies that the posterior probability P(h|D) underour assumed P(h) and P(D|h) is

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$$

where $|VS_{H,D}|$ is the number of hypotheses from H consistent with D(version space)

The Evolution of Probabilities Associated with Hypotheses

- Figure (a) all hypotheses have the same probability.
- Figures (b) and (c), As training data accumulates, the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to 1 isshared equally among the remaining consistent hypotheses.



(a) (b) (c)

MAP Hypotheses and ConsistentLearners

A learning algorithm is a consistent learner if it outputs a hypothesis that commits zero errors over the training examples.

Every consistent learner outputs a MAP hypothesis, if we assume a uniform prior probability distribution over H ($P(h_i) = P(h_j)$ for all i, j), and deterministic, noise free training data ($P(D|h) = 1$ if D and h are consistent, and 0 otherwise).

Example:

- FIND-S outputs a consistent hypothesis, it will output a MAP hypothesis under the probability distributions P(h) and P(D|h) defined above.
- Are there other probability distributions for P(h) and P(D|h) under which FIND- S outputs MAP hypotheses? Yes.
- Because FIND-S outputs a maximally specific hypothesis from the version space, its output hypothesis will be a MAP hypothesis relative to any prior probability distribution that favours more specific hypotheses.

- Bayesian framework is a way to characterize the behaviour of learning algorithms
- By identifying probability distributions P(h) and P(D|h) under which the output is a optimal hypothesis, implicit assumptions of the algorithm can be characterized (Inductive Bias)
- Inductive inference is modelled by an equivalent probabilistic reasoning system based on Bayes theorem

MAXIMUM LIKELIHOOD AND LEAST-SQUARED (ML and LS) ERROR HYPOTHESES

Consider the problem of learning a continuous-valued target function such as neural network learning, linear regression, and polynomial curve fitting

A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood (ML) hypothesis

Learning A Continuous-Valued TargetFunction

- Learner L considers an instance space X and a hypothesis space H consisting of some class ofreal-valued functions defined over X, i.e., $(\forall h \in H)[h : X \rightarrow R]$ and training examples of the form $<x_i,d_i>$
- The problem faced by L is to learn an unknown target function $f : X \rightarrow R$
- A set of m training examples is provided, where the target value of each example is corruptedby random noise drawn according to a Normal probability distribution with zero mean ($d_i = f(x_i) + e_i$)
- Each training example is a pair of the form (x_i, d_i) where $d_i = f(x_i) + e_i$.
  - Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise.
  - It is assumed that the values of the e_i are ***drawn independently*** and that they are distributedaccording to a ***Normal distribution*** with zero mean.
- The task of the learner is to output a ***maximum likelihood hypothesis***, or, equivalently, a MAPhypothesis assuming all hypotheses are equally probable a priori.

Using the previous definition of h_{ML} we have

$$h_{ML} = \underset{h \in H}{argmax}\ p(D|h)$$

Let us take training examples instances(x1,x2,..xn) and target values D=(d1,d2,…dm). We need to multiply all probabilities. Assuming training examples are mutually independent given h, we can write P(D|h) as the product of the various (d_i|h)

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^{m}$$

Here distribution of values can be binomial or distributed. Given the noise e_i obeys a Normal distribution with zero mean and

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2}$$

unknown variance σ^2 , each d_i must also obey a Normal distribution around the true targetvalue $f(x_i)$. Because we are writing the expression for P(D|h), we assume h is the correct description of f. Hence, mean $\mu = f(x_i) = h(x_i)$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-\mu)^2}$$

$$= \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-h(x_i))^2}$$

It is common to maximize the less complicated logarithm, which is justified because of the monotonicity of function *p, e is 1 will logarithm*.

$$= \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} ln\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of *h* and can therefore bediscarded

$$= \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative term is equivalent to minimizing the corresponding positiveterm.

$$= \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally Discard constants that are

$$h_{ML} = \underset{h \in H}{argmin} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

i n d e p e n d e n t o f *h*

- **the h_{ML} is one that minimizes the sum of the squared errors**

Why is it reasonable to choose the Normal distribution to characterize noise?

- **good approximation of many types of noise in physical systems**
- **Central Limit Theorem shows that the sum of a sufficiently large number of independent, identically distributed random variables itself**

obeys a Normal distribution

Only noise in the target value is considered, not in the attributes describing the instancesthemselves

IMUM LIKELIHOOD HYPOTHESES FORPREDICTING PROBABILITIES

Maximum likelihood hypothesis is the one that minimizes the sum of squarederrors over the training examples.

Consider the setting in which we wish to learn a nondeterministic (probabilistic)function $f : X \rightarrow \{0, 1\}$, which has two discrete output values.

We want a function approximator whose output is the probability that $f(x) = 1$

(if hypothesis is correct:1

else 0) In other words , learn the target functionf' $: X \rightarrow [0, 1]$ such that $f'(x) = P(f(x)$

= 1)

How can we learn f' (how much prediction is correct) using a neural network?

Use of brute force way would be to first collect the observed frequencies of 1's and 0's for each possible value of x and to then train the neural network to output the target frequency for each x.

What criterion should we optimize in order to find a maximum likelihood hypothesisfor f' in this setting?

- First obtain an expression for P(D|h)
- Assume the training data D is of the form $D = \{(x_1, d_1) \ldots (x_m, d_m)\}$, where d_i isthe observed 0 or 1 value for $f(x_i)$.
- Both x_i and d_i as random variables, and assuming that each training example is drawn independently, we can write P(D|h) as

$$P(D \mid h) = \prod_{i=1}^{m} P(x_i, d_i \mid h)$$

Applying the product rule

$$P(D \mid h) = \prod_{i=1}^{m} P(d_i \mid h, x_i) P(x_i)$$

The probability $P(d_i|h, x_i)$

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases} \qquad \text{equ (3)}$$

Re-express it in a more mathematically manipulable form, as

$$P(d_i|h, x_i) = h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} \qquad \text{equ (4)}$$

Equation (4) to substitute for $P(d_i |h, x_i)$ in Equation (5) to obtain

$$P(D|h) = \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} P(x_i) \qquad \text{equ (5)}$$

We write an expression for the maximum likelihood hypothesis

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i)$$

The last term is a constant independent of h, so it can be dropped

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \qquad \text{equ (6)}$$

It easier to work with the log of the

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \qquad \text{equ (7)}$$

likelihood, yielding

Equation (7) describes the quantity that must be maximized in order to obtain the maximum likelihood hypothesis in our current problem setting

Gradient Search to Maximize Likelihood in a Neural Net

Derive a weight-training rule for neural network learning that seeks to maximize G(h, D) using gradient ascent

- The gradient of G(h, D) is given by the vector of partial derivatives of G(h, D) with respect to the various network weights that define the hypothesis h represented by the learned network
- In this case, the partial derivative of G(h, D) with

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \frac{\partial G(h,D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$= \sum_{i=1}^{m} \frac{\partial (d_i \ln h(x_i) + (1-d_i)\ln(1-h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$= \sum_{i=1}^{m} \frac{d_i - h(x_i)}{h(x_i)(1-h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}} \qquad \text{equ (1)}$$

respect to weight w_{jk} from input k to unit j is

Suppose our neural network is constructed from a single layer of sigmoid units. Then,

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x_i)x_{ijk} = h(x_i)(1 - h(x_i))x_{ijk}$$

where x_{ijk} is the k^{th} input to unit j for the i^{th} training example, and d(x) is the derivative of the sigmoid squashing function.

Finally, substituting this expression into Equation (1), we obtain a simple expression for the derivatives that constitute the gradient

$$\frac{\partial G(h, D)}{\partial w_{jk}} = \sum_{i=1}^{m}(d_i - h(x_i))\ x_{ijk}$$

Because we seek to maximize rather than minimize P(D|h), we perform ***gradient ascent*** rather than ***gradient descent search***. On each iteration of the search the weight vector is adjusted in the direction of the gradient, using the weight update rule

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where,

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk} \qquad \text{equ (2)}$$

where η is a small positive constant that determines the step size of the i gradient ascent search

It is interesting to compare this weight-update rule to the weight-update rule used by the BACKPROPAGATION algorithm to minimize the sum of squared errors between predicted and observed network outputs.

