

Program: Master of Computer Applications

Curriculum Scheme: MCA 2 year Course

Examination: MCA First Year Semester - II

Course Code: MCA22 and Course Name: Artificial Intelligence and Machine Learning

Time: 2 HRS

Max. Marks: 80

Section I - MCQS (40 Marks) – 40 Minutes

Section II – Subjective (40 Marks) – 80 Minutes

=====

SECTION II

Q1. Solve any **four** questions out of **six** which carry 5 marks each respectively.

20 marks

1. Write a short note on Alpha beta search algorithm
2. Explain PEAS representation with example
3. What is the significance of ensemble methods? Discuss any three ensemble methods.
4. Explain Principal Component Analysis.
5. Write a note on Support Vector Machine.
6. Explain Hill Climbing algorithm

Q2 Solve any **two** questions out of **three** which carry 10 marks each respectively. -

20 marks

1. Implement OR function with binary inputs and bipolar targets using perceptron training algorithm for 1 epoch.

The initial values of the weights and bias are taken as zero. Also the learning rate is 1 and the threshold is 0.2. Activation function is

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

2. Explain Expectation-Maximization algorithm with an example.
3. Explain the K-nearest neighbor algorithm with an example.

Program: Master of Computer Applications

Curriculum Scheme: MCA 2 year Course

Examination: MCA First Year SEMESTER 2

Course Code: MCA22 and Course Name: Artificial Intelligence and Machine Learning

Time: 2 HRS

Max. Marks: 80

Section I - MCQS (40 Marks) – 40 Minutes

Section II – Subjective (40 Marks) – 80 Minutes

The timings

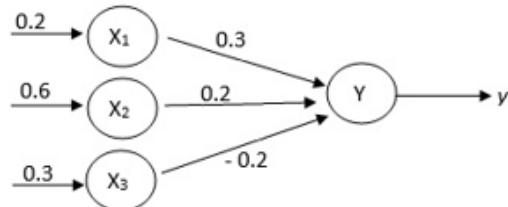
Section I – 2:00 pm – 2:40 pm

Section II – 2:40 pm – 4:00 pm

SECTION II

Q2. Solve any two questions out of three which carry 10 marks each respectively. 20 marks

- A. Describe the environment of the Agent? Discuss the types of Agents.
- B. Explain in detail the working of Depth First Search technique.
- C. Discuss in detail the Perceptron Networks and calculate the net input (y_{in}) of the given network.



Q3. Solve any two questions out of three which carry 10 marks each respectively. 20 marks

- A. Explain in detail Linear Regression with suitable example.
- B. Discuss in detail Support Vector Machine.
- C. Apply the Naive Bayes Classifier on a given Training Dataset and identify the given data (X) belongs to the class Buys Computer = Yes or Buys Computer = No.

Data to be classified:

$X = (\text{Age} \leq 20, \text{Income} = \text{Max}, \text{Student} = \text{Yes}, \text{Credit Rating} = \text{Very Good})$

Training Dataset:

Age	Income	Student	Credit Rating	Buys Computer
<=20	Max	No	Fair	No
<=20	Max	No	Very Good	No
21 to 25	Max	No	Fair	Yes
>25	Medium	No	Fair	Yes
>25	Min	Yes	Fair	Yes
>25	Min	Yes	Very Good	No
26 to 35	Min	Yes	Very Good	Yes
<=25	Medium	No	Fair	No
<=25	Min	Yes	Fair	Yes
>35	Medium	Yes	Fair	Yes
<=25	Medium	Yes	Very Good	Yes
26 to 35	Medium	No	Very Good	Yes
26 to 35	Max	Yes	Fair	Yes
>35	Medium	No	Very Good	No

Program: Master of Computer Applications

Curriculum Scheme: MCA 2 year Course

Examination: MCA First Year Semester - II

Course Code: MCA22 and Course Name: Artificial Intelligence and Machine Learning

Time: 2 HRS

Max. Marks: 80

Section I - MCQS (40 Marks) – 40 Minutes

Section II – Subjective (40 Marks) – 80 Minutes

SECTION II

Q2. Solve any **four** questions out of **six** which carry 5 marks each respectively.

20 marks

- A. What is entropy and what is its significance?
- B. What are Steps for Designing Learning System.?
- C. Write a short note on KNN Classification. .
- D. Give a PEAS representation for an intelligent Robot for cleaning a library..
- E. Write a note on Support Vector Machine.
- F. Explain any method of reducing dimensionality?

Q3 Solve any **two** questions out of **three** which carry 10 marks each respectively. - **20 marks**

- 1. Explain Hill Climbing Searching Technique with its advantages and limitations.
- 2. Explain the K-Means Clustering algorithm with an example.
- 3. Implement AND function using perceptron network for bipolar inputs and bipolar targets.(Initial values are $w_1=w_2=w_3=b=0$, learning rate=1, threshold=0.2)

Introduction to Artificial Intelligence

Kalev Kask

ICS 271

Fall 2018

<http://www.ics.uci.edu/~kkask/Fall-2018%20CS271/>

Course requirements

Assignments:

- There will be weekly homework assignments, a project, a final (12/13 4-6pm).

Course-Grade:

- Homework will account for 20% of the grade, project 30%, final 50% of the grade.

Discussion:

- Optional. Wed. 9-9:50 and 10-10:50 PCB 1200.

MS CE :

- CS271 can be used to satisfy the requirement

Discussion/Submissions

For course material discussion and questions, we use piazza:

- <http://piazza.com/uci/fall2018/cs271>

For homework/project submission, we use gradescope

- <https://www.gradescope.com/courses/27083>
- When register, can use entry code ME7Z2N

Plan of the course

Textbook : <http://aima.cs.berkeley.edu/>

Part I Artificial Intelligence

- 1 Introduction
- 2 Intelligent Agents

Part II Problem Solving

- 3 Solving Problems by Searching
- 4 Beyond Classical Search
- 5 Adversarial Search
- 6 Constraint Satisfaction Problems

Part III Knowledge and Reasoning

- 7 Logical Agents
- 8 First-Order Logic
- 9 Inference in First-Order Logic
- 10 Classical Planning

Resources on the internet

Resources on the Internet

- [AI on the Web](#): A very comprehensive list of Web resources about AI from the Russell and Norvig textbook.

Essays and Papers

- [What is AI](#), John McCarthy
- Computing Machinery and Intelligence, A.M. Turing
- [Rethinking Artificial Intelligence](#), Patrick H.Winston
- [AI Topics: http://aitopics.net/index.php](#)

Today's class

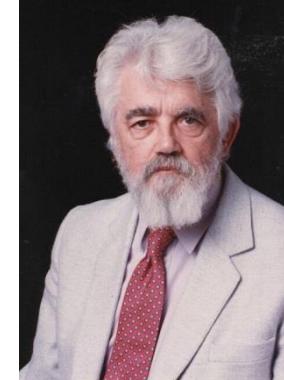
- What is Artificial Intelligence?
- A brief History
- State of the art
- Intelligent agents

Today's class

- What is Artificial Intelligence?

What is Artificial Intelligence

([John McCarthy](#), Basic Questions)



- **What is artificial intelligence?**
- It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.
- **Yes, but what is intelligence?**
- Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.
- **Isn't there a solid definition of intelligence that doesn't depend on relating it to human intelligence?**
- Not yet. The problem is that we cannot yet characterize in general what kinds of computational procedures we want to call intelligent. We understand some of the mechanisms of intelligence and not others.
- More in: <http://www-formal.stanford.edu/jmc/whatisai/node1.html>

What is Artificial Intelligence?

- Human-like vs rational-like
- Thought processes vs behavior
- How to simulate human intellect and behavior by a machine.
 - Mathematical problems (puzzles, games, theorems)
 - Common-sense reasoning
 - Expert knowledge: lawyers, medicine, diagnosis
 - Social behavior
- **Things we would call “intelligent” if done by a human.**

The Turing Test

([Can Machine think? A. M. Turing, 1950](#))



<http://aitopics.net/index.php>

http://amturing.acm.org/acm_tcc_webcasts.cfm

- Requires:
 - Natural language
 - Knowledge representation
 - Automated reasoning
 - Machine learning
 - (vision, robotics) for full test

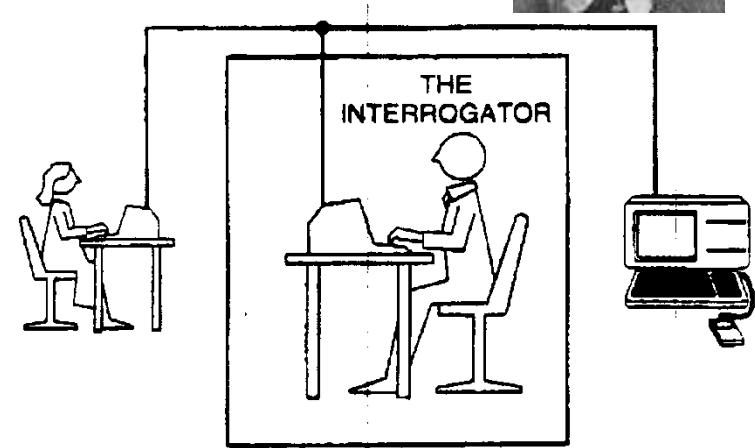


Figure 1.1 The Turing test.

What is Artificial Intelligence?

Views of AI fall into four categories:

Thinking humanly	Thinking rationally
Acting humanly	Acting rationally

The textbook advocates "acting rationally"

How to simulate humans intellect and behavior by a machine.
Mathematical problems (puzzles, games, theorems)
Common-sense reasoning
Expert knowledge: lawyers, medicine, diagnosis
Social behavior

Today's class

- What is Artificial Intelligence?
- A brief history
- State of the art
- Intelligent agents

The foundation of AI

Philosophy, Mathematics, Economics, Neuroscience, Psychology,
Computer Engineering

Features of intelligent system

- Deduction, reasoning, problem solving
- Knowledge representation
- Planning
- Learning
- Natural language processing
- Perception
- Motion and manipulation

Tools

- Search and optimization
- Logic
- Probabilistic reasoning
- Neural networks

The Birthplace of “Artificial Intelligence”, 1956

- **Darmouth workshop, 1956:** historical meeting of the precieved founders of AI met: John McCarthy, Marvin Minsky, Alan Newell, and Herbert Simon.
- A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. J. McCarthy, M. L. Minsky, N. Rochester, and C.E. Shannon. August 31, 1955. "We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." *And this marks the debut of the term "artificial intelligence."*"
- **50 anniversery of Darmouth workshop**
 - [List of AI-topics](#)

History of AI

- McCulloch and Pitts (1943)
 - Neural networks that learn
- Minsky and Edmonds (1951)
 - Built a neural net computer
- Dartmouth conference (1956):
 - McCarthy, Minsky, Newell, Simon met,
 - Logic theorist (LT)- Of Newell and Simon proves a theorem in Principia Mathematica-Russel.
 - The name “Artificial Intelligence” was coined.
- 1952-1969 (early enthusiasm, great expectations)
 - GPS- Newell and Simon
 - Geometry theorem prover - Gelernter (1959)
 - Samuel Checkers that learns (1952)
 - McCarthy - Lisp (1958), Advice Taker, Robinson's resolution
 - Microworlds: Integration, block-worlds.
 - 1962- the perceptron convergence (Rosenblatt)

More AI examples

Common sense reasoning (1980-1990)

- Tweety
- Yale Shooting problem

Update vs revise knowledge

The OR gate example: A or B → C

- Observe C=0, vs Do C=0

Chaining theories of actions

Looks-like(P) → is(P)

Make-looks-like(P) → Looks-like(P)

Makes-looks-like(P) → is(P) ???

Garage-door example: garage door not included.

- Planning benchmarks
- 8-puzzle, 8-queen, block world, grid-space world
- Cambridge parking example

Smoked fish example... what is this?

History, continued

- 1966-1974 a dose of reality
 - Problems with computation
- 1969-1979 Knowledge-based systems
 - Weak vs. strong methods
 - Expert systems:
 - Dendral : Inferring molecular structures (Buchanan et. Al. 1969)
 - Mycin : diagnosing blood infections (Shortliffe et. Al, certainty factors)
 - Prospector : recommending exploratory drilling (Duda).
 - Roger Shank: no syntax only semantics
- 1980-1988: AI becomes an industry
 - R1: McDermott, 1982, order configurations of computer systems
 - 1981: Fifth generation
- 1986-present: return to neural networks
- 1987-present :
 - **AI becomes a science:** HMMs, planning, belief network
- 1995-present: The emergence of intelligent agents
 - Ai agents (SOAR, Newell, Laird, 1987) on the internet, technology in web-based **applications**, recommender systems. Some researchers (Nilsson, McCarthy, Minsky, Winston) express discontent with the progress of the field. AI should return to human-level AI (they say).
- 2001-present: The availability of data;
 - The knowledge bottleneck may be solved for many applications: learn the information rather than hand code it
 - Big Data (e.g. social media, sensors, DBs, etc.)
 - Massive (parallel) computing power – (e.g. Deep Learning/Neural Nets)

State of the art

- **Game Playing:** Deep Blue defeated the reigning world chess champion Garry Kasparov in 1997; AlphaGo 2018 beats GO world champion.
- **Robotics vehicles:**
 - 2005 Standford robot won DARPA Grand Challenge, driving autonomously 131 miles along unrehearsed desert trail
 - Staneley (Thrun 2006). No hands across America (driving autonomously 98% of the time from Pittsburgh to San Diego)
 - 2007 CMU team won DARPA Urban Challenge driving autonomously 55 miles in a city while adhering to traffic laws and hazards
 - Self-driving cars (Google, Uber, Tesla, etc.)
- **Autonomous planning and scheduling:**
 - During the 1991 Gulf War, US forces deployed an AI logistics planning and scheduling program that involved up to 50,000 vehicles, cargo, and people
 - NASA's on-board autonomous planning program controlled the scheduling of operations for a spacecraft
- **Speech recognition (e.g. Siri, ...)**
- DARPA grand challenge 2003-2005, Robocup
- **Machine translation** (From English to Arabic, 2007)
- **Natural language processing:** Watson won Jeopardy (Natural language processing), IBM 2011.
- **Neural Nets + Deep Learning** – 100+B parameters, 100+M nodes, 100+ layers

Current “Hot” areas/applications

- Big Data
- with Machine Learning
- Deep Learning/Neural nets
- Transportation/robotics
- Vision
- Internet/social media

Today's class

- What is Artificial Intelligence?
- A brief History
- State of the art
- Intelligent agents

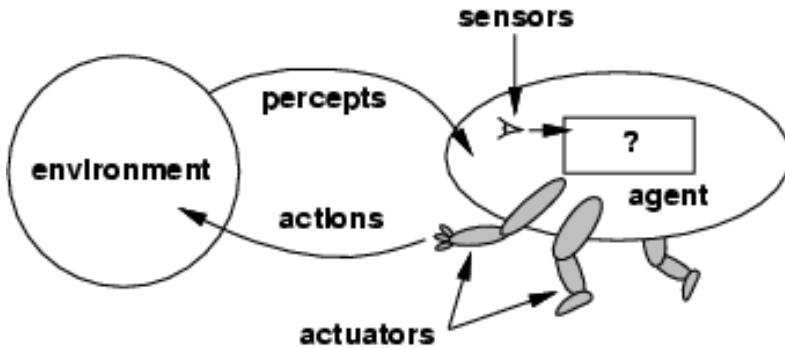
Agents

- An **agent** is anything that can be viewed as **perceiving** its **environment** through **sensors** and **acting** upon that environment through **actuators**
- Human agent: eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators
- Robotic agent: cameras and infrared range finders for sensors; various motors for actuators

Agents

- Agents and environments
- Rationality
- PEAS (Performance measure, Environment, Actuators, Sensors)
- Environment types
- Agent types

Agents and environments



- The **agent function** maps from percept histories to actions:
 $[f: \mathcal{P}^\star \rightarrow \mathcal{A}]$
- The **agent program** runs on the physical **architecture** to produce f
- agent = architecture + program

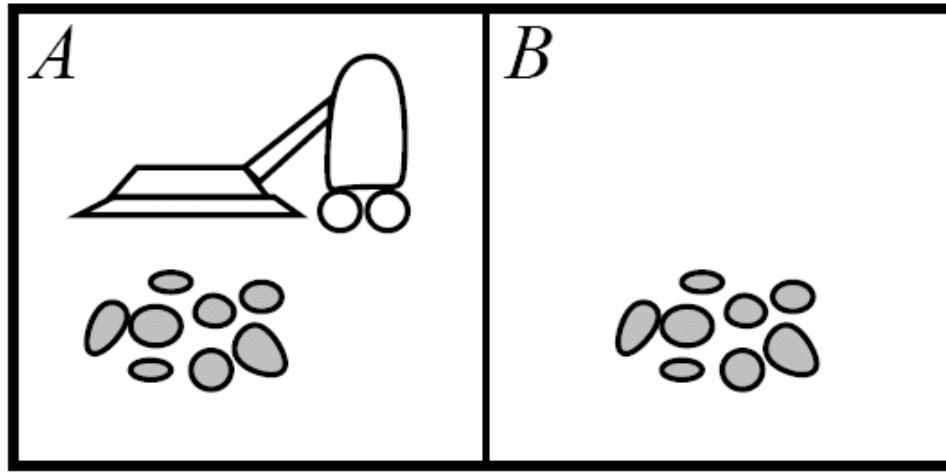
What's involved in Intelligence?

- **Ability to interact with the real world**
 - to perceive, understand, and act
 - e.g., speech recognition and understanding and synthesis
 - e.g., image understanding
 - e.g., ability to take actions, have an effect
- **Knowledge Representation, Reasoning and Planning**
 - modeling the external world, given input
 - solving new problems, planning and making decisions
 - ability to deal with unexpected problems, uncertainties
- **Learning and Adaptation**
 - we are continuously learning and adapting
 - our internal models are always being “updated”
 - e.g. a baby learning to categorize and recognize animals

Implementing agents

- **Table look-ups, Model-based, Goal-oriented, Utility, Learning**
- **Autonomy**
 - All actions are completely specified
 - no need in sensing, no autonomy
 - example: Monkey and the banana
- **Structure of an agent**
 - agent = architecture + program
 - Agent examples
 - medical diagnosis
 - Satellite image analysis system
 - part-picking robot
 - Interactive English tutor
 - cooking agent
 - taxi driver

Vacuum-cleaner world



Percepts: location and contents, e.g., [A, Dirty]

Actions: *Left*, *Right*, *Suck*, *NoOp*

A vacuum-cleaner agent

Percept sequence	Action
$[A, Clean]$	<i>Right</i>
$[A, Dirty]$	<i>Suck</i>
$[B, Clean]$	<i>Left</i>
$[B, Dirty]$	<i>Suck</i>
$[A, Clean], [A, Clean]$	<i>Right</i>
$[A, Clean], [A, Dirty]$	<i>Suck</i>
:	:

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

What is the *right* function?

Can it be implemented in a small agent program?

Rationality

Fixed **performance measure** evaluates the **environment sequence**

- one point per square cleaned up in time T ?
- one point per clean square per time step, minus one per move?
- penalize for $> k$ dirty squares?

A **rational agent** chooses whichever action maximizes the **expected** value of the performance measure **given the percept sequence to date**

Rational \neq omniscient

Rational \neq clairvoyant

Rational \neq successful

Rational \Rightarrow exploration, learning, autonomy

Task Environment

- Before we design a rational agent, we must specify its **task environment**:

PEAS:

Performance measure

Environment

Actuators

Sensors

PEAS

- Example: Agent = taxi driver
 - **Performance measure:** Safe, fast, legal, comfortable trip, maximize profits
 - **Environment:** Roads, other traffic, pedestrians, customers
 - **Actuators:** Steering wheel, accelerator, brake, signal, horn
 - **Sensors:** Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

PEAS

- Example: Agent = Medical diagnosis system
 - **Performance measure:** Healthy patient, minimize costs, lawsuits
 - **Environment:** Patient, hospital, staff
 - **Actuators:** Screen display (questions, tests, diagnoses, treatments, referrals)
 - **Sensors:** Keyboard (entry of symptoms, findings, patient's answers)

PEAS

- Example: Agent = part-picking robot
 - Performance measure: Percentage of parts in correct bins
 - Environment: Conveyor belt with parts, bins
 - Actuators: Jointed arm and hand
 - Sensors: Camera, joint angle sensors

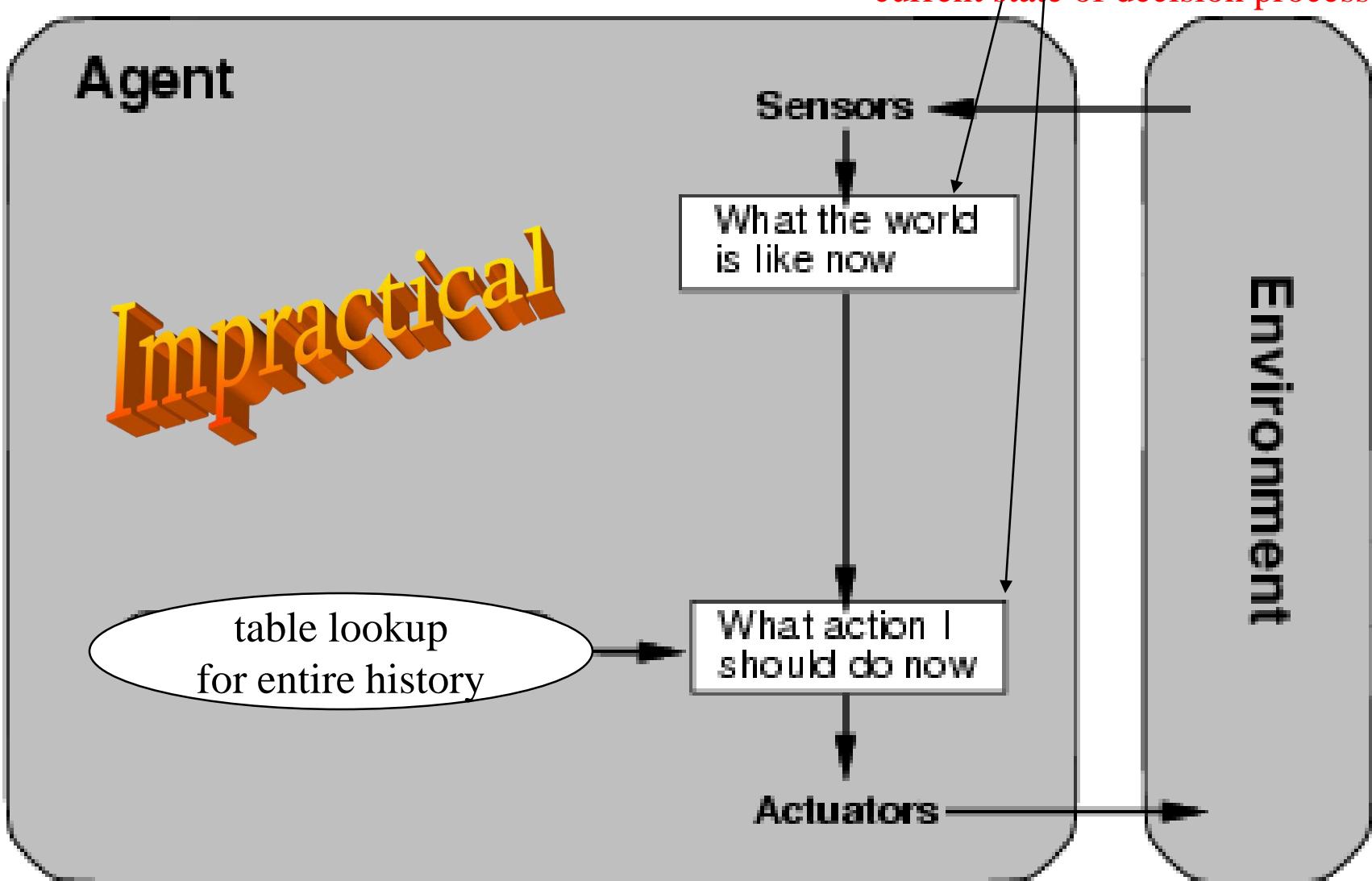
Environment Types

- **Fully observable** (vs. **partially observable**): An agent's sensors give it access to the complete state of the environment at each point in time.
- **Deterministic** (vs. **stochastic**): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is **strategic**)
- **Episodic** (vs. **sequential**): An agent's action is divided into atomic episodes. Decisions do not depend on previous decisions/actions.

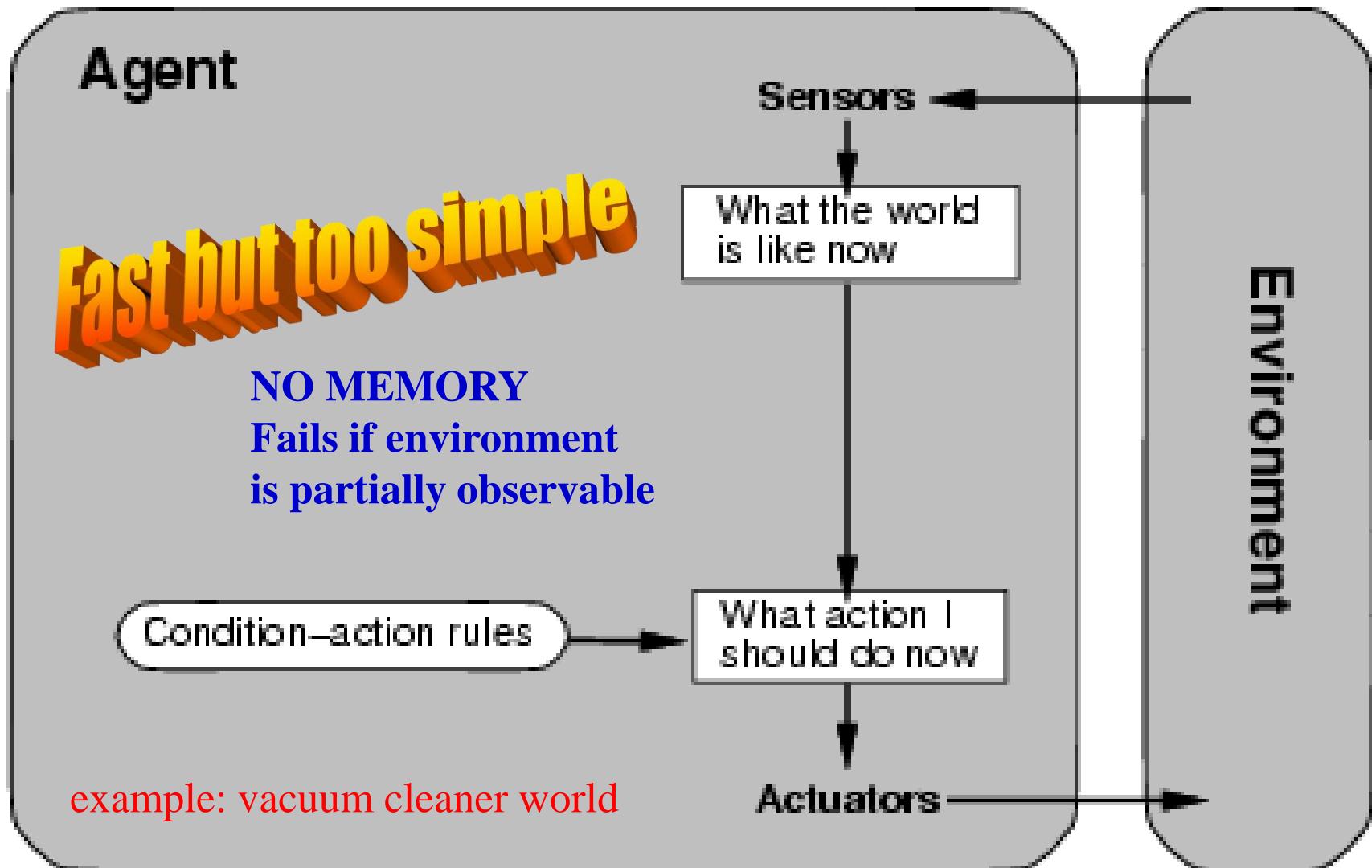
Environment Types

- **Static** (vs. **dynamic**): The environment is unchanged while an agent is deliberating. (The environment is **semidynamic** if the environment itself does not change with the passage of time but the agent's performance score does)
- **Discrete** (vs. **continuous**): A limited number of distinct, clearly defined percepts and actions.
How do we **represent** or **abstract** or **model** the world?
- **Single agent** (vs. **multi-agent**): An agent operating by itself in an environment. Does the other agent interfere with my performance measure?

Table Driven Agent.



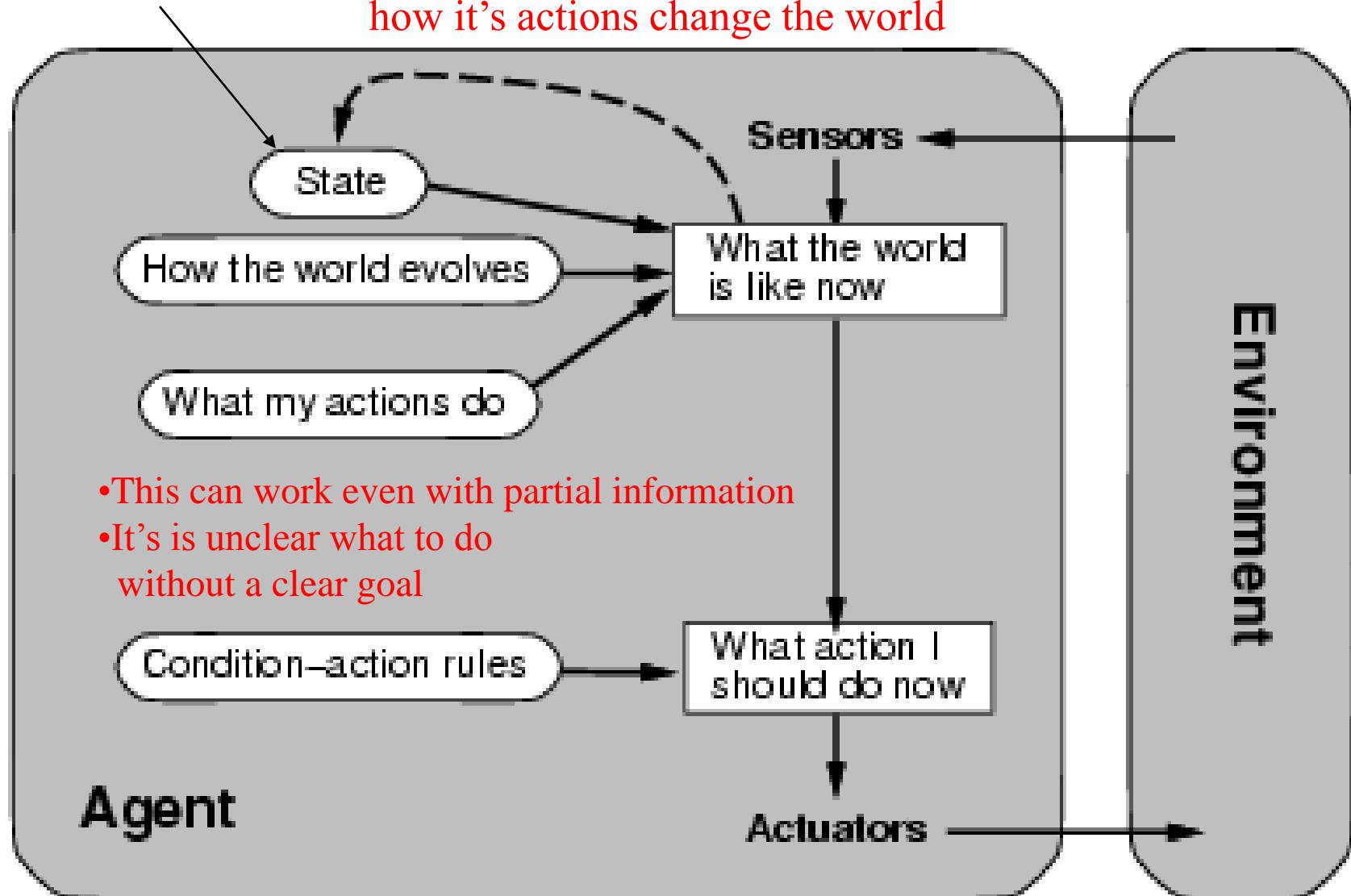
Simple reflex agents



Model-based reflex agents

description of
current world state

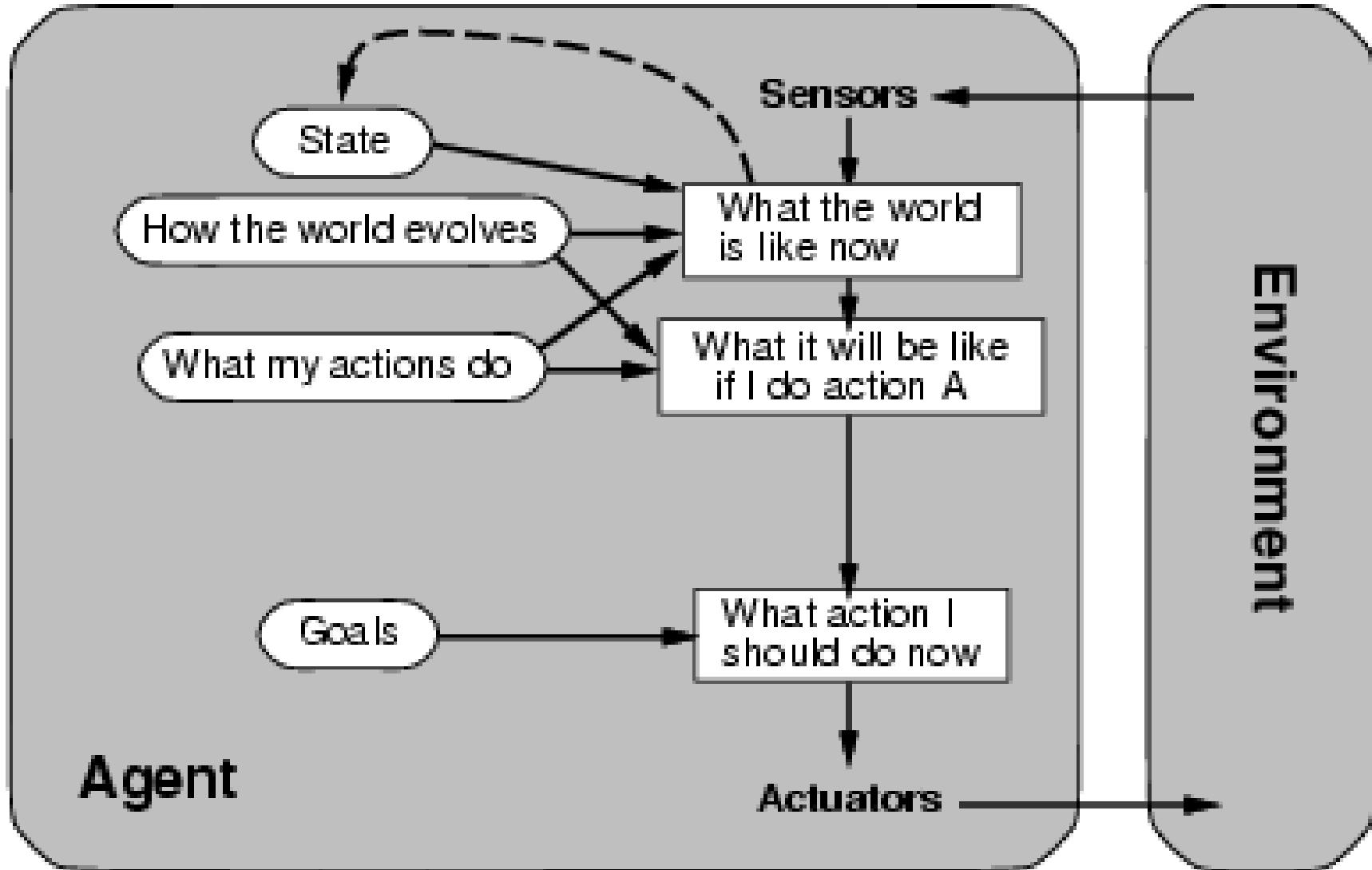
Model the state of the world by:
modeling how the world changes
how its actions change the world



Goal-based agents

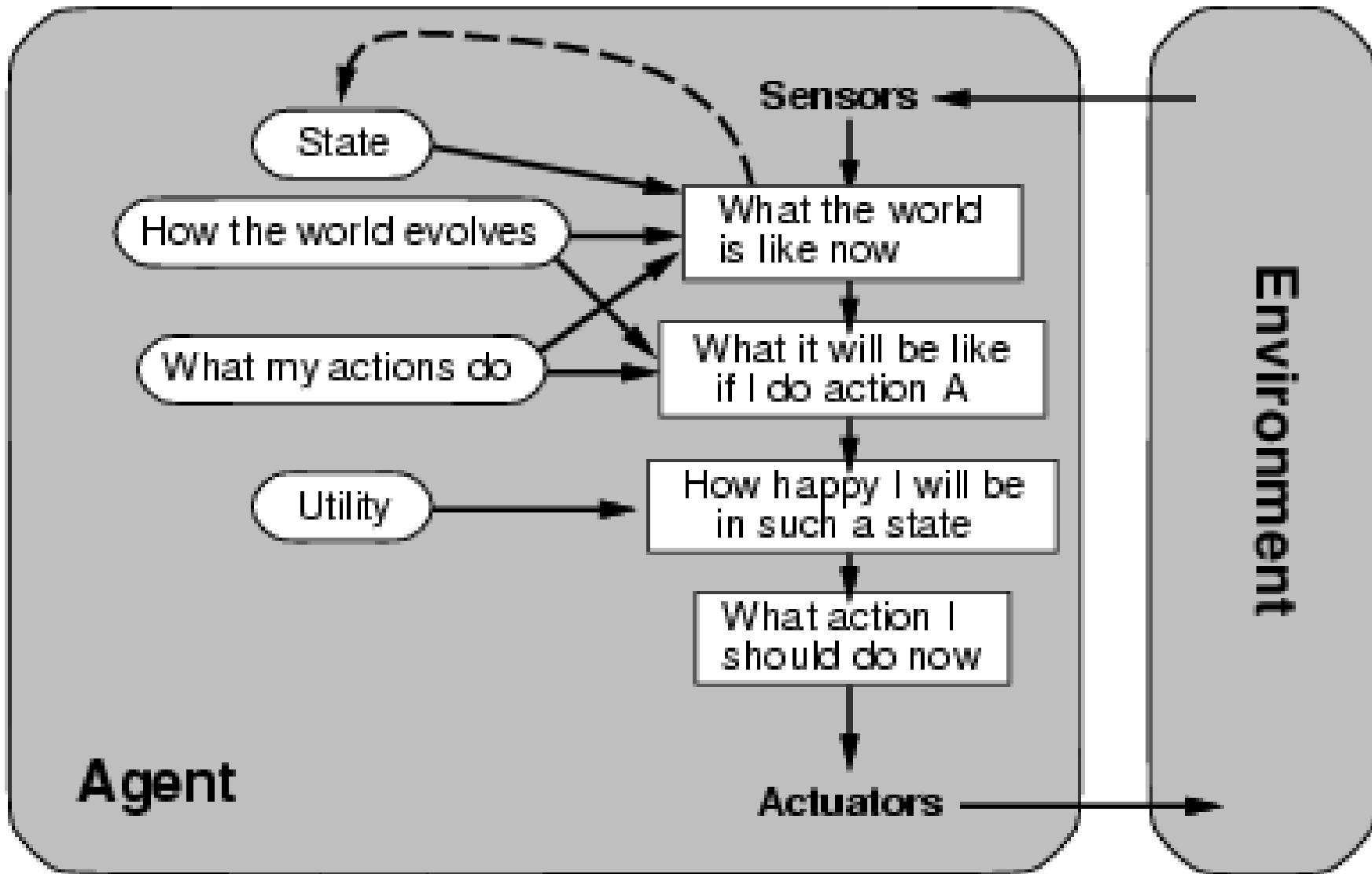
Goals provide reason to prefer one action over the other.