The BACKPROPAGATION update rule for output unit weights, re-expressed using our current notation, is

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where,

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} h(x_i)(1 - h(x_i))(d_i - h(x_i))\ x_{ijk}$$

MINIMUM DESCRIPTION LENGTH PRINCIPLE

- Representing a concept in a minimal way, then the concept is said to be good one.
- Motivated by interpreting the definition of h_{MAP} in the light of basic concepts from information theory.

$$h_{MAP} = \underset{h \in H}{argmax}\, P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the $\log_2$ i.e $\log(ab) = \log a + \log b$

$$h_{MAP} = \underset{h \in H}{argmax}\, \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{argmin}\, -\log_2 P(D|h) - \log_2 P(h)$$

- This equation can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data

MDL Example: Introduction to a basic result of information theory

- consider the problem of designing a code to transmit messages drawn at random from set D
- i is the message
- The probability of encountering message i is p_i
- Interested in the most compact code; that is, interested in the code that minimizes the expected number of bits we must transmit in order to encode a message drawn at random
- To minimize the expected code length we should assign shorter codes to messages that aremore probable
- The number of bits required to encode message i using code C as the ***description length of message i with respect to C***, which we denote by $L_C(i)$.

Interpreting the equation

$$h_{MAP} = \underset{h \in H}{argmin} \ -\log_2 P(D|h) - \log_2 P(h)$$

Rewrite Equation (1) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given thehypothesis.

$$h_{MAP} = \underset{h \in H}{argmin} \ L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

where C_H and $C_{D|h}$ are the optimal encodings for H and for D given h

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths of equ.

$$h_{MAP} = \underset{h \in H}{argmin}\, L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Minimum Description Length principle:

$$h_{MDL} = \underset{h \in H}{\text{argmin}}\, L_{C_1}(h) + L_{C_1}(D \mid h)$$

Where, codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis

The above analysis shows that if we choose C_1 to be the optimal encoding of hypotheses C_H, and ifwe choose C_2 to be the optimal encoding $C_{D|h}$, then $h_{MDL} = h_{MAP}$

Application to Decision Tree Learning

Apply the MDL principle to the problem of learning decision trees from some trainingdata.

What should we choose for the representations C_1 and C_2 of hypotheses and data?

- **For C_1:** C_1 might be some obvious encoding, in which the description length grows with the number of nodes and with the number of edges
- **For C_2:** Suppose that the sequence of instances (x_1 . . .x_m) is already known to both the transmitter and receiver, so that we need only transmit the classifications ($f(x_1)$ . . . $f(x_m)$).

  Now if the training classifications ($f(x_1)$ . . .$f(x_m)$) are identical to the predictions of the hypothesis, then there is no need to transmit any information about these examples. The description length of the classifications given the hypothesis ZERO

  If examples are misclassified by h, then for each misclassification we need to transmit a message that identifies which example is misclassified as well as its correct classification

The hypothesis h_{MDL} under the encoding C_1 and C_2 is just the one that minimizes the sum ofthese description lengths.

- MDL principle provides a way for trading off hypothesis complexity for the number oferrors committed by the hypothesis
- MDL provides a way to deal with the issue of overfitting thedata.
- Short imperfect hypothesis may be selected over a long perfect hypothesis.

NAÏVE BAYES OPTIMAL CLASSIFIER

Bayes optimal classifier is a probabilistic model that makes the probable prediction
for a new example.
P(A/B)= [P(B/A).P(A)]/ P(B) i.e.
P(y/X)=[P(X/y).P(y)] / P(X)
The naïve Bayes classifier is based on the assumption that the attribute values are conditionally independent given the target value.
For a dataset: X={x1,x2……xn}
Here x=feature vector/attributes and y=yes/no
P(y/x1x2….xn)=
[[P(x1/y).P(x2/y)……..P(xn/y)] *
P(y)] / P(x1).P(x2)…….P(xn) P(y)
$\prod_{i=1}^{n}$ P(xi/y) / P(x1).P(x2)…….P(xn) ....

P(y) $\prod_{i=1}^{n}$ P(xi/y) omitted the denominator(as they remain constant) in eqn1 bcz we are not concerned about calculation.

Example1 on naïve Bayes classifier

Find the probability(player will enjoy or not) to play tennis on 15th day where the conditions are: {outlook = sunny and temp=hot}

$$P(yes/sunny,hot) = P(y) \prod_{i=1}^{n} P(xi/y)$$

$= P(sunny/yes) * P(hot/yes) * P(yes)$

$=2/9 * 2/9 * 9/14$

$=0.031$

$$P(no/sunny,hot) = P(y) \prod_{i=1}^{n} P(xi/y)$$

$= P(sunny/no) * P(hot/no) * P(no)$

$=3/5 * 2/5 * 5/14$

$=0.08571$

Total=0.031+0.08571 = 0.27

P(yes)=0.031/0.27= 0.114

P(no)=0.08571/0.27= 0.317

Compare which probability is more, here no probability is more. Therefore player will not enjoy the sport.

S. No.	Outlook	Temperature	Humidity	Windy	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rainy	Mild	High	Weak	Yes
5	Rainy	Cool	Normal	Weak	Yes
6	Rainy	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rainy	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rainy	Mild	High	Strong	No

Example2 on naïve Baye th s classifier

Find the probability(player will enjoy or not) to play d

S. No.	Outlook	Temperature	Humidity	Windy	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rainy	Mild	High	Weak	Yes
5	Rainy	Cool	Normal	Weak	Yes
6	Rainy	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rainy	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rainy	Mild	High	Strong	No

tennis on 15
{outlook =

sunny,temp=cool,humidity=high,wind=strong

}P(yes/sunny,cool,high,strong)

=
P(sunny/yes)*P(cool/yes)*P(high/yes)*P(strong/yes)*p(yes)
=2/9 * 3/9 * 3/9 * 3/9 * 9/14
=0.0053

P(no/sunny,cool,high,strong)

=
P(sunny/no)*P(cool/no)*P(high/no)*P(strong/no)*p(no)
=1/5 * 4/5 * 3/5 * 5/14 * 3/5
=0.0206

Total=0.0053+0.0206 = 0.0259
P(yes)=0.0053/0.0259= 0.2046
P(no)=0.0206/0.0259= 0.7953
Compare which probability is more, here no probability is more. Therefore player will not enjoy the sport.

GIBS ALGORITHM

1. Chooses one hypothesis at random, according to P(h/D)
2. Use this to classify new instanc

e

$$E[error_{gibbs}] <= 2E[error_{BayesOptimal}]$$

BAYESIAN BELIEF NETWORKS (BN)

BAYESIAN BELIEF NETWORKS

- The naive Bayes classifier makes significant use of the assumption that the values of the attributes $a_1 \ldots a_n$ are conditionally independent given the target value v.
- This assumption dramatically reduces the complexity of learning the target function

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities

Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variables

BAYESIAN BELIEF NETWORKS (contd.,)

Notation

- Consider an arbitrary set of random variables $Y_1 \ldots Y_n$, where each variable Yi can take on the set of possible values $V(Y_i)$.
- The joint space of the set of variables Y to be the cross product $V(Y_1) \times V(Y_2) \times \ldots V(Y_n)$.
- In other words, each item in the joint space corresponds to one of the possible assignments of values to the tuple of variables $(Y_1 \ldots Y_n)$. The probability distribution over this joint' space is called the joint probability distribution.
- The joint probability distribution specifies the probability for each of the possible variable bindings for the tuple $(Y_1 \ldots Y_n)$.
- A Bayesian belief network describes the joint probability distribution for a set of variables.

Conditional Independence

Let X, Y, and Z be three discrete-valued random variables. X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given a value for Z, that is, if

$$(\forall x_i, y_j, z_k)\ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

Where,

$$x_i \in V(X),\ y_j \in V(Y),\ \text{and}\ z_k \in V(Z).$$

BAYESIAN BELIEF NETWORKS (contd.,)

The above expression is written in abbreviated form as

$$P(X \mid Y, Z) = P(X \mid Z)$$

Conditional independence can be extended to sets of variables. The set of variables $X_1 \dots X_l$ is conditionally independent of the set of variables $Y_1 \dots Y_m$ given the set of variables $Z_1 \dots Z_n$ if

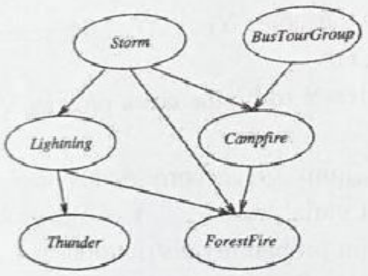
$$P(X_1 \dots X_l | Y_1 \dots Y_m, Z_1 \dots Z_n) = P(X_1 \dots X_l | Z_1 \dots Z_n)$$

The naive Bayes classifier assumes that the instance attribute A_1 is conditionally independent of instance attribute A_2 given the target value V. This allows the naive Bayes classifier to calculate $P(A_1, A_2 \mid V)$ as follows,

$$P(A_1, A_2|V) = P(A_1|A_2, V)P(A_2|V)$$
$$= P(A_1|V)P(A_2|V)$$

Representation

A Bayesian belief network represents the joint probability distribution for a set of variables. Bayesian networks (BN) are represented by directed acyclic graphs.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

The Bayesian network in above figure represents the joint probability distribution over the boolean variables *Storm, Lightning, Thunder, ForestFire, Campfire,* and *BusTourGroup*

BAYESIAN BELIEF NETWORKS (contd.,)

A Bayesian network (BN) represents the joint probability distribution by specifying a set of *conditional independence assumptions*

- BN represented by a directed acyclic graph, together with sets of local conditional probabilities
- Each variable in the joint space is represented by a node in the Bayesian network
- The network arcs represent the assertion that the variable is conditionally independent of its non-descendants in the network given its immediate predecessors in the network.
- A ***conditional probability table*** (**CPT**) is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors

Footer Page 89 of 120.

The joint probability for any desired assignment of values $(y_1, \ldots, y_n)$ to the tuple of network variables $(Y_1 \ldots Y_m)$ can be computed by the formula

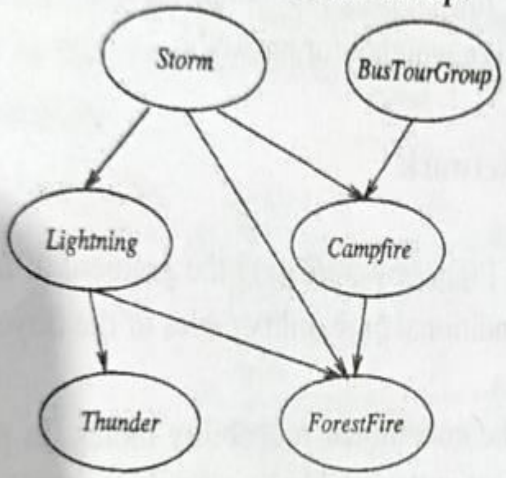
$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

Where, $Parents(Y_i)$ denotes the set of immediate predecessors of Y_i in the network.

BAYESIAN BELIEF NETWORKS (contd.,)

Example:

Consider the node ***Campfire***. The network nodes and arcs represent the assertion that ***Campfire*** is conditionally independent of its non-descendants ***Lightning*** and ***Thunder***, given its immediate parents Storm and ***BusTourGroup***.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

This means that once we know the value of the variables ***Storm*** and ***BusTourGroup***, the variables ***Lightning*** and ***Thunder*** provide no additional information about ***Campfire***

The conditional probability table associated with the variable ***Campfire***. The assertion is

$$P(\text{Campfire} = \text{True} \mid \text{Storm} = \text{True}, \text{BusTourGroup} = \text{True}) = 0.4$$

BAYESIAN BELIEF NETWORKS (contd.,)

Inference

- Use a Bayesian network to infer the value of some target variable (e.g., ForestFire) given the observed values of the other variables.
- Inference can be straightforward if values for all of the other variables in the network are known exactly.
- A Bayesian network can be used to compute the probability distribution for any subset of network variables given the values or distributions for any subset of the remaining variables.
- An arbitrary Bayesian network is known to be NP-hard

Learning Bayesian Belief Networks

Affective algorithms can be considered for learning Bayesian belief networks from training data by considering several different settings for learning problem

- First, the network structure might be given in advance, or it might have to be inferred from the training data.
- Second, all the network variables might be directly observable in each training example, or some might be unobservable.
  - In the case where the network structure is given in advance and the variables are fully observable in the training examples, learning the conditional probability tables is straightforward and estimate the conditional probability table entries
  - In the case where the network structure is given but only some of the variable values are observable in the training data, the learning problem is more difficult. The learning problem can be compared to learning weights for an ANN.

GRADIENT ASCENT
BAYESIAN NETWORKS

Gradient Ascent Training of Bayesian Network

The gradient ascent rule which maximizes P(D|h) by following the gradient of ***ln P(D|h)*** with respect to the parameters that define the conditional probability tables of the Bayesian network.

Let w_{ijk} denote a single entry in one of the conditional probability tables. In particular w_{ijk} denote the conditional probability that the network variable Y_i will take on the value y_{ij}, given that its immediate parents U_i take on the values given by u_{ik}.

The gradient of ***ln P(D|h)*** is given by the derivatives $\frac{\partial \ln P(D|h)}{\partial w_{ijk}}$ for each of the w_{ijk}.
As shown below, each of these derivatives can be calculated as

$$\frac{\partial \ln P(D|h)}{\partial w_{ij}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|d)}{w_{ijk}} \qquad \textbf{equ(1)}$$

Derive the gradient defined by the set of derivatives $\frac{\partial P_h(D)}{\partial w_{ijk}}$ for all ***i, j,*** and ***k***. Assuming the training examples ***d*** in the data set D are drawn independently, we write this derivative as

$$\begin{aligned}
\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \frac{\partial}{\partial w_{ijk}} \ln \prod_{d \in D} P_h(d) \\
&= \sum_{d \in D} \frac{\partial \ln P_h(d)}{\partial w_{ijk}} \\
&= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial P_h(d)}{\partial w_{ijk}}
\end{aligned}$$

GRADIENT ASCENT BAYESIAN NETWORKS (cont.,)

This last step makes use of the general equality $\frac{\partial \ln f(x)}{\partial x} = \frac{1}{f(x)} \frac{\partial f(x)}{\partial x}$. We can now introduce the values of the variables Y_i and $U_i = Parents(Y_i)$, by summing over their possible values $y_{ij'}$ and $u_{ik'}$.

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}, u_{ik'})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}|u_{ik'}) P_h(u_{ik'})$$

This last step follows from the product rule of probability . Now consider the rightmost sum in the final expression above. Given that $w_{ijk} \equiv P_h(y_{ij}|u_{ik})$, the only term in this sum for which $\frac{\partial}{\partial w_{ijk}}$ is nonzero is the term for which $j' = j$ and $i' = i$. Therefore

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) P_h(y_{ij}|u_{ik}) P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) w_{ijk} P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} P_h(d|y_{ij}, u_{ik}) P_h(u_{ik})$$

GRADIENT ASCENT BAYESIAN NETWORKS (cont.,)

Applying Bayes theorem to rewrite $P_h(d|y_{ij}, u_{ik})$, we have

$$
\begin{aligned}
\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{P_h(y_{ij}, u_{ik}|d) P_h(d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})} \\
&= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})} \\
&= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{P_h(y_{ij}|u_{ik})} \\
&= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}} \qquad \text{equ (2)}
\end{aligned}
$$

GRADIENT ASCENT BAYESIAN NETWORKS (cont.,)

Thus, we have derived the gradient given in Equation (1). There is one more item that must be considered before we can state the gradient ascent training procedure. In particular, we require that as the weights w_{ijk} are updated they must remain valid probabilities in the interval [0,1]. We also require that the sum $\sum_j w_{ijk}$ remains 1 for all i, k. These constraints can be satisfied by updating weights in a two-step process. First we update each w_{ijk} by gradient ascent

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

where η is a small constant called the learning rate. Second, we renormalize the weights w_{ijk} to assure that the above constraints are satisfied. this process will converge to a locally maximum likelihood hypothesis for the conditional probabilities in the Bayesian network.