We need to predict the future: we need to plan & search



Utility-based agents

Some solutions to goal states are better than others.
Which one is best is given by a utility function.
Which combination of goals is preferred?

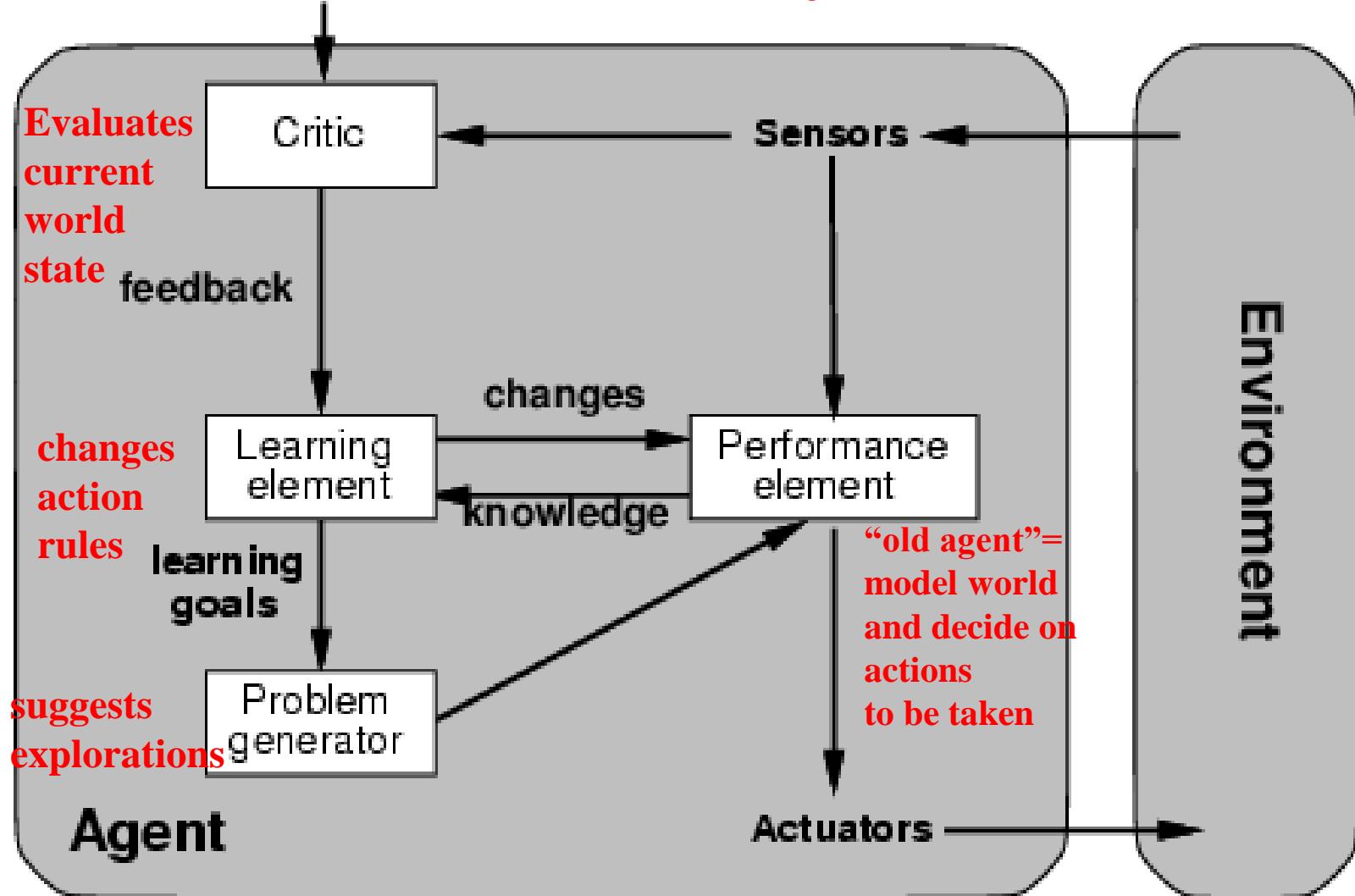


Learning agents

How does an agent improve over time?

By monitoring it's performance and suggesting

Performance standard better modeling, new action rules, etc.



Summary

- **What is Artificial Intelligence?**
 - modeling humans thinking, acting, should think, should act.
- **History of AI**
- **Intelligent agents**
 - We want to build agents that act rationally
- **Real-World Applications of AI**
 - AI is alive and well in various “every day” applications
 - many products, systems, have AI components
- **Assigned Reading**
 - Chapters 1 and 2 in the text R&N

Introduction to search

- Artificial Intelligent

UNIT-2ND

PRASHANT TOMAR

Problem solving and search

In intelligent agents knowledge base corresponds to environment, operators correspond to sensors and the search techniques are actuators. Hence, three parameters of an intelligent are....

1. Knowledge base
2. Operators
3. Control strategy(search technique)

The knowledge base describes the current task domain and the goal. In other words ,goal is nothing but the state. Operator are manipulate the knowledge base .

The control strategy decides what operators to apply and where. The aim of any search technique is the application of an appropriate sequence of operators to initial state to achieve the goal .

The objective to get the goal state : The objective can be achieved in two ways.

Forward reasoning: It refer to the application of operators to those structure in the knowledge base that describe the task domain in order to produce modified state . Such a method is also referred to as bottom up and data driven reasoning.

Backward reasoning: It break down the goal(problem) statement into sub goals which are easier to solve and whose solution are sufficient to solve original problem.

A problem solving agent or system uses either forward or backward reasoning. Each of its operators works to produce a new state in the knowledge base which is said to represent problem in a state space.

Problem Formulation: The problem can be defined formally by four component:

- i. **The Initial state :** The start state
- ii. **State space:** It involve a description of all the possible action available , i.e. , the set of all states reachable from the initial state . the state space forms a graph in which the nodes are states and the arcs b/t node are actions .
- iii. **Goal Test:** It determines whether a given state is goal state .
- iv. **Path cost:** It is a function that assign a numeric cost to each path.

A solution to a problem is a path from the initial state to a goal path

Any problem can be solved by the following series of step:

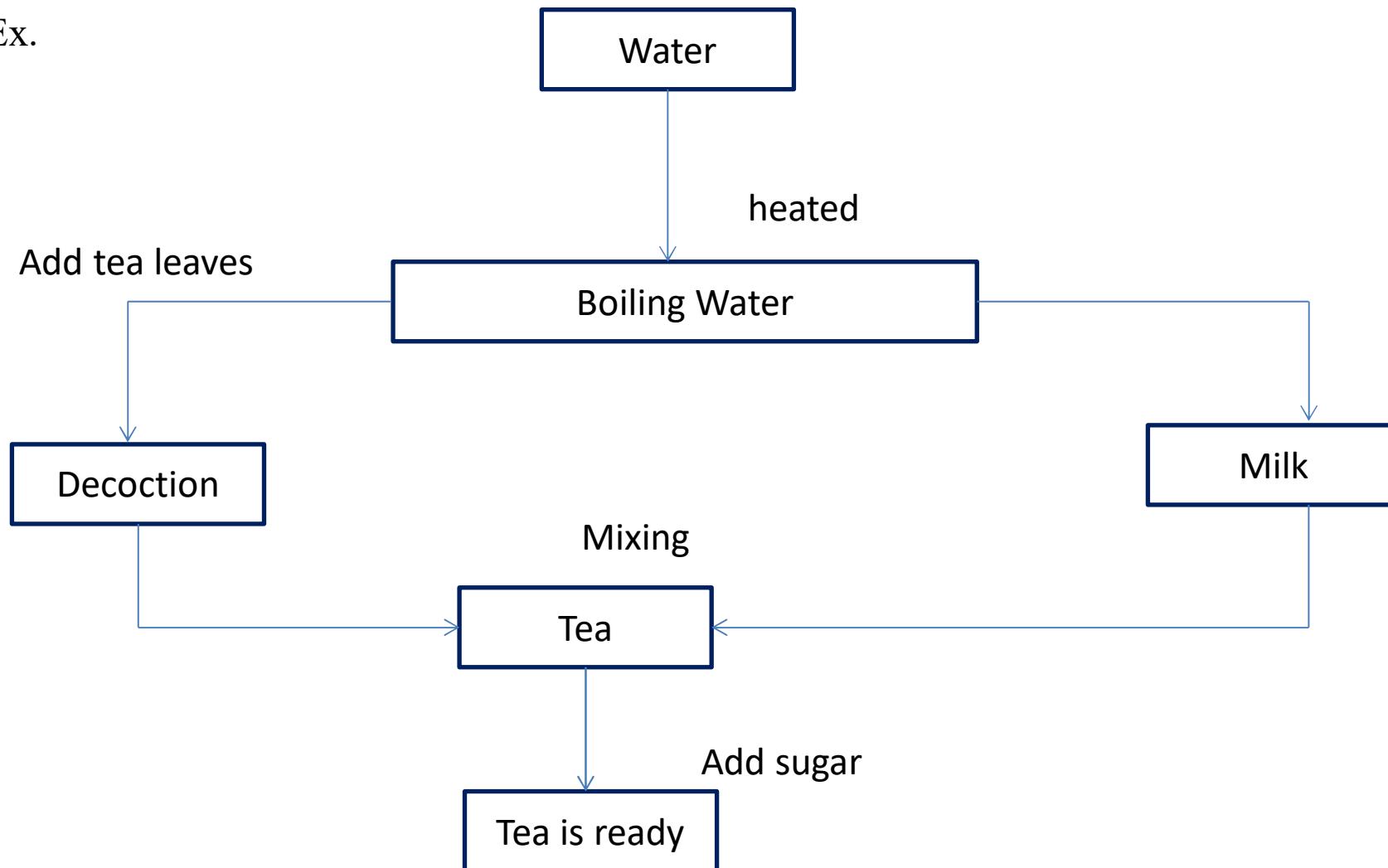
1. Define a state space which contain all the possible configuration of the relevant object.
2. Specify one or more states within that space which would describe possible situation from which find out the initial states.
3. Specify one or more states which would be acceptable as solution to the problem .the state are called goal states.
4. Specify a set of rules which describe the actions (operators) available and a control strategy to decide the order of application of these rules.

The most common methods of problem solving representation in AI..

1. State space representation
2. Problem reduction

State space Representation: Set of all possible state is known as the state space of the problem.

Ex.



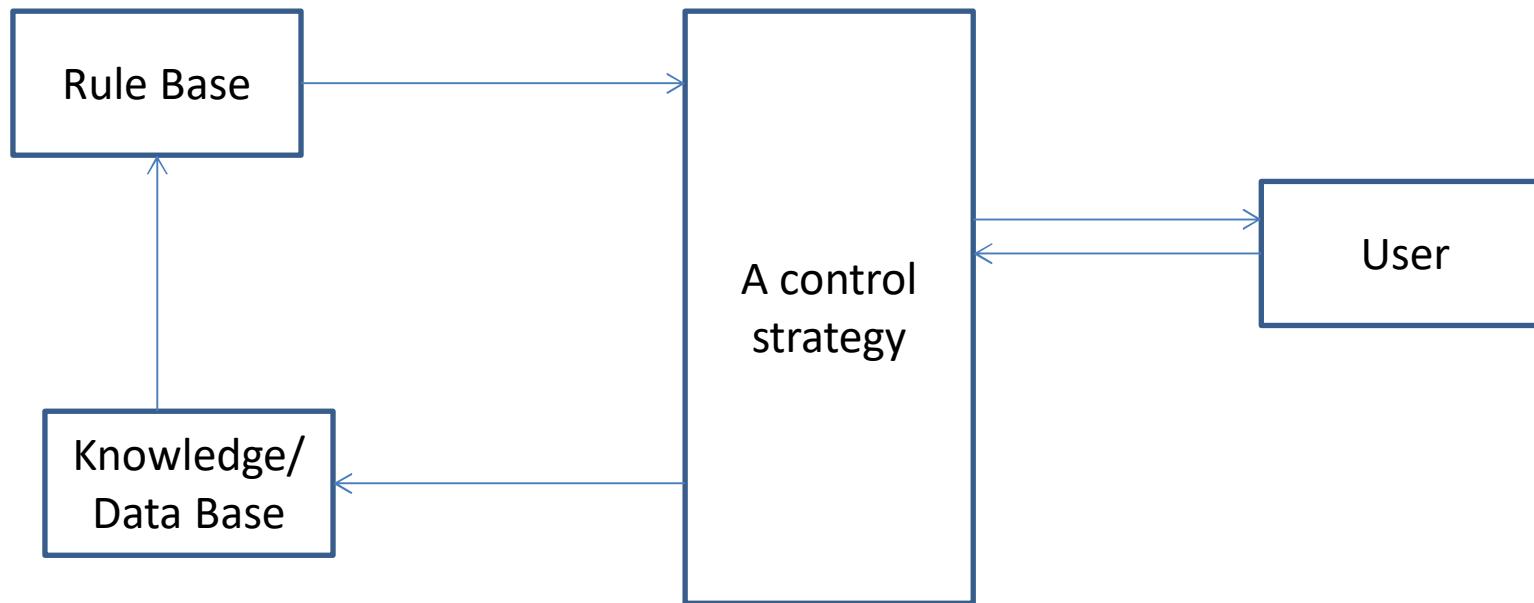
State space representation for tea making

Production System: The main function of the production system is to provide a useful tool for problem solving in A.I. Production System are Frequently referred to as inferential system or rule base –based system or production system.

It is useful to structure AI problems in a way that facilitates describing & performing the search process. Production systems provide these structures. A production system consists of following components----

1. **A set of rules** each consisting of a left side that determines the acceptability of the rule and a right that describes the operation to be performed if the rule is applied.
2. One or more **knowledge bases** that contain whatever information are appropriate for a particular task.
3. **A control strategy** that specifies the order in which in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
4. **A rule applier(user).**

All of these components provide the overall architecture of a production system and allow the programmer to write rules that define particular problems to be solved.



Block diagram of production system

Advantages of production systems:-

1. Production systems provide an excellent tool for structuring AI programs.
2. Production Systems are highly modular because the individual rules can be added, removed or modified independently.
3. The production rules are expressed in a natural form, so the statements contained in the knowledge base should be a recording of an expert thinking out loud.

Disadvantages of Production Systems:-

One important disadvantage is the fact that it may be very difficult analyze the flow of control within a production system because the individual rules don't call each other.

AI Problem:

- 1. Water Jug Problem**
- 2. Playing chess**
- 3. 8-puzzle problem**
- 4. Tic-tac-toe problem**
- 5. 8-queen problem**
- 6. The Tower of Hanoi problem**
- 7. The missionaries and cannibals problem**

Water Jug Problem: you are two jugs ,a 4-gallon and a 3- gallon one . Neither has any measuring markers on it .there is pump that can be used to fill the jugs with water. How can you exactly 2 gallons of water into the 4-gallon jug .Solve the problem using production Rules.

To solve the problem, we define rules whose left sides are matched against the current state and whose right sides describe the new state that results from applying the rule.

The assumptions are:

1. We can fill a jug from pump.
2. We can pour water out of the jug onto the ground.
3. We can pour water from one jug to another.
4. There are no measuring devices available.

To solve the water jug problem, we need a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, and the resulting state is checked if it corresponds to the goal state. As long it does not, the cycle continues.

Initial state: (0,0) start state.

State Space: The state space for this problem can be described as the set of ordered pair of integer(X,Y) such that $X=0,1,2,3,0r\ 4$ and $Y=0,1,2,0r\ 3$. X represent the number of gallons of water in the 4-gallon jug and Y represent the quantity water in the 3-gallon jug.

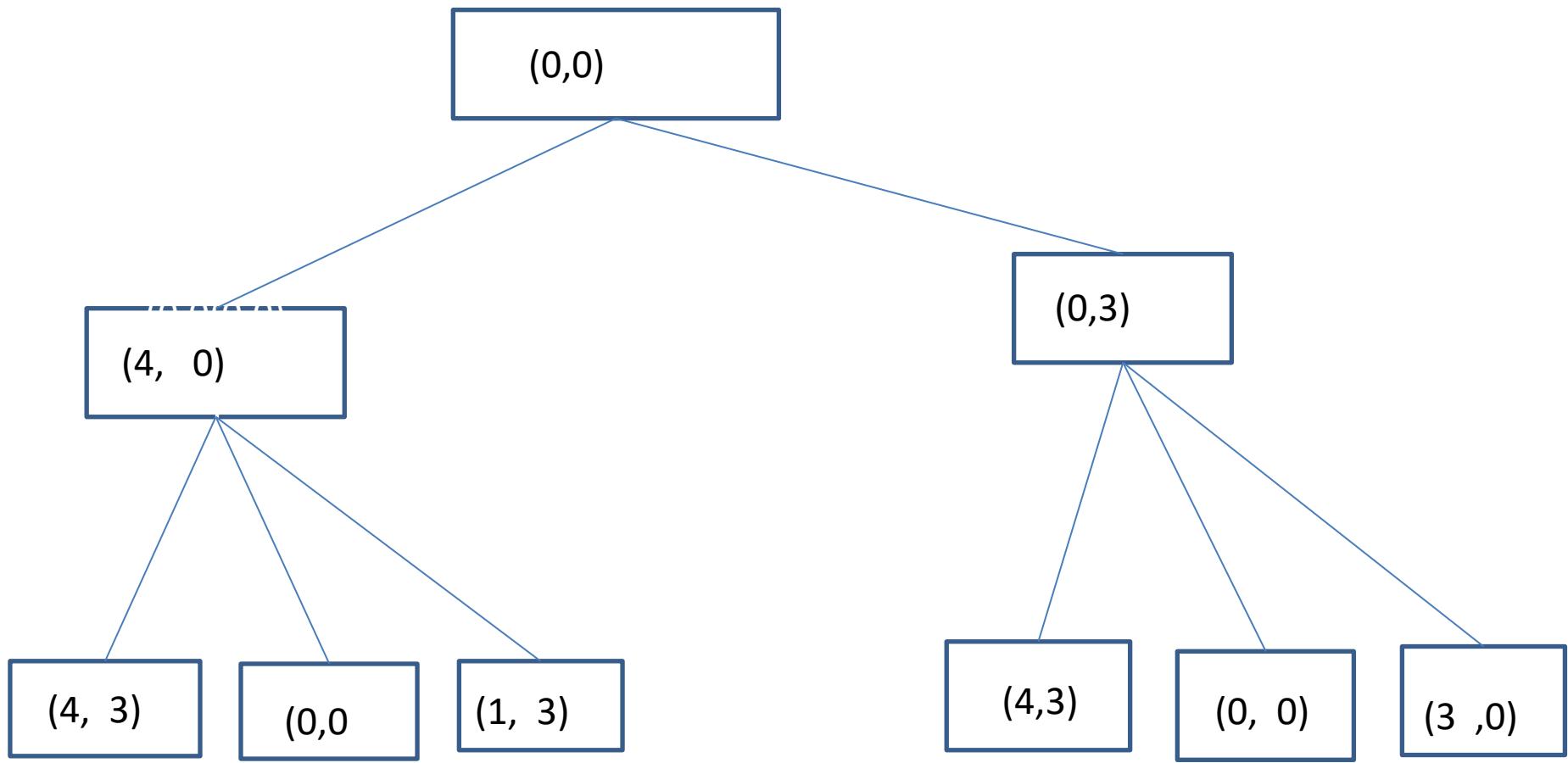
Production rules for water jug problem:

1.(X, Y) If($x < 4$)	- (4, Y)	Fill the 4-gallon jug
2. (X, Y) if $y < 3$	- (X, 3)	Fill the 3-gallon jug
3.(X, Y) if $x > 0$	- (X-d, Y)	Pour some water out of 4-gallon jug
4.(X, Y) if $y > 0$	- (X, Y-d)	Pour some water out of 3-gallon jug
5.(X, Y) if $x > 0$	- (0,Y)	Empty the 4-gallon jug on the ground
6. (X, Y) if $y > 0$	- (X, 0)	Empty the 3-gallon jug on the ground

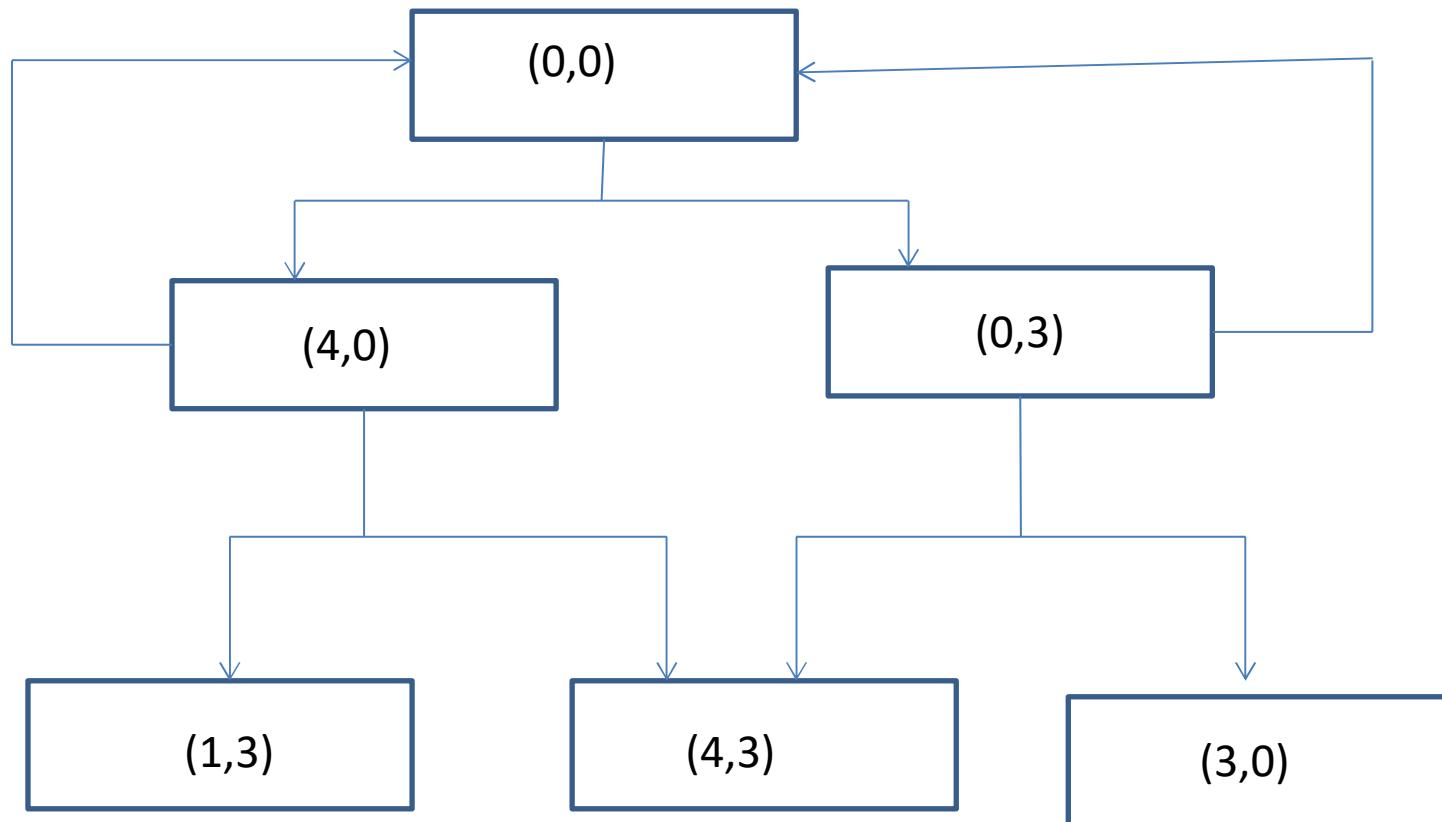
7. (X, Y)	- (4, y- (4-x))	Pour water from the 3 gallon jug into 4 gallon jug until 4-gallon jug is full if $x+y \geq 4$ and $y > 0$
8. (X, Y)	- (X- (3-y), 3)	Pour water from 4-gallon jug into 3- if gallon jug until 3-gallon jug is full $x+y \geq 3$ and $x > 0$
9.(X, Y)	- (X+Y, 0)	Pour all the water from the 3-gallon jug into the 4-gallon jug if $x+y \leq 4$ and $y > 0$
10. (X, Y)	- (0, x+y)	Pour all the water from the 4 gallon jug into the 3-gallon jug if $x+y \leq 3$ and $x > 0$
11.(0, 2)	-(2, 0)	Pour the 2-gallons from the 3-jug gallon into 4-gallon jug
12. (2, y)	- (0,y)	Empty the 2-gallons in the 4-gallon jug on the ground

Solution for water jug problem:

4-gallon jug	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 12



A search tree of a water jug problem



A search graph for the water Jug problem

8 Puzzle Problem.

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell.

1	2	3
8		4
7	6	5

Goal

2	8	3
1	6	4
7		5

Initial

The state space representation:

States space: A state is a description of each of the eight tiles in each location that it can occupy.

Operators/Action: The blank moves left, right, up or down

Goal Test: The current state matches a certain state

Path Cost: Each move of the blank costs 1

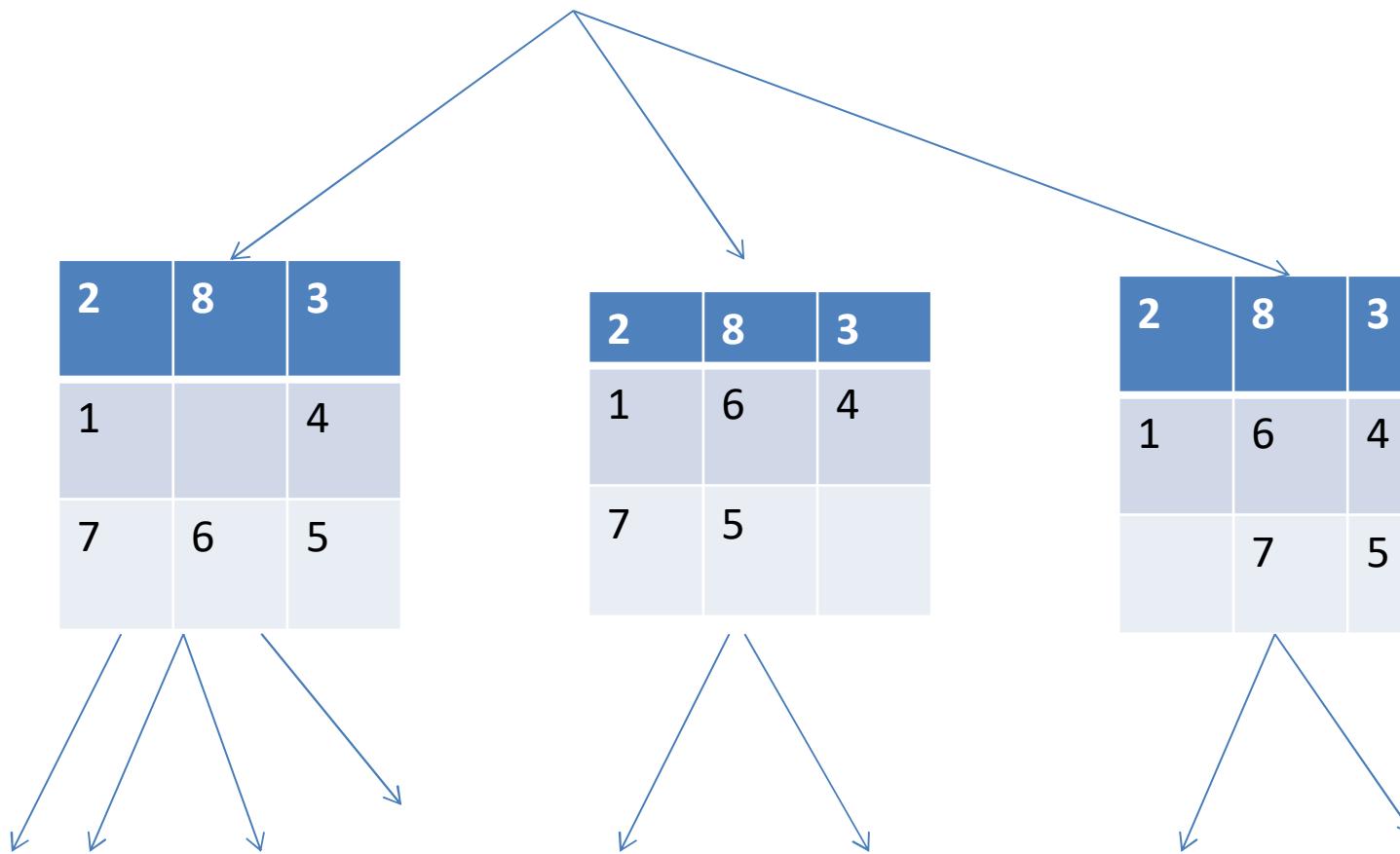
Note that we do not need to generate all the states before the search begins. The states can be generated when required.

2	8	3
1	6	4
7		5

2	8	3
1		4
7	6	5

2	8	3
1	6	4
7	5	

2	8	3
1	6	4
	7	5



2	8	3
1		4
7	6	5



2		3
1	8	4
7	6	5



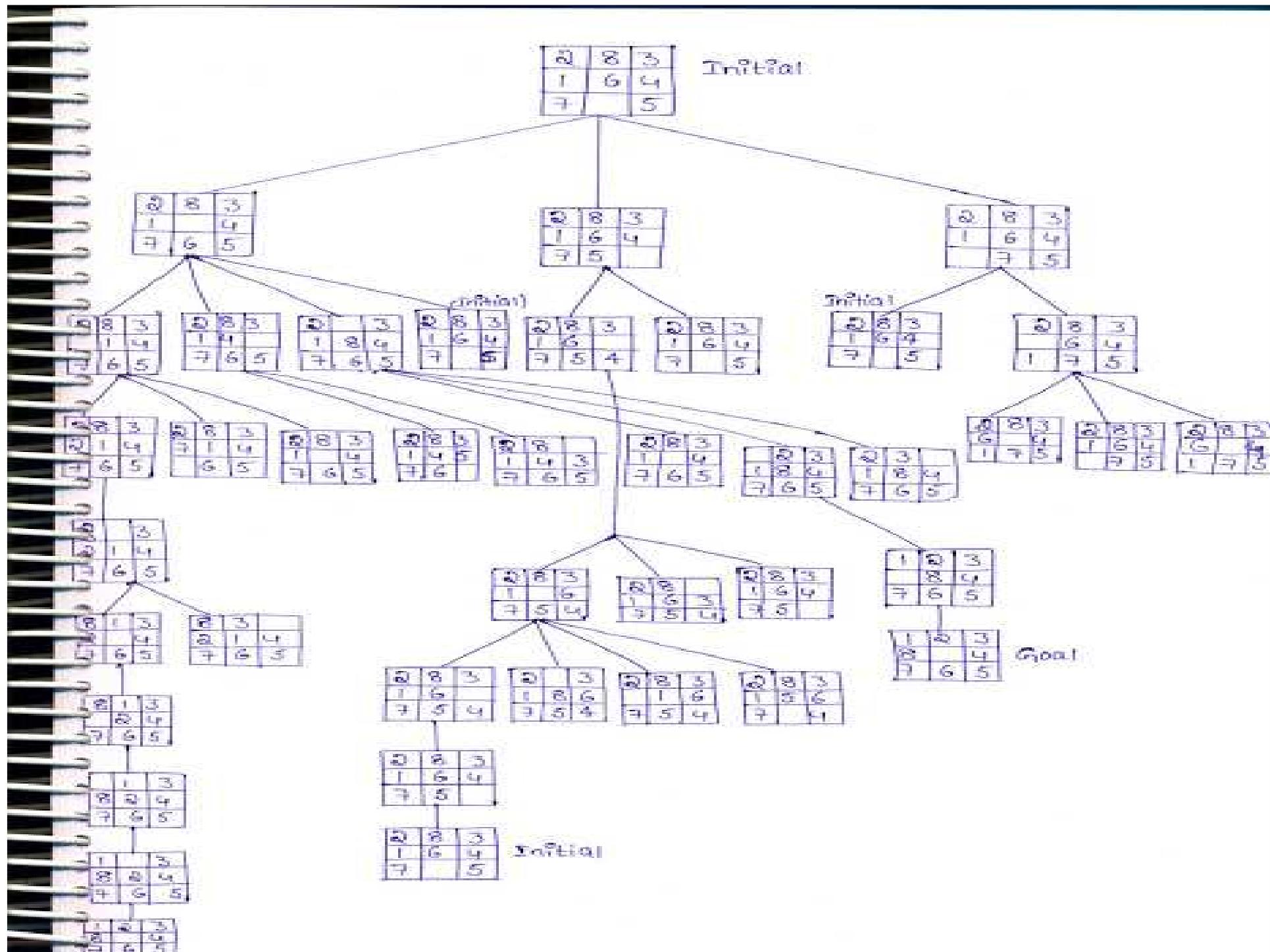
	2	3
1	8	4
7	6	5



1	2	3
8		4
7	6	5



1	2	3
	8	4
7	6	5

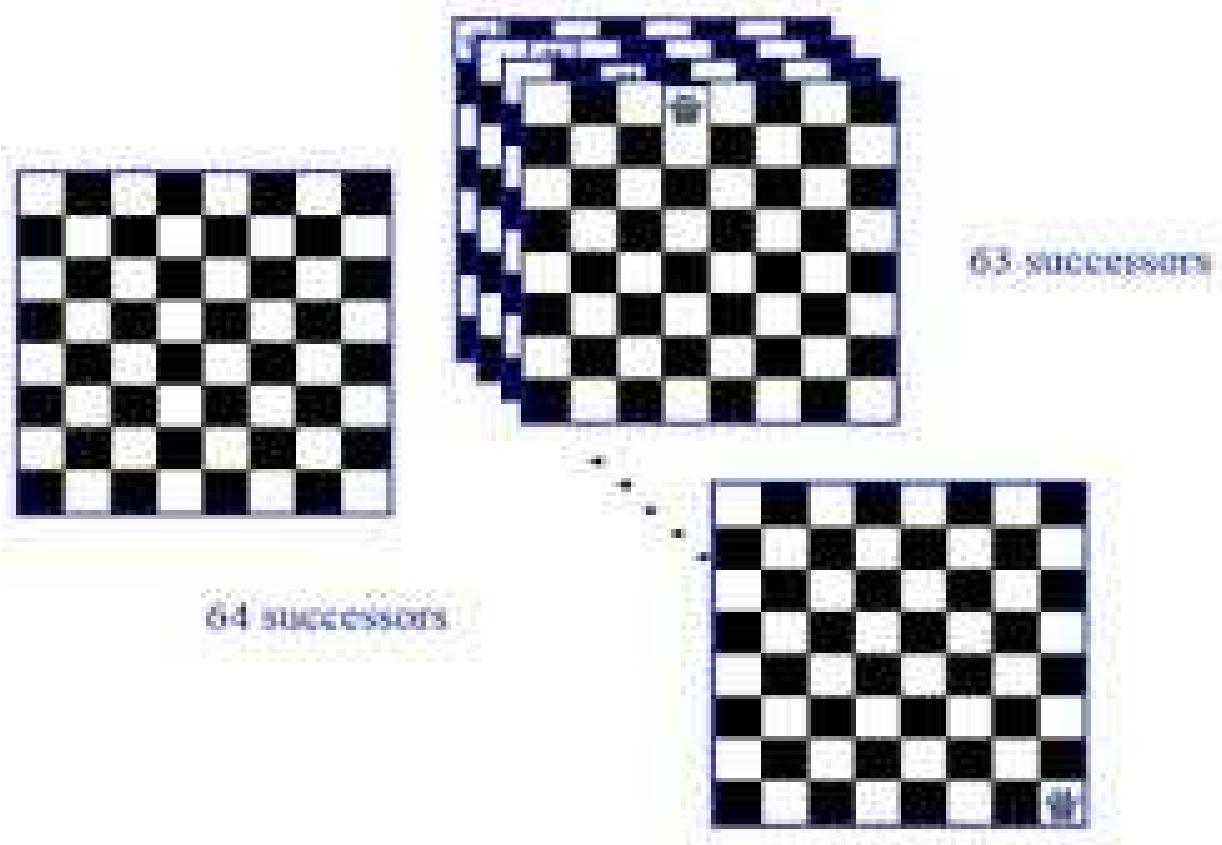


8-queens problem :The problem is to place 8 queens on a chessboard so that no two queens are in the same row, column or diagonal .How do we formulate this in terms of a state space search problem? The problem formulation involves deciding the representation of the states, selecting the initial state representation, the description of the operators, and the successor states. We will now show that we can formulate the search problem in several different ways for this problem.

N queens problem formulation 1

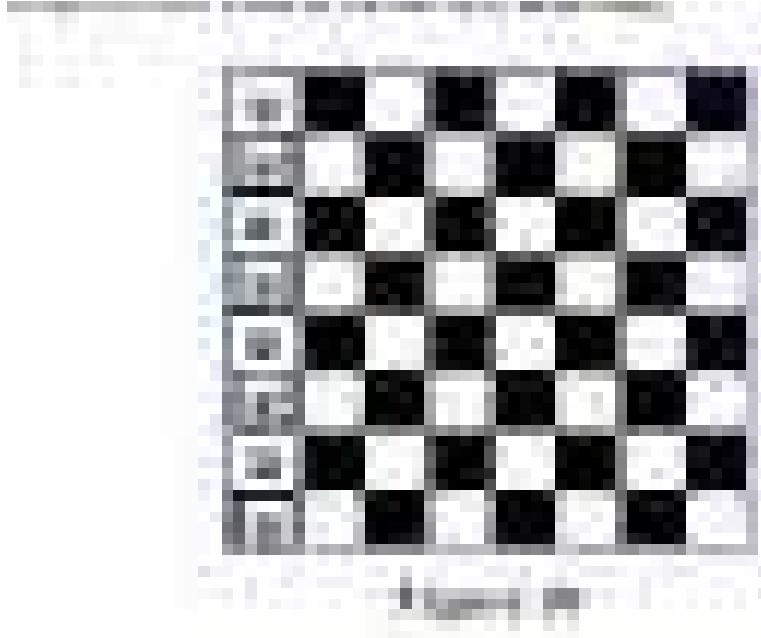
- **States:** Any arrangement of 0 to 8 queens on the board
- **Initial state:** 0 queens on the board
- **Successor function:** Add a queen in any square
- **Goal test:** 8 queens on the board, none are attacked

The initial state has 64 successors. Each of the states at the next level have 63 successors, and so on. We can restrict the search tree somewhat by considering only those successors where no queen is attacking each other. To do that we have to check the new queen against all existing queens on the board. The solutions are found at a depth of 8.