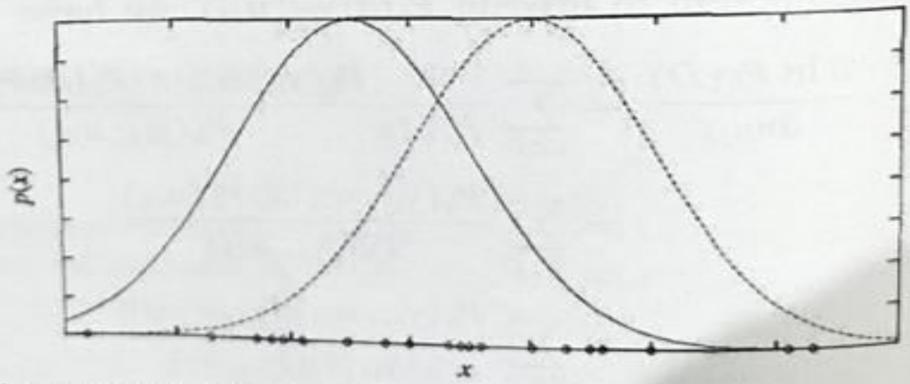
EM ALGORITHM

THE EM ALGORITHM

The EM algorithm can be used even for variables whose value is never directly observed, provided the general form of the probability distribution governing these variables is known.

Estimating Means of k Gaussians

- Consider a problem in which the data D is a set of instances generated by a probability distribution that is a mixture of k distinct Normal distributions.



- This problem setting is illustrated in Figure for the case where k = 2 and where the instances are the points shown along the x axis.
- Each instance is generated using a two-step process.
  - First, one of the k Normal distributions is selected at random.
  - Second, a single random instance x_i is generated according to this selected distribution.
- This process is repeated to generate a set of data points as shown in the figure.

EM ALGORITHM contd.,

- To simplify, consider the special case
  - The selection of the single Normal distribution at each step is based o each with uniform probability
  - Each of the k Normal distributions has the same variance σ^2, known
- The learning task is to output a hypothesis $h = (\mu_1, \ldots, \mu_k)$ that describes th each of the k distributions.
- We would like to find a maximum likelihood hypothesis for these means hypothesis *h* that maximizes p(D |h).

$$\mu_{ML} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{m} (x_i - \mu)^2 \qquad (1)$$

In this case, the sum of squared errors is minimized by the sample mean

$$\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} x_i \qquad (2)$$

- Our problem here, however, involves a mixture of k different Normal distrib we cannot observe which instances were generated by which distribution.
- Consider full description of each instance as the triple (x_i, z_{i1}, z_{i2}).
  - where x_i is the observed value of the ith instance and
  - where z_{i1} and z_{i2} indicate which of the two Normal distributions generate the value x_i
- In particular, z_{ij} has the value 1 if x_i was created by the j^{th} Normal distrib otherwise.
- Here x_i is the observed variable in the description of the instance, and z_{i} hidden variables.
- If the values of z_{i1} and z_{i2} were observed, we could use following Equation the means p_1 and p_2
- Because they are not, we will instead use the EM algorithm

EM ALGORITHM contd.,

EM algorithm

Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

Step 2: Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$ and iterate.

Let us examine how both of these steps can be implemented in practice. Step 1 must calculate the expected value of each z_{ij}. This $E[z_{ij}]$ is just the probability that instance x_i was generated by the jth Normal distribution

$$E[z_{ij}] = \frac{p(x = x_i|\mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i|\mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i-\mu_n)^2}}$$

Thus the first step is implemented by substituting the current values $\langle \mu_1, \mu_2 \rangle$ and the observed x_i into the above expression.

In the second step we use the $E[z_{ij}]$ calculated during Step 1 to derive a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$. maximum likelihood hypothesis in this case is given by

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}]\ x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

MODULE 5

INSTANCE BASED LEARNING

INTRODUCTION

- Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions.
- Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance
- Instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified

Advantages of Instance-based learning

1. Training is very fast
2. Learn complex target function
3. Don't lose information

Disadvantages of Instance-based learning

- The cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered.
- In many instance-based approaches, especially nearest-neighbor approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

k- NEAREST NEIGHBOR LEARNING

- The most basic instance-based method is the K- Nearest Neighbor Learning. This algorithm assumes all instances correspond to points in the n-dimensional space R^n.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- Let an arbitrary instance x be described by the feature vector

$$((a_1(x), a_2(x), \ldots\ldots\ldots, a_n(x))$$

Where, $a_r(x)$ denotes the value of the r^{th} attribute of instance x.

- Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$ Where,

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.

Let us first consider learning ***discrete-valued target functions*** of the form

$$f : \Re^n \rightarrow V.$$

Where, V is the finite set $\{v_1, \ldots v_s\}$

<u>The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is given below:</u>

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

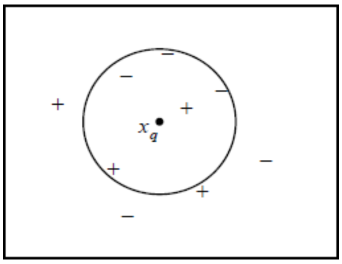
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

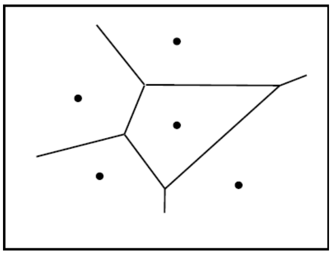
- The value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of f among the k training examples nearest to x_q.
- If k = 1, then the 1- Nearest Neighbor algorithm assigns to $\hat{f}(x_q)$ the value f(xi). Where x_i is the training instance nearest to x_q.
- For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

- Below figure illustrates the operation of the k-Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.



- The positive and negative training examples are shown by "+" and "-" respectively. A query point x_q is shown as well.
- The 1-Nearest Neighbor algorithm classifies x_q as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.

- Below figure shows the shape of this **decision surface** induced by 1- Nearest Neighbor over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples.



- For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the ***Voronoi diagram*** of the set of training example

The K- Nearest Neighbor algorithm for approximation a **real-valued target function** $f : \Re^n \rightarrow \Re$ is given below

---

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Distance-Weighted Nearest Neighbor Algorithm

- The refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point x_q, giving greater weight to closer neighbors.
- For example, in the k-Nearest Neighbor algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from x_q

Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions

Training algorithm:

- For each training example $\langle x, f(x)\rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Distance-Weighted Nearest Neighbor Algorithm for approximation a Real-valued target functions

---

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Terminology

- **Regression** means approximating a real-valued target function.
- **Residual** is the error $f(x)$ - f (x) in approximating the target function.
- **Kernel function** is the function of distance that is used to determine the weight of eachtraining example. In other words, the kernel function is the function K such that
  $w_i = K(d(x_i, x_q))$

LOCALLY WEIGHTED REGRESSION

- The phrase "**locally weighted regression**" is called **local** because the function is approximated based only on data near the query point, **weighted** because the contribution of each training example is weighted by its distance from the query point, and **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

- Given a new query instance x_q, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding x_q. This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.

Locally Weighted Linear Regression

- Consider locally weighted regression in which the target function ***f*** is

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

approximated near x_q using a linear function of the form

Where, $a_i(x)$ denotes the value of the i^{th} attribute of the instance x

- Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

Which led us to the gradient descent training rule

Where, η is a constant learning rate

- Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \, k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \qquad \textbf{equ(1)}$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \ K(d(x_q, x)) \qquad \textbf{equ(2)}$$

K of its distance from x_q :

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \ k \ nearest \ nbrs \ of \ x_q} (f(x) - \hat{f}(x))^2 \ K(d(x_q, x)) \qquad \textbf{equ(3)}$$

If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule

$$\Delta w_j = \eta \sum_{x \in \, k \ nearest \ nbrs \ of \ x_q} K(d(x_q, x)) \ (f(x) - \hat{f}(x)) \ a_j(x)$$

The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **$K(d(x_q, x))$,** and that the error is summed over only the k nearest training examples.