N queens problem formulation 2

- **States:** Any arrangement of 8 queens on the board
- **Initial state:** All queens are at column 1
- **Successor function:** Change the position of any one queen
- **Goal test:** 8 queens on the board, none are attacked

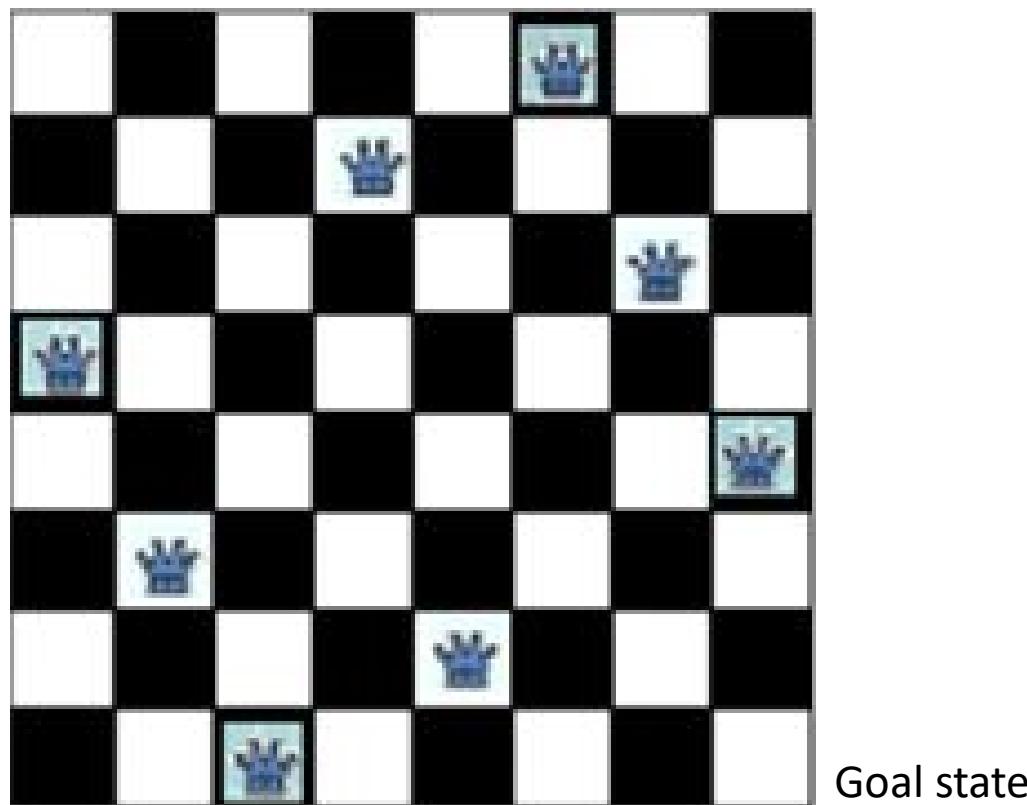


If we consider moving the queen at column 1, it may move to any of the seven remaining columns.

N queens problem formulation 3

- **States:** Any arrangement of k queens in the first k rows such that none are attacked
- **Initial state:** 0 queens on the board
- **Successor function:** Add a queen to the $(k+1)$ th row so that none are attacked.
Keep on shuffling the queen until the goal is reached.
- **Goal test :** 8 queens on the board, none are attacked

This formulation is more systematic hence it is called as iterative formulation.



Playing Chess:

State Space Search: Playing Chess

Each position can be described by an 8-by-8 array.

Initial position is the game opening position.

Goal position is any position in which the opponent does not have a legal move and his or her king is under attack.

Legal moves can be described by a set of rules:

- Left sides are matched against the current state.
- Right sides describe the new resulting state.

- **State space** is a set of legal positions.
- Starting at the initial state.
- Using the set of rules to move from one state to another.
- Attempting to end up in a goal state

- i. There are roughly 10^{120} board position. So it is difficult problem.
- ii. No program can easily handle all rules

In order to handle ,to minimize such problem we introduce some convenient notation.

White pawn at

Square(file e , rank 2) Move pawn from

AND

Square(file e , rank 3)->Square(file e , rank 2) to Square(file e , rank 4);

AND

Square(file e , rank 4) is empty

Searching For Solutions: Solution to an AI problem involves performing an action to go to one proper state among possible numerous possible state of agent. Thus the process of finding solution can be boiled down to searching of that best state among all the possible state.

Search Strategy: We will evaluate strategy in term of four criteria.....

- 1. Completeness:** Is the strategy, guaranteed to find a solution when there is one ?
- 2. Time Complexity:** How long does it take to find solution ?
- 3. Space Complexity:** How much memory does it need to perform the search?
- 4. optimality:** Does the strategy find the highest quality solution when there are several different solution ?

Search Strategies:

1. Uninformed search or blind search:

- Uninformed search means , searching through the state space without using any extra information or domain specific information
- Having no information about the number of steps from the current state to the goal.

2. Informed search or heuristic search:

- More efficient than uninformed search.

3. Constraint Satisfaction Search

4. Adversary Search

Uninformed search or blind search:

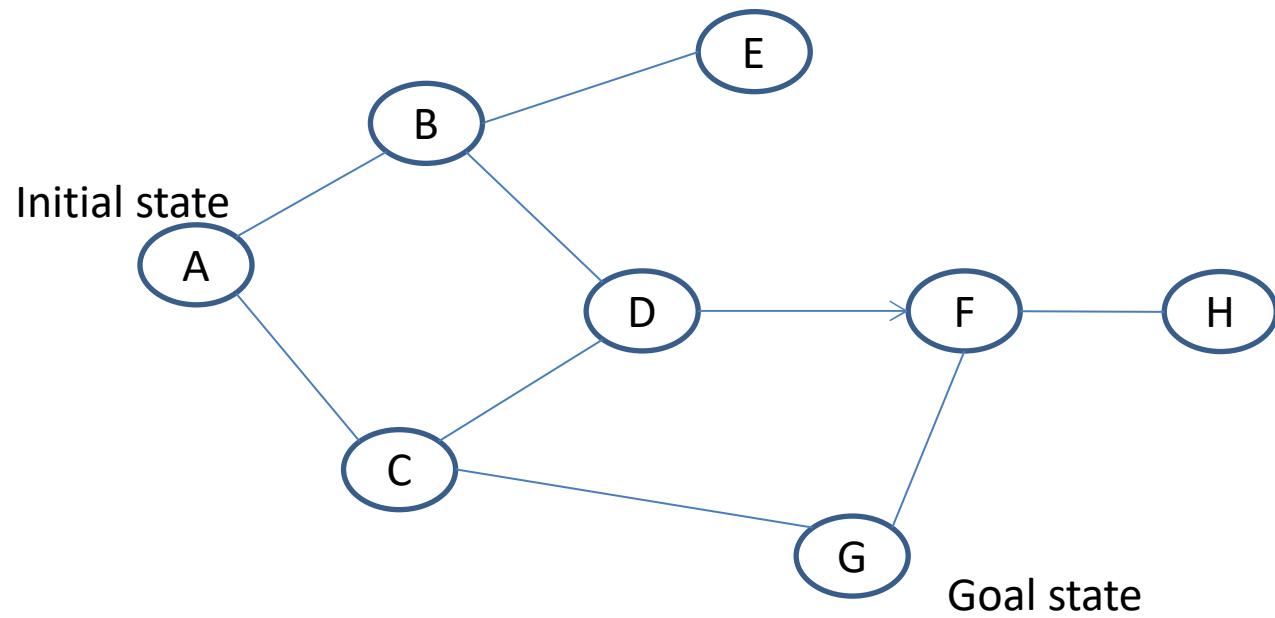
1. Depth first search
2. Breadth first search
3. Iterative deepening search
4. Bidirectional Search
5. Uniform cost search

Blind Search: Blind search or uninformed search that does not use any extra information about the problem domain. The two common methods of blind search are:

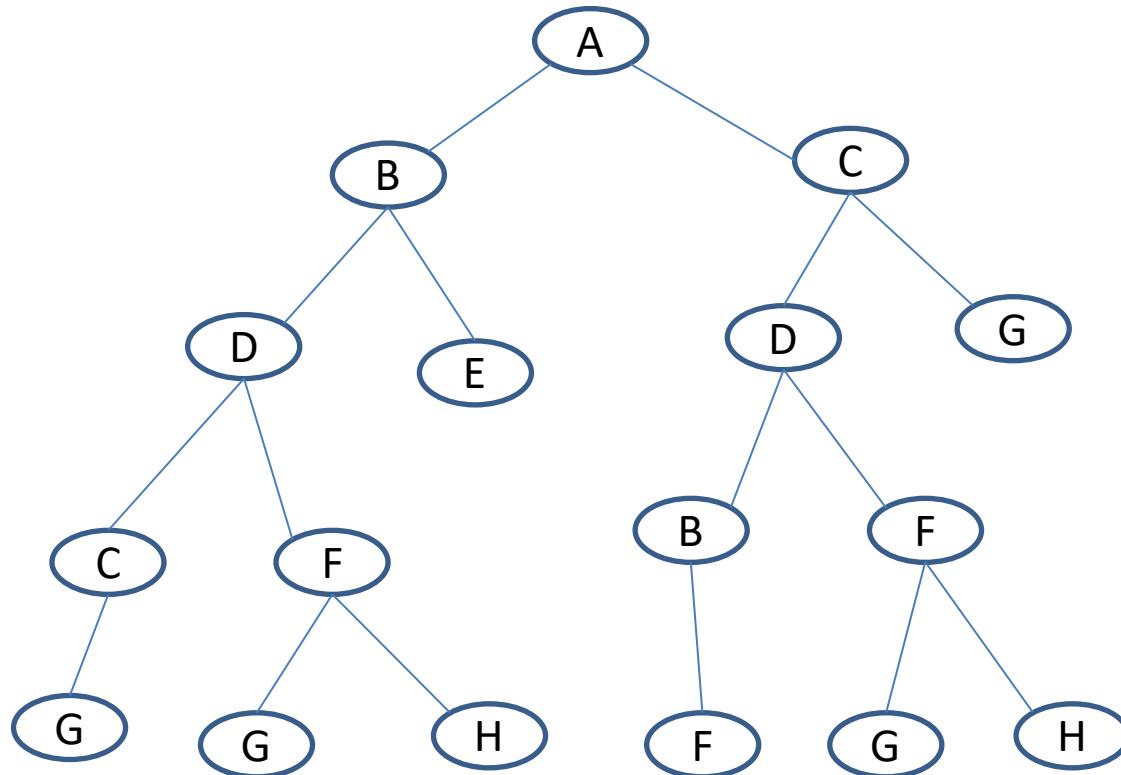
- BFS or Breadth First Search
- DFS or Depth First Search

Search Tree

A search tree is a data structure containing a root node, from where the search starts. Every node may have 0 or more children. If a node X is a child of node Y, node Y is said to be a parent of node X.



State space graph



A Search tree for state space graph

Search Tree – Terminology

- **Root Node:** The node from which the search starts.
- **Leaf Node:** A node in the search tree having no children.
- **Ancestor/Descendant:** X is an ancestor of Y if either X is Y's parent or X is an ancestor of the parent of Y. If S is an ancestor of Y, Y is said to be a descendant of X.
- **Branching factor:** the maximum number of children of a non-leaf node in the search tree
- **Path:** A path in the search tree is a complete path if it begins with the start node and ends with a goal node. Otherwise it is a partial path.

We also need to introduce some data structures that will be used in the search algorithms.

Node data structure

1. **A state description**
2. **A pointer to the parent of the node**
3. **Depth of the node**
4. **The operator that generated this node**
5. **Cost of this path (sum of operator costs) from the start state**

The nodes that the algorithm has generated are kept in a data structure called OPEN or **fringe**. Initially only the start node is in OPEN.

The search starts with the root node. The algorithm picks a node from OPEN for expanding and generates all the children of the node. Expanding a node from OPEN results in a closed node. Some search algorithms keep track of the closed nodes in a data structure called CLOSED.

A solution to the search problem is a sequence of operators that is associated with a path from a start node to a goal node. The cost of a solution is the sum of the arc costs on the solution path. For large state spaces, it is not practical to represent the whole space.

The search process constructs a search tree, where

- **root** is the initial state and
- **leaf nodes** are nodes
 - not yet expanded (i.e., in fringe) or
 - having no successors (i.e., “dead-ends”)

Search tree may be infinite because of loops even if state space is small

Breadth First Search: First Search is a great algorithm for getting the shortest path to your goal. Breadth First Search by the name itself suggests that the breadth of the search tree is expanded fully before going to the next step.

Two list:

Open List/fringe: initial node

Closed List: <empty>

Algorithm:

Let OPEN be a list containing the initial state and a CLOSED list that store the remove Node

Loop

 if OPEN is empty return failure

 Node \leftarrow remove-first (OPEN) , and

 Add to CLOSED List

 if Node is a goal

 then

 return the path from initial state to Node

 else

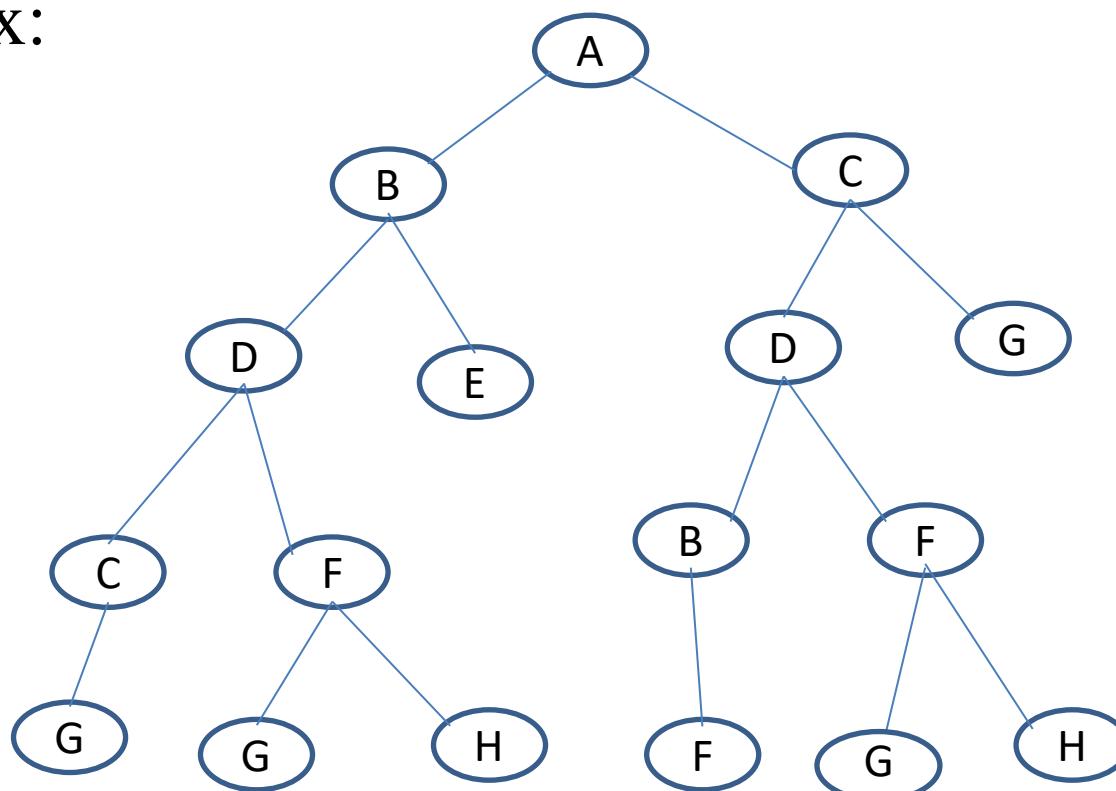
 generate all successors of Node, and

 add generated nodes to the back of fringe(add in end of OPEN list)

 End Loop

In breadth first search the newly generated nodes are put at the back of fringe or the OPEN list , will be expanded in a FIFO (First In First Out) order. The node that enters OPEN earlier will be expanded earlier.

Ex:



Properties of BFS:

We assume that every non-leaf node has b children. Suppose that d is the depth of the shallowest goal node, and m is the depth of the node found first.

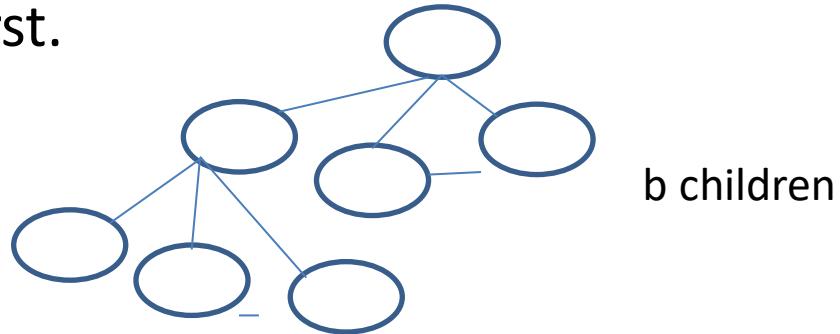


Figure 9: Model of a search tree with uniform branching factor b

Breadth first search is:

Complete- Yes (if b is finite)

Time $1+b+b^2+b^3+\dots+b^d + (b^{d+1}-b)$ total nodes
= $O(b^{d+1})$

Space- $O(b^{d+1})$ (keeps every node in memory) where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node.

Optimal- Yes (if cost = 1 per step)

- The algorithm has exponential time and space complexity.

Advantage:

1. If there is solution ,BFS guaranteed to find it.
2. If there are multiple solution, a minimal solution will be found.

Disadvantage:

- 1.BFS requires more memory space because all the node of tree must be stored on that level
- 2.It take more time if the goal state occurs in depth.

Depth first Search: DFS always expands one of the node of the deepest level of the tree .DFS uses LIFO queue for keeping the unexpanded nodes . DFS has very modest memory requirement. It need to store only single path from root to a leaf node, along with the remaining unexpanded sibling node for each node on the path.

Algo:

Let fringe be a list containing the initial state

Loop

If fringe is empty return failure

 Node \leftarrow remove-first (fringe)

 if Node is a goal

 then return the path from initial state to Node

 else generate all successors of Node, and

 merge the newly generated nodes into fringe

 add generated nodes to the front of fringe

End Loop

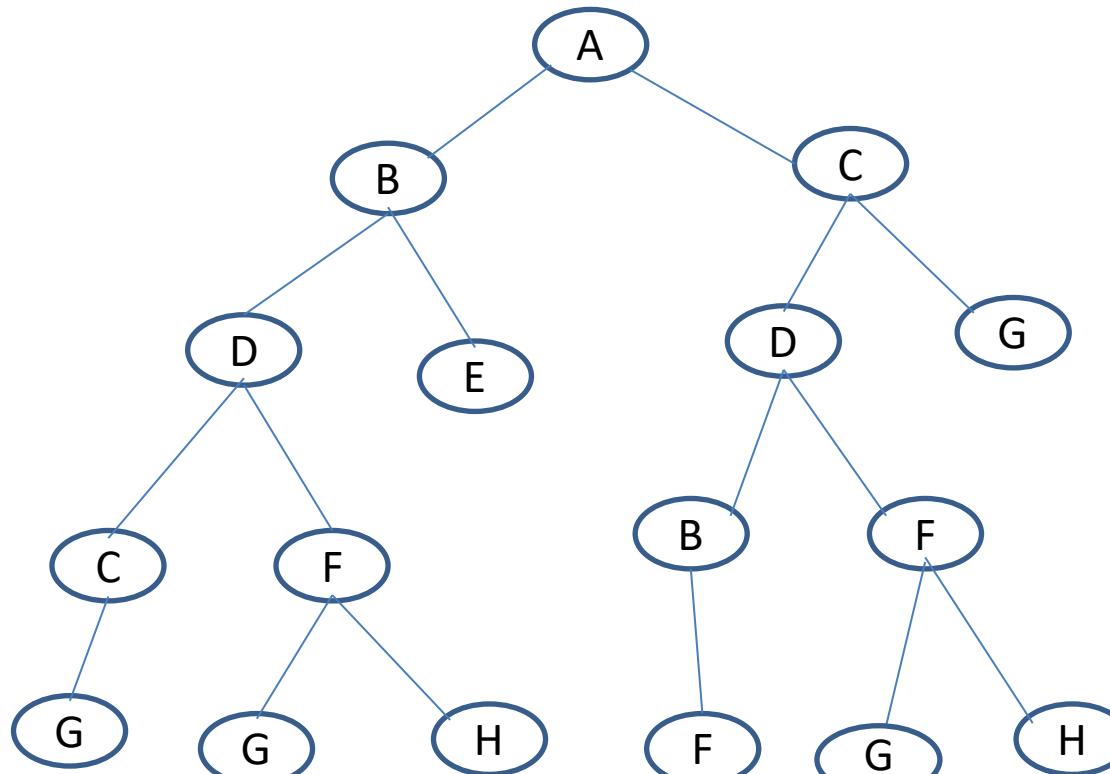
Note: *fringe* = LIFO queue, i.e., put
successors at front.