RADIAL BASIS FUNCTIONS

- One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions
- In this approach, the learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x)) \qquad \textbf{equ (1)}$$

- Where, each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance d(xu, x) increases.
- Here k is a user provided constant that specifies the number of kernel

functions to be included.

- $\hat{f}$ is a global approximation to f (x), the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u.

Choose each function $K_u(d(x_u, x))$ to be a Gaussian function centred at the point x_u with some variance σ_u^2

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

- The functional form of equ(1) can approximate any function with arbitrarily small error, provided a sufficiently large number k of such Gaussian kernels and provided the width σ^2 of each kernel can be separately specified
- The function given by equ(1) can be viewed as describing a two layer network where the first layer of units computes the values of the various $K_u(d(x_u, x))$ and where the second layer computes a linear combination of these first-layer unit values

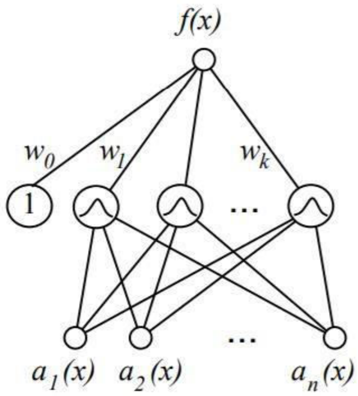
Example: Radial basis function (RBF) network

Given a set of training examples of the target function, RBF networks are typically trained in a two-stage process.

1. First, the number k of hidden units is determined and each hidden unit u is defined by choosing the values of x_u and σ_u^2 that define its kernel function $K_u(d(x_u, x))$
2. Second, the weights w, are trained to maximize the fit of the network to the training data, using the global error criterion given by

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

Because the kernel functions are held fixed during this second stage, the linear weight values w, can be trained very efficiently



Several alternative methods have been proposed for choosing an appropriate number of hidden units or, equivalently, kernel functions.

- One approach is to allocate a Gaussian kernel function for each training example $(x_i, f(x_i))$, centring this Gaussian at the point x_i.

  Each of these kernels may be assigned the same width σ^2. Given this approach, the RBF network learns a global approximation to the target function in which each training example $(x_i, f(x_i))$ can influence the value of ***f*** only in the neighbourhood of x_i.
- A second approach is to choose a set of kernel functions that is smaller than the number of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large.

Summary

- Radial basis function networks provide a global approximation to the target function, represented by a linear combination of many local kernel functions.
- The value for any given kernel function is non-negligible only when the input x falls into the region defined by its particular centre and width. Thus, the network can be viewed as a smooth linear combination of many local approximations to the target function.
- One key advantage to RBF networks is that they can be trained much more efficiently than feedforward networks trained with BACKPROPAGATION.

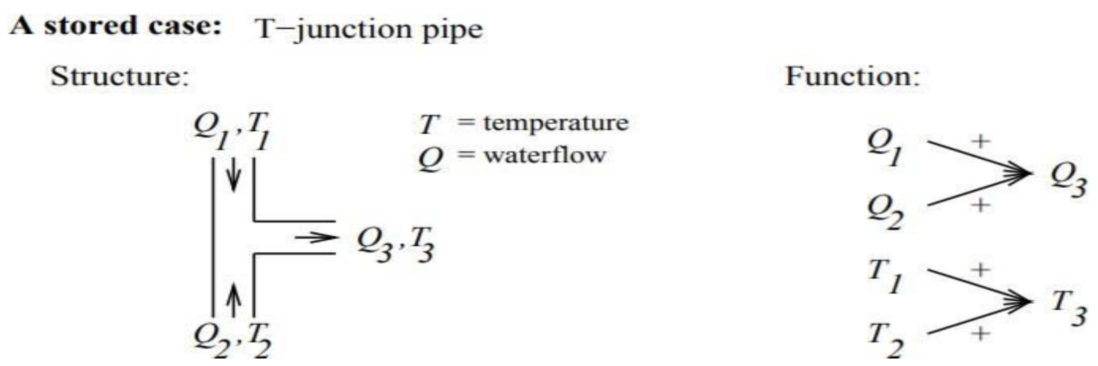
CASE-BASED REASONING

- Case-based reasoning (CBR) is a learning paradigm based on lazy learning methods and they classify new query instances by analysing similar instances while ignoring instances that are very different from the query.
- In CBR represent instances are not represented as real-valued points, but instead, they use a ***rich symbolic*** representation.
- CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems

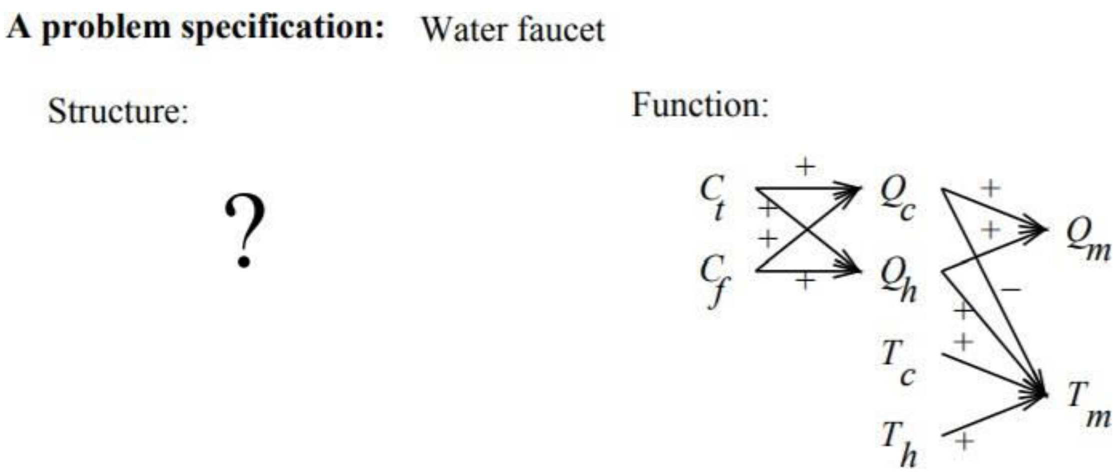
A prototypical example of a case-based reasoning

- The CADET system employs case-based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.
- It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems.
- Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.
- New design problems are then presented by specifying the desired function and requesting the corresponding structure.

The problem setting is illustrated in below figure



- The function is represented in terms of the qualitative relationships among the water- flow levels and temperatures at its inputs and outputs.
- In the functional description, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. A "-" label indicates that the variable atthe head decreases with the variable at the tail.
- Here Q_c refers to the flow of cold water into the faucet, Q_h to the input flow of hot water,and Q_m to the single mixed flow out of the faucet.
- T_c, T_h, and T_m refer to the temperatures of the cold water, hot water, and mixed water respectively.
- The variable C_t denotes the control signal for temperature that is input to the faucet, and C_f denotes the control signal for waterflow.
- The controls C_t and C_f are to influence the water flows Q_c and Q_h, thereby indirectly influencing the faucet output flow Q_m and temperature T_m.



- CADET searches its library for stored cases whose functional descriptions match the design problem. If an exact match is found, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem. If no exact match occurs, CADET may find cases that match various subgraphs of the desired functional specification.

REINFORCEMENT LEARNING

Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals.

INTRODUCTION

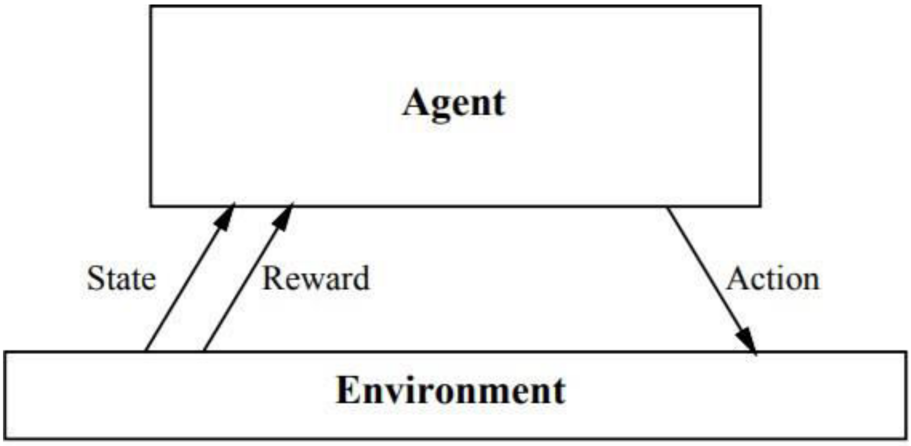
- Consider building a **learning robot**. The robot, or ***agent***, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.
- Its task is to learn a control strategy, or ***policy***, for choosing actions that achieve its goals.
- The goals of the agent can be defined by a ***reward function*** that assigns a numerical value to each distinct action the agent may take from each distinct state.
- This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot.
- The **task** of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.
- The control policy is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

Example:

- A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn."
- The robot may have a goal of docking onto its battery charger whenever its battery level is low.
- The goal of docking to the battery charger can be captured by assigning a positive reward (Eg., +100) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.

Reinforcement Learning Problem

- An agent interacting with its environment. The agent exists in an environment described by some set of possible states S.
- Agent perform any of a set of possible actions A. Each time it performs an action a, in some state s_t the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states s_i, actions a_i, and immediate rewards r_i as shown in the figure.
- The agent's task is to learn a control policy, $\boldsymbol{\pi}$**: S** → **A**, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \text{ , where } 0 \leq \gamma < 1$$

Reinforcement learning problem characteristics

1. **Delayed reward**: The task of the agent is to learn a target function π that maps from the current state s to the optimal action a = π (s). In reinforcement learning, training information is not available in (s, π (s)). Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of ***temporal credit assignment***: determining which of the actions in its sequence are to be credited with producing the eventual rewards.

2. **Exploration:** In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning. The learner faces a trade-off in choosing whether to favor exploration of unknown states and actions, or exploitation of states and actions that it has already learned will yield high reward.

3. **Partially observable states:** The agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information. In such cases, the agent needs to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.

4. **Life-long learning:** Robot requires to learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

THE LEARNING TASK

- Consider Markov decision process (MDP) where the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform.
- At each discrete time step t, the agent senses the current state s_t, chooses a current action a_t, and performs it.
- The environment responds by giving the agent a reward $r_t = r(s_t, a_t)$ and by producing the succeeding state $s_{t+l} = \delta(s_t, a_t)$. Here the functions $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.

The task of the agent is to learn a policy, π: **S** → **A**, for selecting its next action a, based on the current observed state s_t; that is, **$(s_t) = a_t$.**

How shall we specify precisely which policy π we would like the agent to learn?

1. One approach is to require the policy that produces the greatest possible ***cumulative reward*** for the robot over time.

- To state this requirement more precisely, define the cumulative value $V^{\pi}(s_t)$

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad \text{equ (1)}$$

achieved by following an arbitrary policy π from an arbitrary initial state s_t as follows:

- Where, the sequence of rewards r_{t+i} is generated by beginning at state s_t and by repeatedly using the policy π to select actions.
- Here $0 \leq \gamma \leq 1$ is a constant that determines the relative value of delayed versus immediate rewards. if we set $\gamma = 0$, only the immediate reward is considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward.
- The quantity $V^{\pi}(s_t)$ is called the ***discounted cumulative reward*** achieved by policy π from initial state ***s***. It is reasonable to discount future rewards relative to immediate rewards because, in many cases, we prefer to obtain the reward sooner rather than later.

2. Other definitions of total reward is ***finite horizon reward,***

$$\sum_{i=0}^{h} r_{t+i}$$

Considers the undiscounted sum of rewards over a finite number ***h*** of steps

3. Another approach is ***average reward***

$$\lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$$

Considers the average reward per time step over the entire lifetime of the agent.

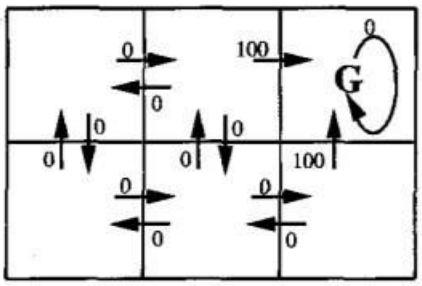
We require that the agent learn a policy п that maximizes V^{π} (s_t) for all states s. such a policy is called an ***optimal policy*** and denote it by п*

$$\pi^* \equiv \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s), (\forall s) \qquad \textbf{equ (2)}$$

Refer the value function V^{π}*(s) an optimal policy as V*(s). V*(s) gives the maximum discounted cumulative reward that the agent can obtain starting from state ***s***.

Example:

A simple grid-world environment is depicted in the diagram

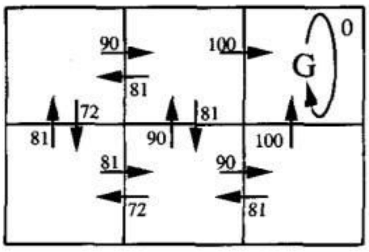


$r(s, a)$ (immediate reward) values

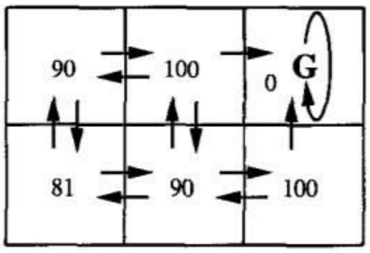
- The six grid squares in this diagram represent six possible states, or locations, for the agent.
- Each arrow in the diagram represents a possible action the agent can take to move fromone state to another.
- The number associated with each arrow represents the immediate reward ***r(s, a)*** the agent receives if it executes the corresponding state-action transition
- The immediate reward in this environment is defined to be zero for all state-action transitions except for those leading into the state labelled G. The state G as the goal state, and the agent can receive reward by entering this state.

Once the states, actions, and immediate rewards are defined, choose a value for the discount factor γ, determine the optimal policy π * and its value function V*(s).

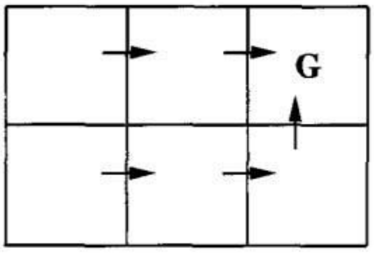
Let's choose γ = 0.9. The diagram at the



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

bottom of the figure shows one optimal policy for this setting.

Values of V*(s) and Q(s, a) follow from r(s, a), and the discount factor γ = 0.9. An optimal policy, corresponding to actions with maximal Q values, is also shown.

The discounted future reward from the bottom centre state is

$0+ \gamma\ 100+ \gamma^2\ 0+ \gamma^3\ 0+\ldots = 90$

Q LEARNING

How can an agent learn an optimal policy π * for an arbitrary environment?

The training information available to the learner is the sequence of immediate rewards $r(s_i,a_i)$ for i = 0, 1,2,Given this kind of training information it is easier to learn a numerical
evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.

What evaluation function should the agent attempt to learn?

One obvious choice is V*. The agent should prefer state s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$, because the cumulative future reward will be greater from s_1
The optimal action in state ***s*** is the action ***a*** that maximizes the sum of the immediate reward r(s, a) plus the value V* of the immediate successor state, discounted by γ.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}[r(s,a) + \gamma V^*(\delta(s,a))] \qquad \textbf{equ (3)}$$

The *Q* Function

The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state ***s***, plus the value (discounted by ***γ***) of following the optimal policy thereafter

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \qquad \text{equ (4)}$$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a) \qquad \text{equ (5)}$$

Rewrite Equation (3) in terms of ***Q(s, a)*** as Equation (5) makes clear, it need only consider each available action ***a*** in its current state ***s*** and choose the action that maximizes ***Q(s, a)***.