Ex.

Fringe:

A

- A is expanded and its children B and C are put in front of fringe.



B,C

D,E,C

C,F,E,C

G,F,E,C

Node G is expanded and found to be a goal node. The solution path A-B-D-C-G is returned and the algorithm terminates.

Properties of depth-first search:

Complete: No: fails in infinite-depth spaces.

→ complete in finite spaces

Time: $O(b^m)$: terrible if m is much larger than d

but if solutions are dense, may be much faster than breadth-first.

b=no. of expanded node

m=depth of the shallowest goal

Space: For a state space with branching factor **b** and maximum depth-m, then DFS require storage of only b^m nodes, then space complexity $O(bm)$, i.e., linear space.

Optimal: No

Advantage:

1. *DFS require less memory since only the node on the current path are stored.*
2. *If DFS finds solution without exploring much in a path then the time and space will be very less.*

Disadvantage:

1. If stop after one solution is found . So minimal solution may not be found.
2. *In DFS there is a possibility that may go down the left-most path forever , even a finite graph can generate a infinite tree*

Depth Limited Search :

A solution of Depth First Search problem . Define a limit in DFS and Nodes are only expanded if they have depth less than the limit. This algorithm is known as depth-limited search.

Let fringe be a list containing the initial state
Loop if fringe is empty return failure
Node \leftarrow remove-first (fringe)
if Node is a goal
then return the path from initial state to Node
else if depth of Node = limit return cutoff
else add generated nodes to the front of fringe
End Loop

Problem: If we choose limit at which depth solution is not found. Then search is not complete.

Depth-First Iterative Deepening (DFID): It is the solution of Depth limited search. The Idea is that if do not find the solution up to limit , increase the limit by one.

First do DFS to depth 0 (i.e., treat start node as having no successors), then, if no solution found, do DFS to depth 1, etc.

Algo:

```
until solution found do  
  DFS with depth cutoff c  
  c = c+1
```

Advantage :

- Linear memory requirements of depth-first search
- Guarantee for goal node of minimal depth

Properties: For large d the ratio of the number of nodes expanded by DFID compared to that of DFS is given by $b/(b-1)$. For a branching factor of 10 and deep goals, 11% more nodes expansion in iterative- deepening search than breadth-first search

The algorithm is

- Complete
- Optimal/Admissible if all operators have the same cost.
Otherwise, not optimal but guarantees finding solution of shortest length (like BFS).
- Time complexity is a little worse than BFS or DFS because nodes near the top of the search tree are generated multiple times, but because almost all of the nodes are near the bottom of a tree, the worst case time complexity is still exponential, $O(b^d)$.

If branching factor is b and solution is at depth d , then nodes at depth d are generated once, nodes at depth $d-1$ are generated twice, etc.

$$\text{Hence } b^d + 2b^{(d-1)} + \dots + b^d \leq b^d / (1 - 1/b)^2 = O(b^d).$$

- Linear space complexity, $O(b^d)$, like DFS

Depth First Iterative Deepening combines the advantage of BFS (i.e., completeness) with the advantages of DFS (i.e., limited space and finds longer paths more quickly). This algorithm is generally preferred for large state spaces where the solution depth is unknown.

Informed Search

Informed Search or heuristic search: We know that uninformed search methods systematically explore the state space and find the goal. They are inefficient in most cases. Informed search methods use problem specific knowledge, and may be more efficient. They often depend on the use of heuristic function.

1. Generate and test
2. Hill climbing – i) Simple hill climbing ii) Steepest –ascent hill climbing
3. Best first search
A* Search and AO* search(AND –OR graph searching)
4. Problem reduction
5. Means-ends analysis
6. Branch and bound

Heuristic function: Heuristic means “rule of thumb” or judgmental technique that leads to solution. Heuristics are criteria , method or principles that might not always find the best solution but it is guaranteed to find good solution in reasonable time. In heuristic search or informed search, heuristics are used to identify the most promising search path.

A heuristics function at a node n is an estimate of the optimum cost from the current node to goal node. It is denoted by $h(n)$.

$h(n)$ =estimated cost of the cheapest path from the n node to a goal node

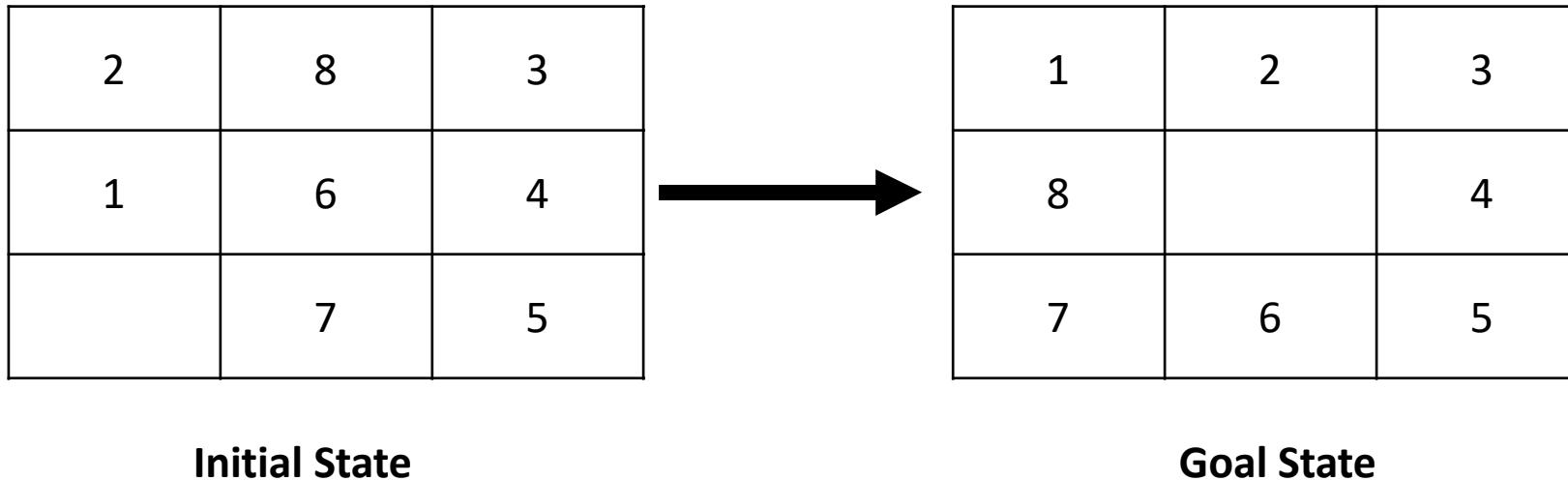
Ex . We want a path from mzn to delhi
heuristic for delhi may be straight -line distance
b/w mzn to delhi then heuristics function is-----

$$\underline{h(mzn) = \text{distance}(mzn, delhi)}$$

There are heuristics function of some problem.

8-puzzle- total no. of the misplaced tiles.

TSP- The sum of distance traveled so far.



total no. of the misplaced tiles = 5, because the tiles 2,8,1,6,7 are out of place, then heuristic function is $h(n)=4$

Manhattan Distance heuristics: Another heuristics for 8-puzzle is MDH. This heuristics sums the distance that the tiles are out of place. The distance of a titles is measured by the sum of the differences in the x-position and y-position.

$$h(n)=1+1+0+0+0+1+0+2=5$$

Generate and Test: The Generate and Test strategy is the simplest of all approaches.

Algorithm:

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space.
2. Test to see if this is a solution by comparing the chosen point or the end-point of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

If the generation of possible solutions is done systematically, then the process will find a solution, if it exists. This is called “*Exhaustive search*”.

If the generation of possible solutions is done randomly, there is no guarantee that a solution is found. This type of search is called “*British Museum Algorithm*”

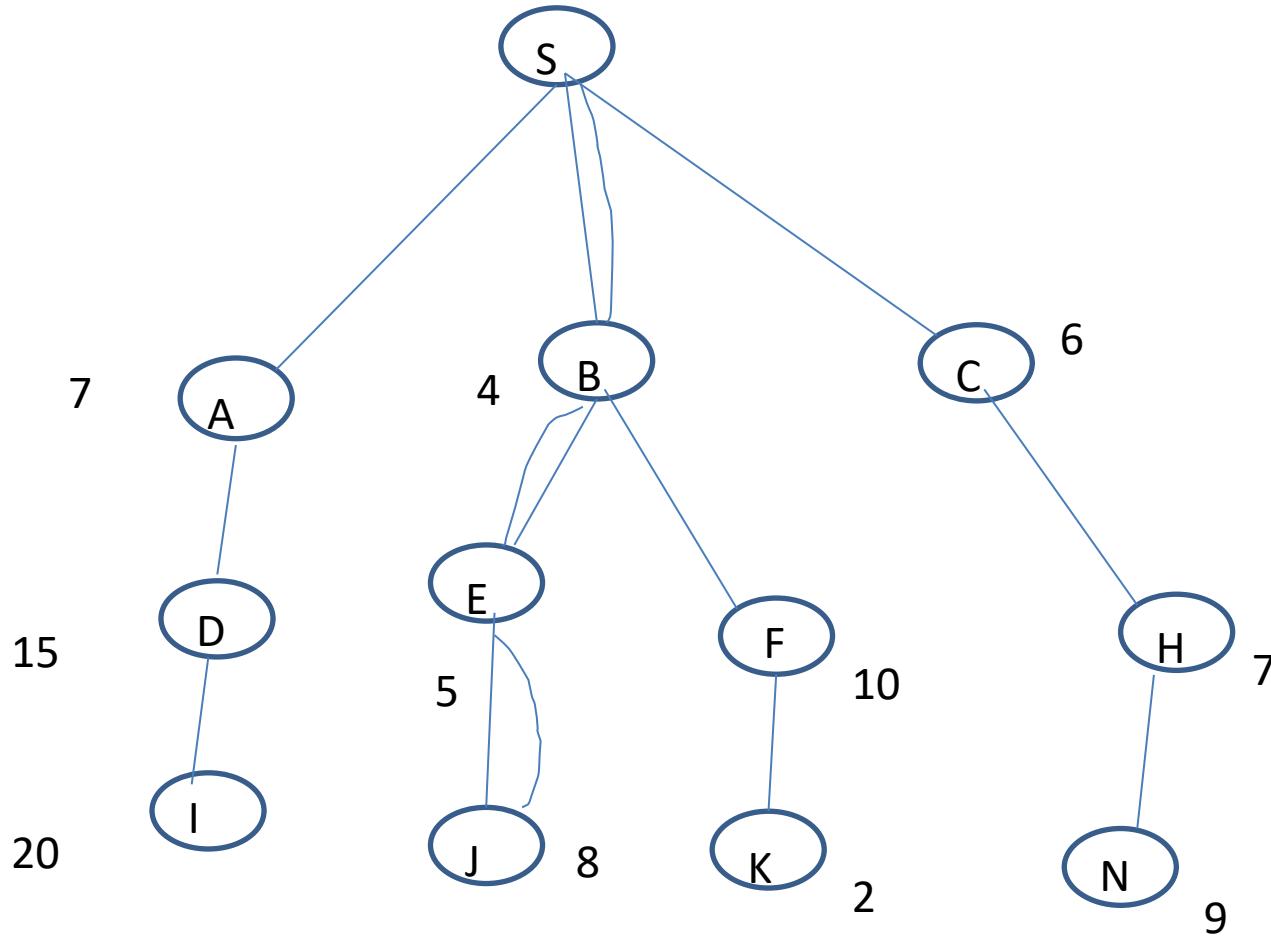
For simple problems, exhaustive generate-and-test is reasonable technique. For harder problems, the heuristic generate-and-test is not effective technique. This technique can be made effective by combining with other techniques.

HILL CLIMBING SEARCH: Hill climbing is the variant of generate –and- test in which feedback from the test procedure is used to help . The generator decide which direction to move in the search space . Here the G &T function is augmented by an heuristic function which measure the closeness of the current state to the goal state.

Hill climbing is often used when a good heuristics function is available for evaluating state but no other useful knowledge is available.

Simple Hill climbing: In simple hill climbing, the first closer node is chosen.

Steepest –Ascent hill climbing-In steepest ascent hill climbing all successors are compared and the closest to the solution is chosen. Also called gradient search.



Start at S

Children of $S = [A(7), B(4), C(6)]$

Best child of $S = B(4)$

Children of $S = [E(5), F(10)]$

Best child of $B = E(5)$

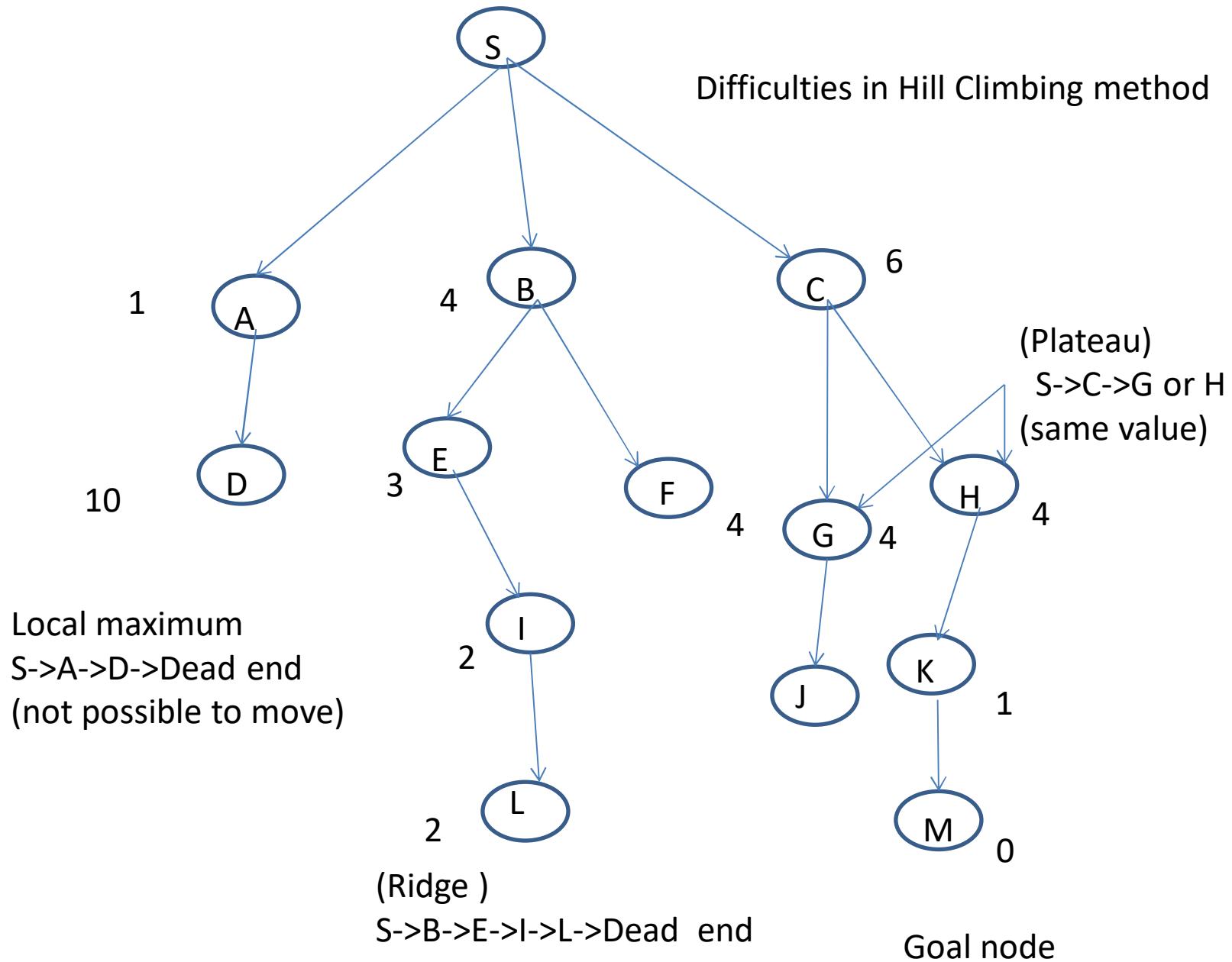
Children of $E = [J(8)]$

Difficulties in Hill climbing:

- (a) A "**local maximum**" which is a state better than all its neighbors , but is not better than some other states farther away. Local maxim sometimes occur with in sight of a solution. In such cases they are called " Foothills".
- (b) A "**plateau**" which is a flat area of the search space, in which neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.
- (c) A "**ridge**" which is an area in the search that is higher than the surrounding areas, but can not be searched in a simple move.

To overcome theses problems we can

- (a) Back track to some earlier nodes and try a different direction. This is a good way of dealing with local maxim.
- (b) Make a big jump an some direction to a new area in the search. If the rules available describe only single small steps then apply them several times in the same direction
- (c) Applying two more rules of the same rule several times, before testing. This is corresponding to moving in several directions at once



Algorithm:

1. Evaluate the initial state If it is goal state then quit ,otherwise make the current state this initial state and proceed;
2. Loop until a solution is found or until there are no new operator left to be applied in the current state
 - (a)Select an operator an operator and apply to the current state and procedure all its children as a new state
 - (b) Evaluate the best new state.
 - (i) If this state is goal state then return
 - (ii) If not a goal state but better than the current state then make it the current state
 - (iii) If not better than current state then continue in loop.

Best-First Search: This method is similar to the hill climbing, In hill climbing ,one move to be selected and all the other is rejected, never to be considered .But in best-first search we can not reject node but kept around so that they can be revisited later.

Family of best first search algorithms exists with different evaluation functions.

A key component is a **heuristic function $h(n)$:**

$h(n) = \text{estimated cost of the cheapest path from node } n \text{ to a goal node.}$

If n is goal node, $h(n)=0$.

Special cases:

greedy best-first search

A^* search

Greedy Search:

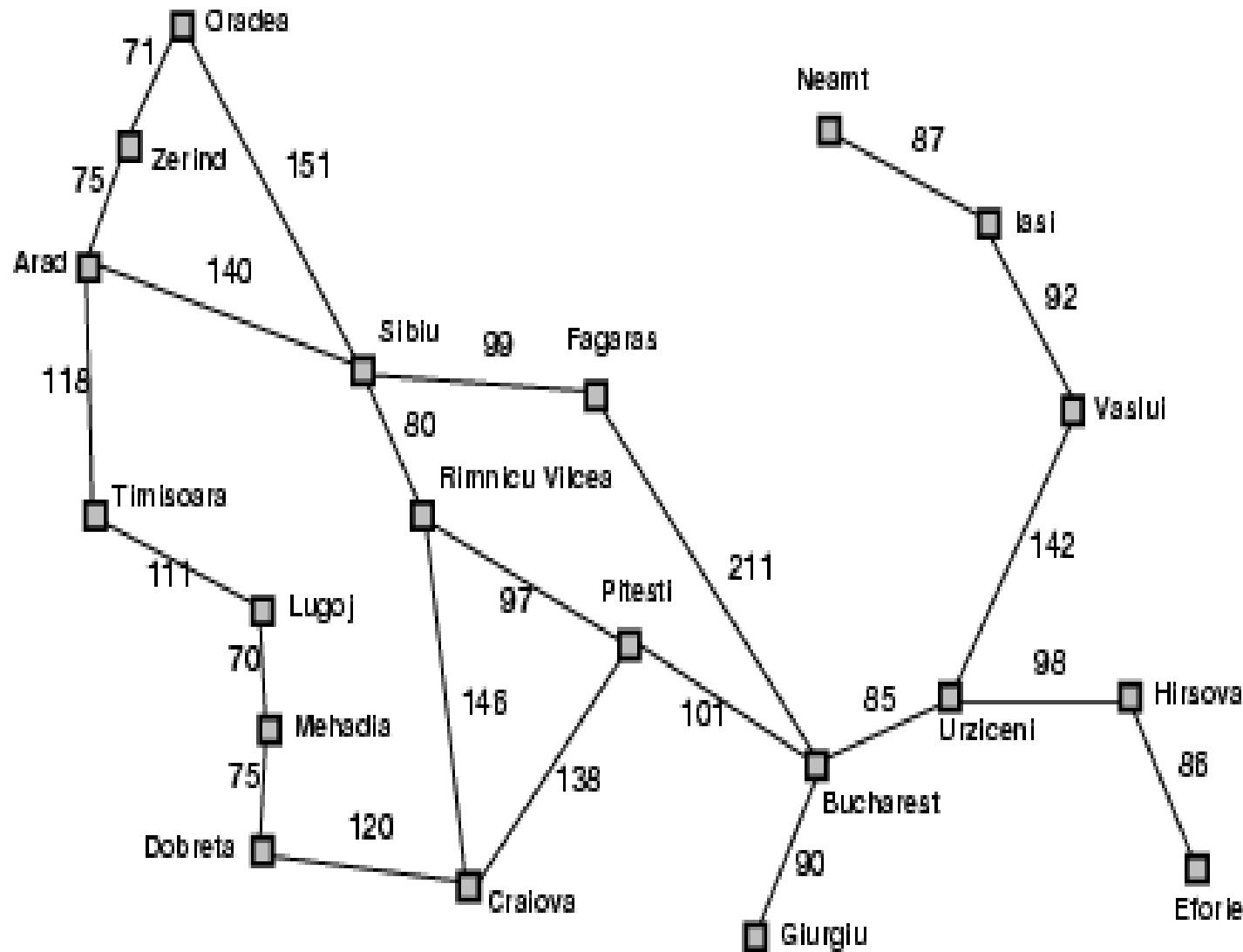
Evaluation function $f(n) = h(n)$ (**heuristic**)

= estimate of cost from n to *goal*.

e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest.

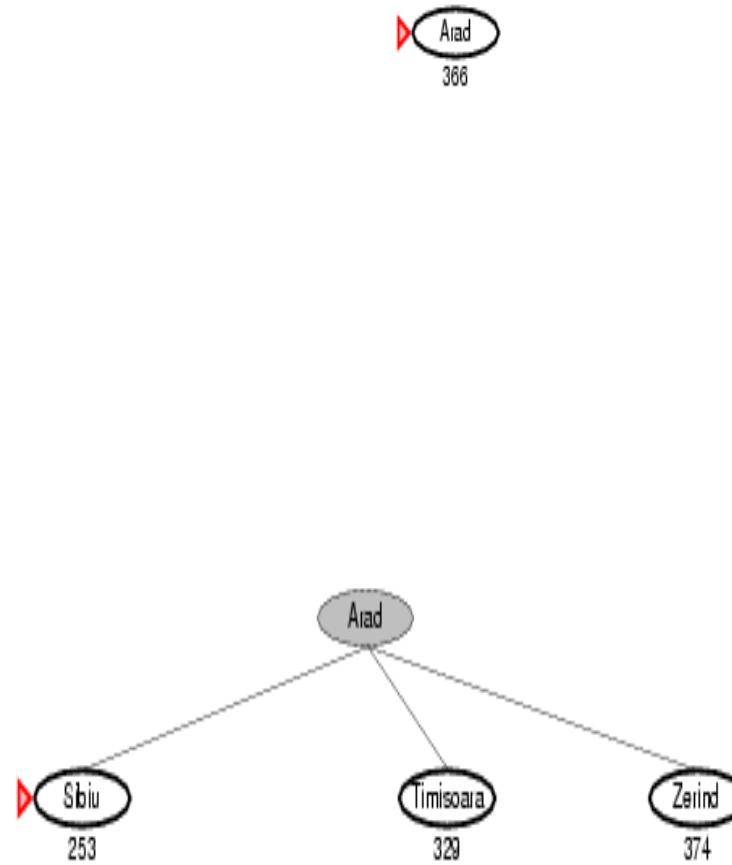
Greedy best-first search expands the node that **appears** to be closest to goal

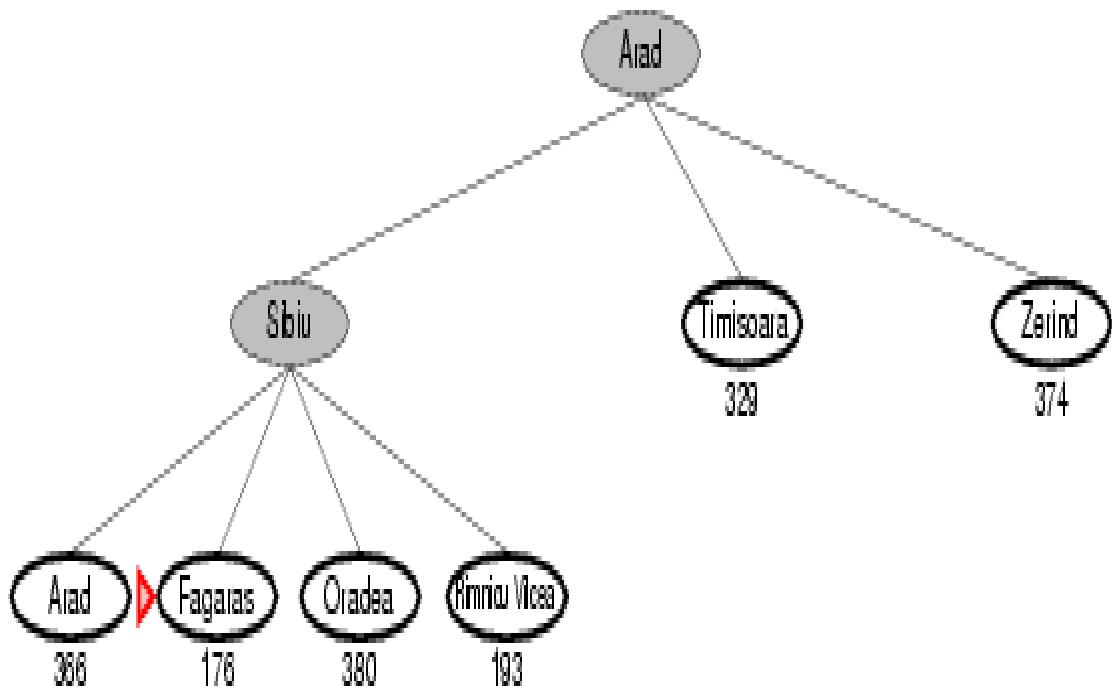
Romania with step costs in km

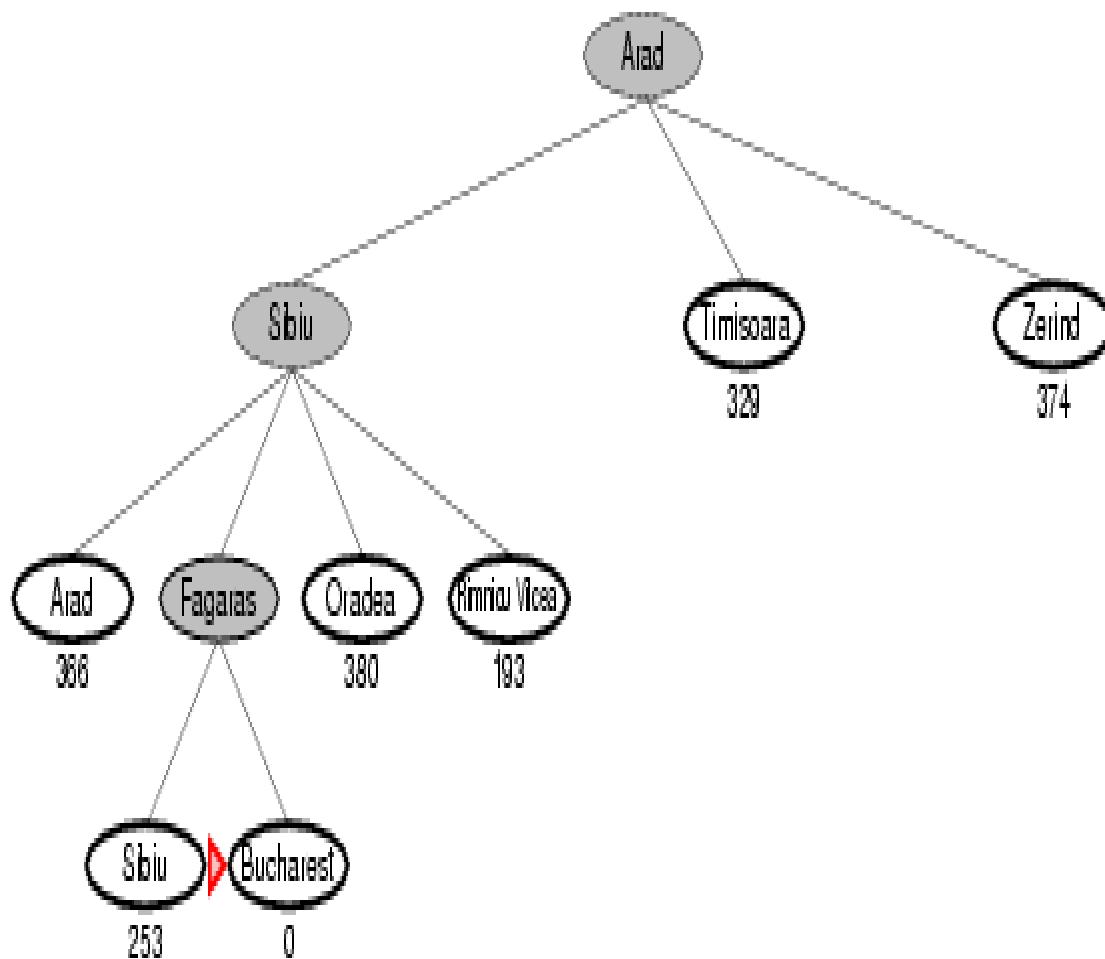


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example







Properties:

Complete? No – can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Time? $O(b^m)$, but a good heuristic can give dramatic improvement

Space? $O(b^m)$ -- keeps all nodes in memory

Optimal? No

A* search:

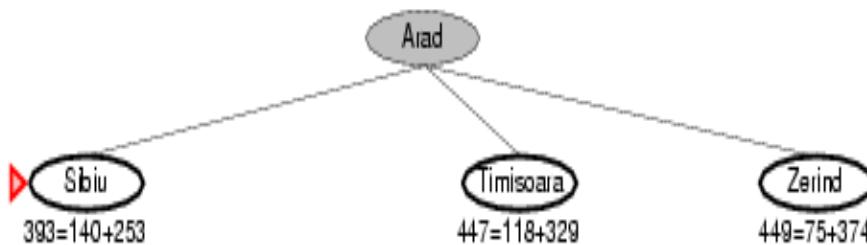
Idea: avoid expanding paths that are already expensive.

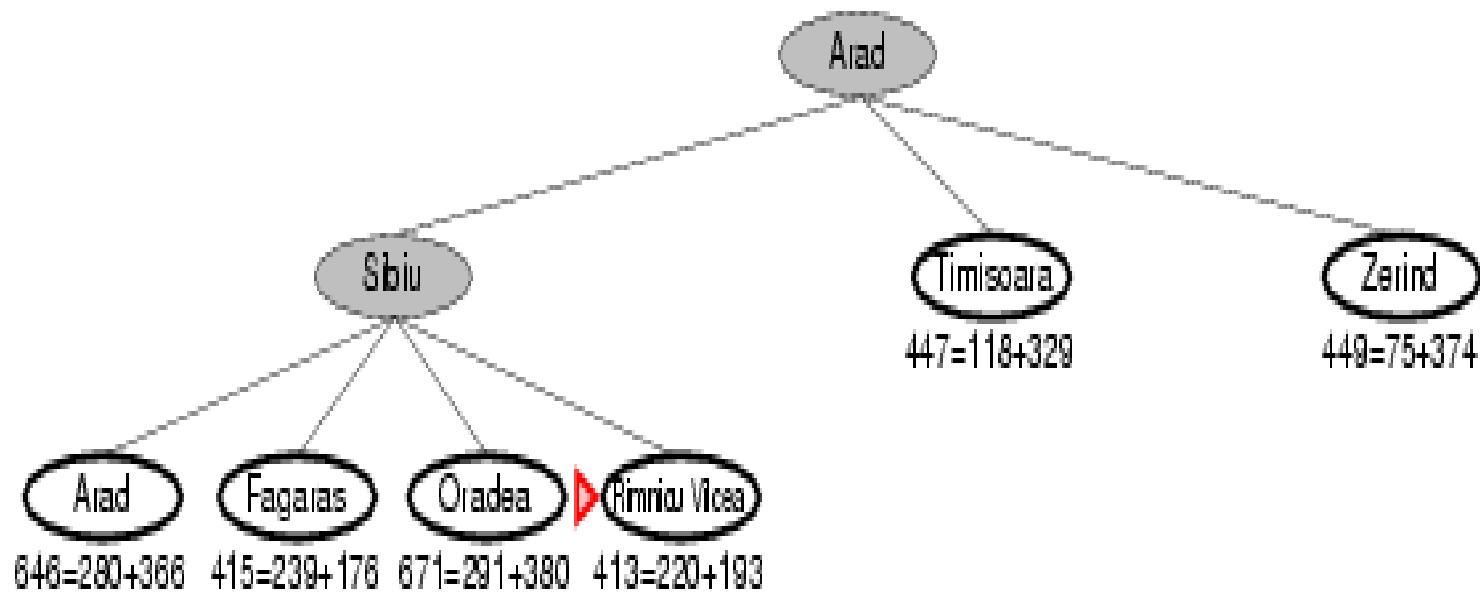
Evaluation function $f(n) = g(n) + h(n)$.

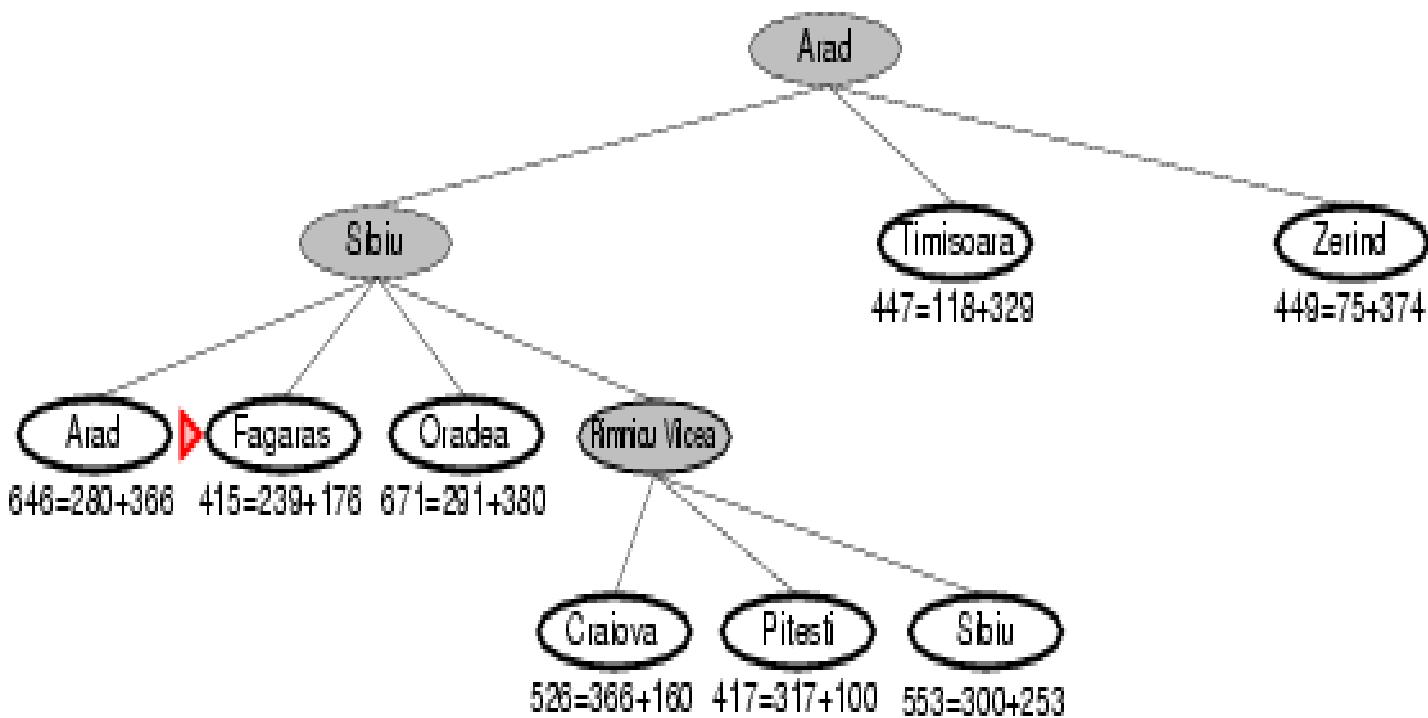
$g(n)$ = cost so far to reach n .

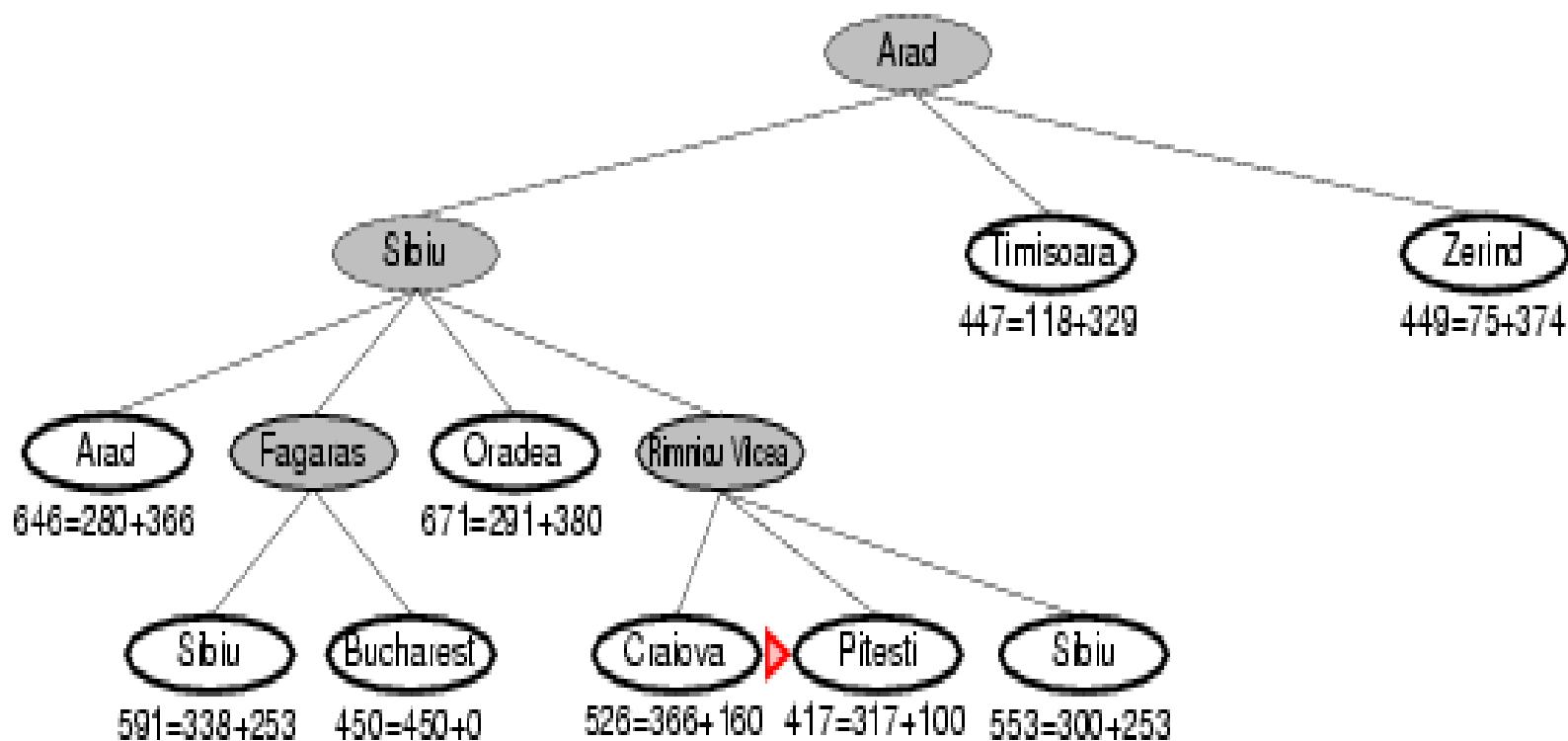
$h(n)$ = estimated cost from n to goal.