An Algorithm for Learning *Q*

- Learning the Q function corresponds to learning the **optimal policy**.
- The key problem is finding a reliable way to estimate training values for ***Q***, given only a sequence of immediate rewards ***r*** spread out over time. This can be accomplished through ***iterative approximation***

$$V^*(s) = \max_{a'} Q(s,a')$$

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

Rewriting Equation

- **Q learning algorithm:**

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

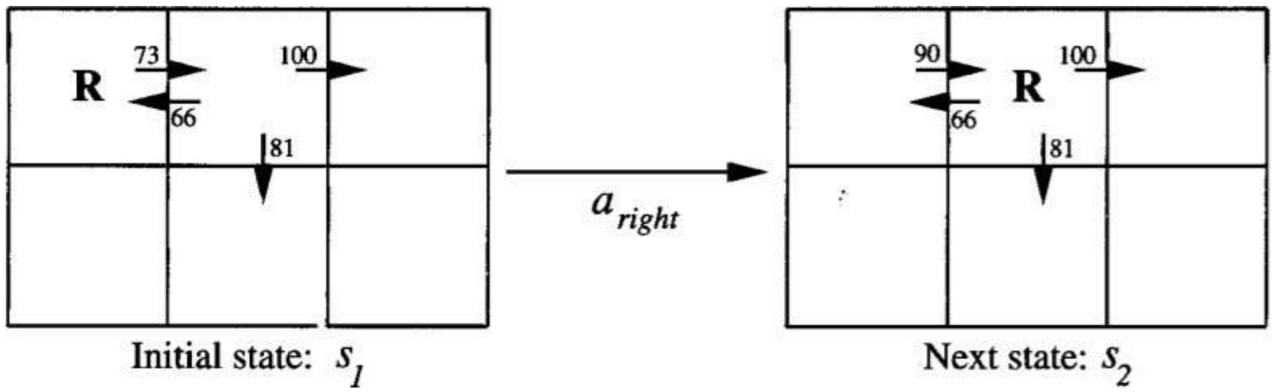
$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

- Q learning algorithm assuming deterministic rewards and actions. The discount factor γ may be any constant such that $0 \le \gamma < 1$
- $\hat{Q}$ to refer to the learner's estimate, or hypothesis, of the actual Q function

An Illustrative Example

- To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to $\hat{Q}$ shown in below figure



- The agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition.
- Apply the training rule of Equation

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

to refine its estimate Q for the state-action transition it just executed.

- According to the training rule, the new $\hat{Q}$ estimate for this transition is the sum of the received reward (zero) and the highest $\hat{Q}$ value associated with the resulting state (100), discounted by γ (.9).

$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \ \max\{66, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

Convergence

Will the Q Learning Algorithm converge toward a Q equal to the true Q function?

Yes, under certain conditions.

1. Assume the system is a deterministic MDP.
2. Assume the immediate reward values are bounded; that is, there exists some positiveconstant c such that for all states s and actions a, **| r(s, a)| < c**
3. Assume the agent selects actions in such a fashion that it visits every possible state-action pair infinitely often

Theorem Convergence of Q learning for deterministic Markov decision processes.

Consider a Q learning agent in a deterministic MDP with bounded rewards $(\forall s, a)|r(s,a)| \leq c$.

The Q learning agent uses the training rule of Equation $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$ initializes its table $\hat{Q}(s,a)$ to arbitrary finite values, and uses a discount factor γ such that $0 \leq \gamma < 1$. Let $\hat{Q}_n(s,a)$ denote the agent's hypothesis $\hat{Q}(s,a)$ following the nth update. If each state-action pair is visited infinitely often, then $\hat{Q}_n(s,a)$ converges to $Q(s,a)$ as $n \to \infty$, for all s, a.

Proof. Since each state-action transition occurs infinitely often, consider consecutive intervals during which each state-action transition occurs at least once. The proof consists of showing that the maximum error over all entries in the $\hat{Q}$ table is reduced by at least a factor of γ during each such interval. $\hat{Q}_n$ is the agent's table of estimated Q values after n updates. Let Δ_n be the maximum error in $\hat{Q}_n$; that is

$$\Delta_n \equiv \max_{s,a} |\hat{Q}_n(s,a) - Q(s,a)|$$

Below we use s' to denote $\delta(s,a)$. Now for any table entry $\hat{Q}_n(s,a)$ that is updated on iteration $n+1$, the magnitude of the error in the revised estimate $\hat{Q}_{n+1}(s,a)$ is

$$\begin{aligned}
|\hat{Q}_{n+1}(s,a) - Q(s,a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s',a')) - (r + \gamma \max_{a'} Q(s',a'))| \\
&= \gamma |\max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q(s',a')| \\
&\leq \gamma \max_{a'} |\hat{Q}_n(s',a') - Q(s',a')| \\
&\leq \gamma \max_{s'',a'} |\hat{Q}_n(s'',a') - Q(s'',a')| \\
|\hat{Q}_{n+1}(s,a) - Q(s,a)| &\leq \gamma \Delta_n
\end{aligned}$$

The third line above follows from the second line because for any two functions f_1 and f_2 the following inequality holds

$$|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$$

In going from the third line to the fourth line above, note we introduce a new variable s'' over which the maximization is performed. This is legitimate because the maximum value will be at least as great when we allow this additional variable to vary. Note that by introducing this variable we obtain an expression that matches the definition of Δ_n.

Thus, the updated $Q_{n+1}(s, a)$ for any s, a is at most γ times the maximum error in the $\hat{Q}_n$ table, Δ_n. The largest error in the initial table, Δ_0, is bounded because values of $\hat{Q}_0(s, a)$ and $Q(s, a)$ are bounded for all s, a. Now after the first interval during which each s, a is visited, the largest error in the table will be at most $\gamma \Delta_0$. After k such intervals, the error will be at most $\gamma^k \Delta_0$. Since each state is visited infinitely often, the number of such intervals is infinite, and $\Delta_n \rightarrow 0$ as $n \rightarrow \infty$. This proves the theorem.

Experimentation Strategies

The Q learning algorithm does not specify how actions are chosen by the agent.

- One obvious strategy would be for the agent in state ***s*** to select the action ***a*** thatmaximizes $\hat{Q}(s, a)$, thereby exploiting its current approximation $\hat{Q}$
- However, with this strategy the agent runs the risk that it will overcommit to actionsthat are found during early training to have high Q values, while failing to explore other actions that have even higher values.
- For this reason, Q learning uses a probabilistic approach to selecting actions. Actionswith higher $\hat{}$ values are assigned higher probabilities, but every action is assigned a nonzero probability.
- One way to assign such probabilities is

$$P(a_i|s) = \frac{k^{\hat{Q}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$

Where, **$P(a_i | s)$** is the probability of selecting action **a_i**, given that the agent is in state **s**, and **k > 0** is a constant that determines how strongly the selection favors actions with high $\hat{Q}$ values

MODULE 5

EVALUATING HYPOTHESES

MOTIVATION

It is important to evaluate the performance of learned hypotheses as precisely as possible.

- One reason is simply to understand whether to use the hypothesis.
- A second reason is that evaluating hypotheses is an integral component of many learning methods.

Two key difficulties arise while learning a hypothesis and estimating its future accuracy given only a limited set of data:

1. **Bias in the estimate**. The observed accuracy of the learned hypothesis over the training examples is often a poor estimator of its accuracy over future examples. Because the learned hypothesis was derived from these examples, they will typically provide an optimistically biased estimate of hypothesis accuracy over future examples. This is especially likely when the learner considers a very rich hypothesis space, enabling it to overfit the training examples. To obtain an unbiased estimate of future accuracy, test the hypothesis on some set of test examples chosen independently of the training examples and the hypothesis.

2. **Variance in the estimate.** Even if the hypothesis accuracy is measured over an unbiased set of test examples independent of the training examples, the measured accuracy can still vary from the true accuracy, depending on the

makeup of the particular set of test examples. The smaller the set of test examples, the greater the expected variance.

ESTIMATING HYPOTHESIS ACCURACY

Sample Error –

The sample error of a hypothesis with respect to some sample S of instances drawn from X is the fraction of S that it misclassifies.

Definition: The sample error (***$error_s(h)$***) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where n is the number of examples in S, and the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

True Error –

The true error of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution D.

Definition: The true error ($error_D(h)$) of

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

hypothesis h with respect to target function f and distribution D, is the probability that h will misclassify an instance drawn at random according to D.

Confidence Intervals for Discrete-Valued Hypotheses

Suppose we wish to estimate the true error for some discrete valued hypothesis h, based on its observed sample error over a sample S, where

- The sample S contains n examples drawn independent of one another, and independent of h, according to the probability distribution D
- $n \geq 30$
- Hypothesis h commits r errors over these n examples (i.e., $error_S(h) = r/n$).

Under these conditions, statistical theory allows to make the following assertions:

1. Given no other information, the most probable value of $error_D(h)$ is $error_S(h)$
2. With approximately **95% probability**, the true error $error_D(h)$ lies in the interval

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Example:

Suppose the data sample S contains n = 40 examples and that hypothesis h commits r = 12 errors over this data.

- The ***sample error*** is $error_S(h) = r/n = 12/40 = 0.30$
- Given no other information, ***true error*** is $error_D(h) = error_S(h)$, i.e., $error_D(h) = 0.30$
- With the 95% confidence interval estimate for $error_D(h)$.

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

$$= 0.30 \pm (1.96 * 0.07) = 0.30 \pm 0.14$$

3. A different constant, ***ZN***, is used to calculate the **N% confidence interval**. The general expression for approximate N% confidence intervals for $error_D(h)$

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

is

Where,

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

The above equation describes how to calculate the confidence intervals, or error bars, for estimates of $error_D(h)$ that are based on $error_S(h)$

Example:

Suppose the data sample S contains n = 40 examples and that hypothesis h commits r = 12 errors over this data.

- The ***sample error*** is $error_S(h) = r/n = 12/40 = 0.30$
- With the 68% confidence interval estimate for $error_D(h)$.

$$error_S(h) \pm 1.00 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

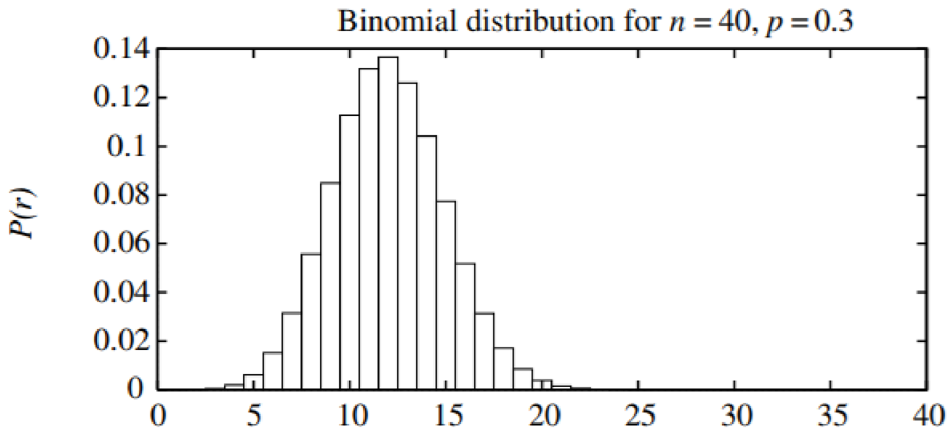
$$= 0.30 \pm (1.00 * 0.07)$$
$$= 0.30 \pm 0.07$$

BASICS OF SAMPLING THEORY

Error Estimation and Estimating Binomial Proportions

- Collect a random sample S of *n* independently drawn instances from the distribution *D*, and then measure the sample error $error_s(h)$. Repeat this experiment many times, each time drawing a different random sample S_i of size *n*, we would expect to observe different values for the various $error_{si}(h)$, depending on random differences in the makeup of the various S_i. We say that $error_{si}(h)$, the outcome of the i^{th} such experiment, is a ***random variable***.

- Imagine that we were to run k random experiments, measuring the random variables $error_{s1}(h)$, $error_{s2}(h)$ . . . $error_{sk}(h)$ and plotted a histogram displaying the frequency with which each possible error value is observed.
- As ***k*** grows, the histogram would approach a particular probability distribution called the ***Binomial distribution*** which is shown in below figure.



$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

A Binomial distribution is defined by the probability function

If the random variable *X* follows a Binomial distribution, then:

- The probability ***Pr(X = r)*** that ***X*** will take on the value ***r*** is given by ***P(r)***

- Expected, or mean value of X, $E[X]$, is

$$E[X] \equiv \sum_{i=0}^{n} iP(i) = np$$

- Variance of X is

$$Var(X) \equiv E[(X - E[X])^2] = np(1-p)$$

- Standard deviation of X, σ_X, is

$$\sigma_X \equiv \sqrt{E[(X - E[X])^2]} = \sqrt{np(1-p)}$$

The Binomial Distribution

Consider the following problem for better understanding of Binomial Distribution

- Given a worn and bent coin and estimate the probability that the coin will turn up heads when tossed.
- Unknown probability of heads ***p***. Toss the coin ***n*** times and record the number of times

  r that it turns up heads.

  Estimate of ***p = r / n***
- If the experiment were *rerun*, generating a new set of ***n*** coin tosses, we might expect the number of heads ***r*** to vary somewhat from the value measured in the first experiment, yielding a somewhat different estimate for ***p***.
- The Binomial distribution describes for each possible value of ***r*** (i.e., from 0 to n), the probability of observing exactly ***r*** heads given a sample of ***n*** independent tosses of a coin whose true probability of heads is ***p***.

<u>The general setting to which the Binomial distribution applies is:</u>

1. There is a base experiment (e.g., toss of the coin) whose outcome can be described by a random variable 'Y'. The random variable Y can take on two possible values (e.g., Y = 1 if heads, Y = 0 if tails).
2. The probability that Y = 1 on any single trial of the base experiment is given by some constant p, independent of the outcome of any other experiment. The probability that Y

   = 0 is therefore (1 - p). Typically, p is not known in advance, and the problem is to estimate it.
3. A series of n independent trials of the underlying experiment is performed (e.g., n independent coin tosses), producing the sequence of independent, identically distributed random variables $Y_1, Y_2, \ldots, Y_n$. Let R denote the

$$R \equiv \sum_{i=1}^{n} Y_i$$

number of trials for which $Y_i = 1$ in this series of n experiments

4. The probability that the random variable R will take on a specific value r (e.g., the probability of observing exactly r heads) is given by the Binomial distribution

$$\Pr(R = r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r} \qquad \textbf{equ (1)}$$

Mean, Variance and Standard Deviation

The Mean (expected value) is the average of the values taken on by repeatedly sampling the random variable

Definition: Consider a random variable Y

$$E[Y] \equiv \sum_{i=1}^{n} y_i \Pr(Y = y_i)$$

that takes on the possible values $y_1, \ldots y_n$. The expected value (Mean) of Y, E[Y], is

The Variance captures how far the random variable is expected to vary from its mean value.

Definition: The variance of a random variable Y, Var[Y], is

$$Var[Y] \equiv E[(Y - E[Y])^2]$$

The variance describes the expected squared error in using a single observation of Y to estimateits mean E[Y].
The square root of the variance is called the standard deviation of Y, denoted σ_Y

$$\sigma_Y \equiv \sqrt{E[(Y - E[Y])^2]}$$

Definition: The standard deviation of a random variable Y, σ_y, is

In case the ***random variable Y is governed by a Binomial distribution***, then the Mean, Varianceand standard deviation are given by

$$E[Y] = np$$
$$Var[Y] = np(1 - p)$$
$$\sigma_Y = \sqrt{np(1 - p)}$$

Estimators, Bias, and Variance

Let us describe $error_S(h)$ and $error_D(h)$

$$error_S(h) = \frac{r}{n}$$

$$error_{\mathcal{D}}(h) = p$$

using the terms in Equation (1) defining the Binomial distribution. We then have

Where,

- n is the number of instances in the sample S,
- r is the number of instances from S misclassified by h
- p is the probability of misclassifying a single instance drawn from *D*

- Estimator:

  $error_S(h)$ an ***estimator*** for the true error $error_D(h)$: An estimator is any random variable used to estimate some parameter of the underlying population from which the sample is drawn

- Estimation bias: is the difference between the expected value of the estimator and the truevalue of the parameter.

Definition: The estimation bias of an estimator Y for an arbitrary parameter p is

$$E[Y] - p$$