$f(n)$ = estimated total cost of path through n to goal.









Admissible heuristics:

A heuristic $h(n)$ is **admissible** if for every node n ,

$h(n) \leq h^*(n)$, where $h^*(n)$ is the **true** cost to reach the goal state from n .

An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is **optimistic**.

Example: $h_{SLD}(n)$ (never overestimates the actual road distance).

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

Properties:

Complete- Yes.

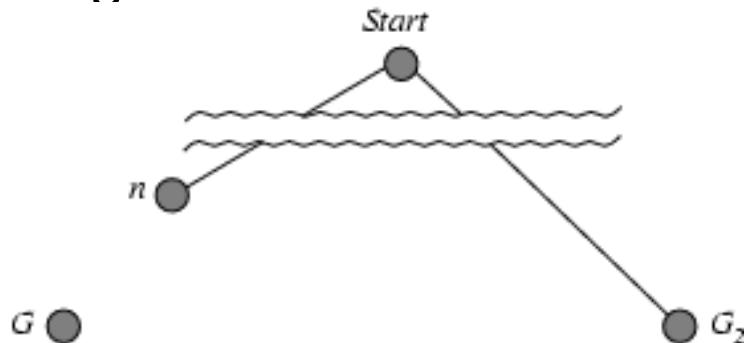
Time- Exponential.

Space-Keeps all nodes in memory.

Optimal- Yes.

Optimality of A* (proof):

Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G .



$$f(G_2) > f(G) \quad \text{from above}$$

$$h(n) \leq h^*(n) \quad \text{since } h \text{ is admissible}$$

$$g(n) + h(n) \leq g(n) + h^*(n)$$

$$f(n) \leq f(G)$$

Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion.

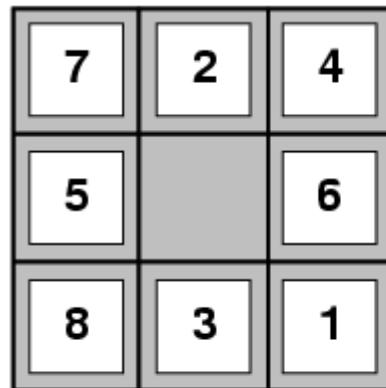
Admissible heuristics:

E.g., for the 8-puzzle:

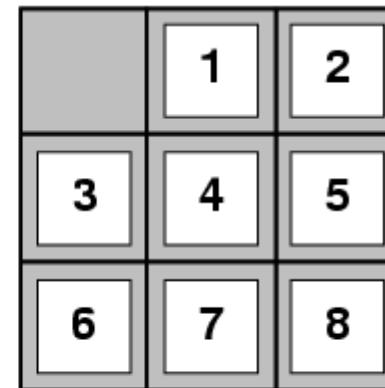
$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

$$\underline{h_1(S) = ?} \quad 8$$

$$\underline{h_2(S) = ?} \quad 3+1+2+2+2+3+3+2 = 18$$

Dominance: If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 **dominates** h_1
 h_2 is better for search.

Properties of Heuristic Algorithms:

Admissibility: Algorithm A is admissible if it is guaranteed to return an optimal solution when one exists.

Completeness: Algorithm A is complete if it is guaranteed to return a solution when one exists.

Dominance: A1 dominates A2 if the heuristic function of A1 is better than that of A2.

Optimality: Algorithm A is optimal over a class of algorithms if A dominates all members of the class.

Constraints Satisfaction: It is a heuristics based technique. Constraints satisfaction is a process of finding a solution to set of constraints that impose condition that the variable must satisfy. A solution is therefore set of value for the variable that satisfy all the constraints .

How to solve problem:

1. We need to analyzing the problem.
2. We need to derive the constraints given in problem.
3. We need to derive the solution of given constraints.
4. We need to find whether find a good state if we not reach to goal state then we need to Guess, that has add new constraints and again solve find solution of problem.

- 1.Crypt-arithmetic problem
2. N-Queen Problem
3. Map coloring problem
4. Crossword Puzzle

1.Crypt-arithmetic problem

Ex.	1.TWO + TWO=FOUR	2. SEND+MORE=MONEY
	3. NO+NO=YES	4. HERE+SHE= COMES
	5.TOM+NAG=GOAT	ODD+ODD=EVEN

Constraints:

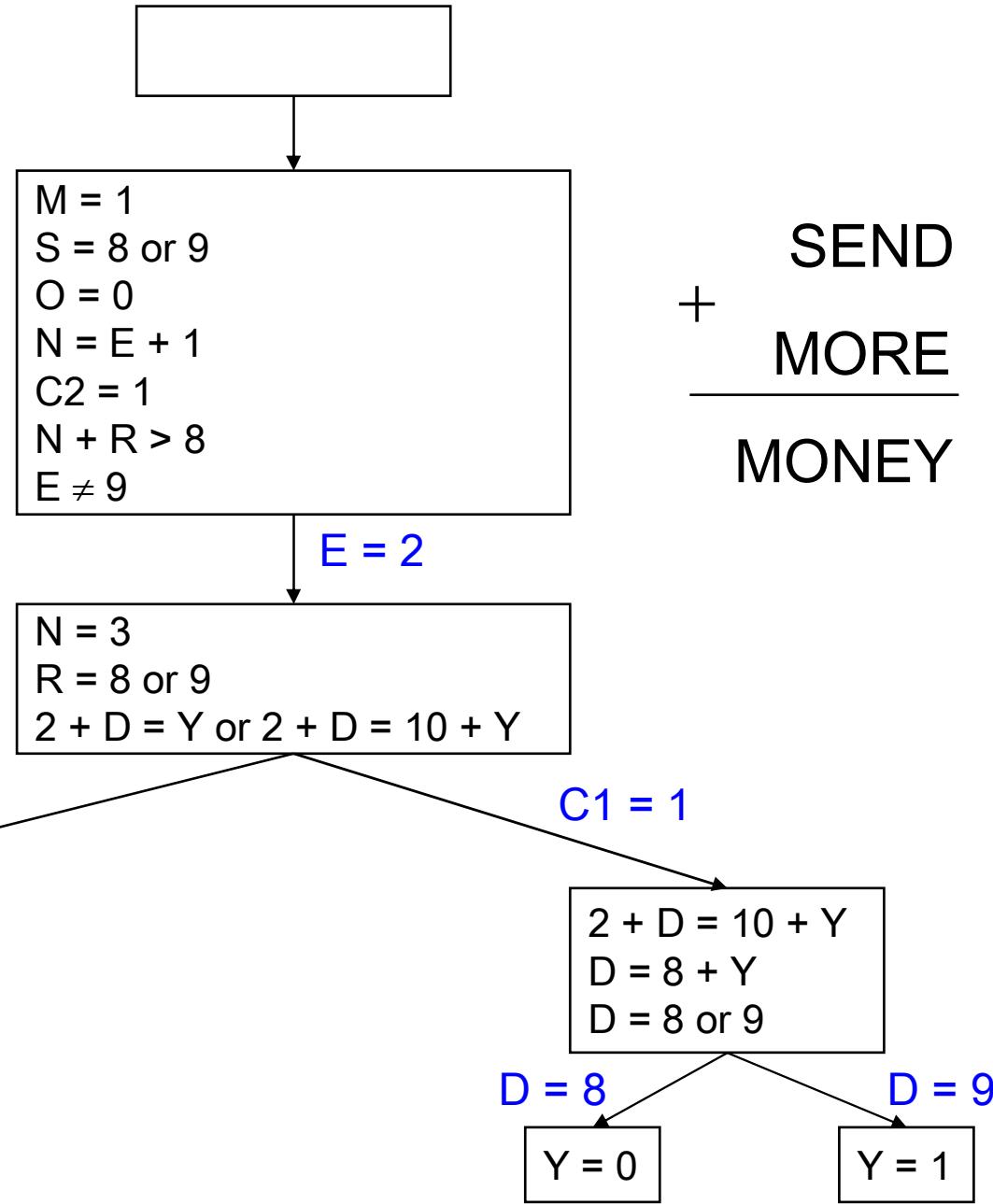
1. Assign some unique digit to each letter
2. We can't assign deferent digit of same letter
- 3.No two letter have same digit

Two-step process:

1. Constraints are discovered and propagated as far as possible.
2. If there is still not a solution, then search begins, adding new constraints.

Initial state:

- No two letters have the same value.
- The sum of the digits must be as shown.



Two kinds of rules:

1. Rules that define valid constraint propagation.
2. Rules that suggest guesses when necessary.

A map coloring problem: We are given a map, i.e. a planar graph, and we are told to color it using k colors, so that no two neighboring countries have the same color.

You have to color a planner map using only four colors .In such a way no two adjacent regions have the color.

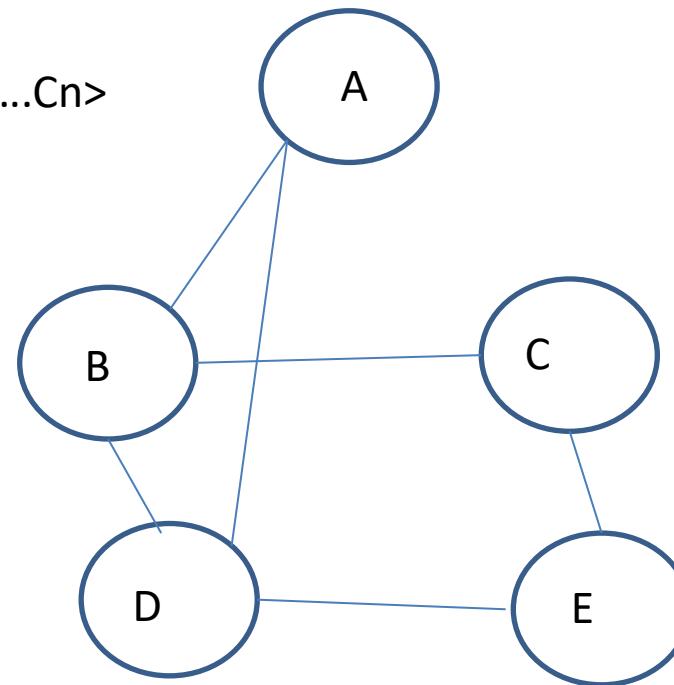
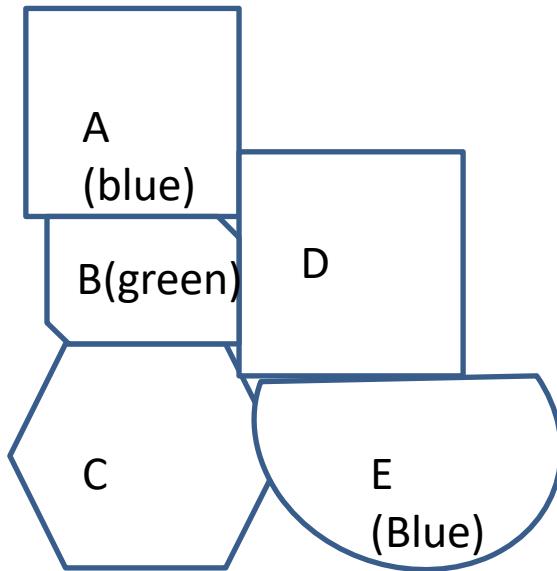
This map is represented by a graph .each regions corresponding to vertex of a graph .If two regions are adjacent ,there is an edge connecting the corresponding vertices.

The vertices are $< v_1, v_2, v_3, \dots, v_n >$

The color are represented by $< c_1, c_2, c_3, \dots, c_n >$

A state represented as a N-tuple.

Ex.



Particular state of graph : Representation of current state
{ blue, green ,X,X blue)

Initial state: { x,x,x,x}

Goal state :

if X_i and X_j are adjacent
Color(i) not equal to color(j).

All regions have colored.

What are different operation apply the graph change the color of a state i to c ...

->change (i,c)

->change(2,yellow)-> B is color(where B is state)

-> change(3,green) -> C is color

Adversarial Search: A framework for formulating a multi-person game as a search problem. We will consider games in which the players alternate making moves and try respectively to maximize and minimize a scoring function (also called utility function). we will only consider games with the following two properties:

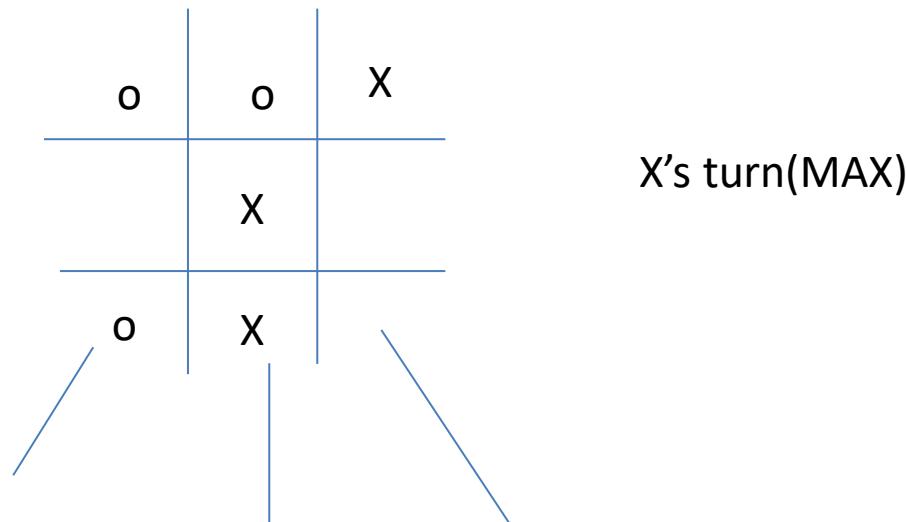
- **Two player** - we do not deal with coalitions, etc.
- **Zero sum** - one player's win is the other's loss; there are no cooperative victories

Game tree:

The above category of games can be represented as a tree where the nodes represent the current state of the game and the arcs represent the moves. The game tree consists of all possible moves for the current players starting at the root and all possible moves for the next player as the children of these nodes, and so forth, as far into the future of the game as desired. Each individual move by one player is called a "ply". The leaves of the game tree represent terminal positions as one where the outcome of the game is clear (a win, a loss, a draw, a payoff). Each terminal position has a score. High scores are good for one of the player, called the MAX player. The other player, called MIN player, tries to minimize the score. For example, we may associate 1 with a win, 0 with a draw and -1 with a loss for MAX.

Ex. tic-tac-toe

In tic-tac-toe, as in many turn based, adversarial games, the game can end in one of only three ways: win, lose, draw. We may then assign the utility, 1, to terminal states in which the game is decided in the agent's favor, -1 to those states in which the agent has lost the game, and 0 to the states in which a draw has been reached.



- Above is a section of a game tree for tic tac toe. Each node represents a board position, and the children of each node are the legal moves from that position

Minimax Algorithm

How do we compute our optimal move? We will assume that the opponent is rational; that is, the opponent can compute moves just as well as we can, and the opponent will always choose the optimal move with the assumption that we, too, will play perfectly. One algorithm for computing the best move is the minimax algorithm:

```
minimax(player,board)
    if(game over in current board position)
        return winner
        children = all legal moves for player from this board
    if(max's turn)
        return maximal score of calling minimax on all the children
    else (min's turn)
        return minimal score of calling minimax on all the children
```

If the game is over in the given position, then there is nothing to compute; minimax will simply return the score of the board. Otherwise, minimax will go through each possible child, and (by recursively calling itself) evaluate each possible move. Then, the best possible move will be chosen, where ‘best’ is the move leading to the board with the most positive score for player 1, and the board with the most negative score for player 2.

Heuristic evaluation function:

-An evaluation function estimate how good the current board configuration is for a player

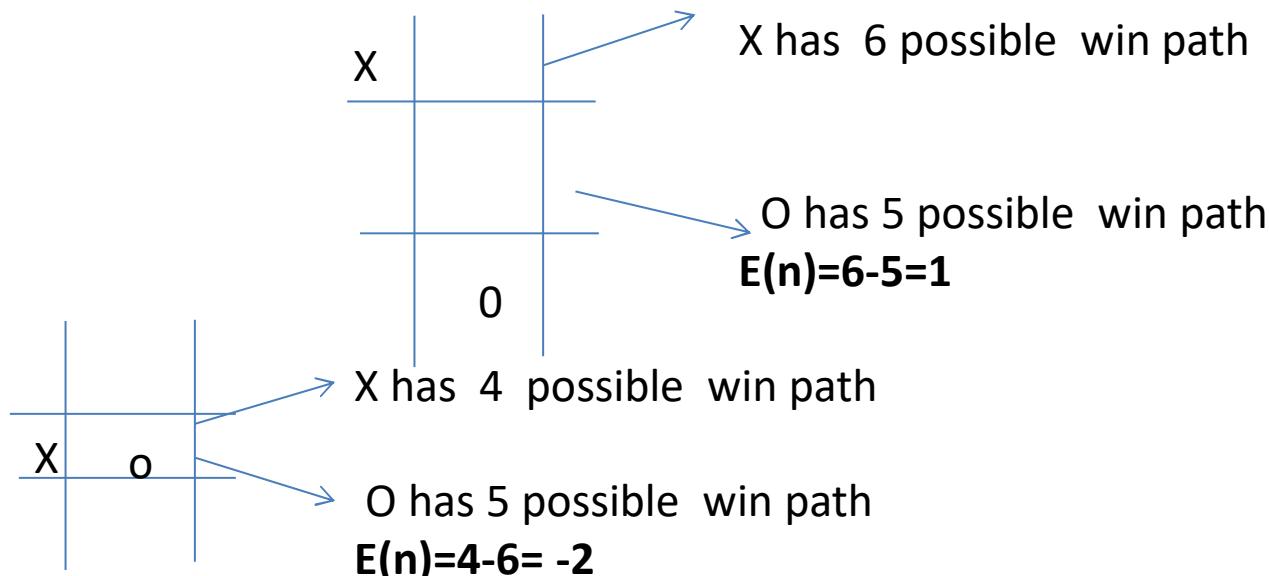
$$E(n) = M(n) - O(n) \quad [\text{subtract the opponents score from the players}]$$

$M(n)$ – total of MAX player possible wining line

$O(n)$ – total of Opponent possible wining line

$E(n)$ – total evaluation for state n

- values range from $-\infty$ (loss) to $+\infty$ (win) or $[-1, +1]$



Properties:

Complete	-	yes(if tree is infine)
optimal	-	yes(against an optimal opponent)
Time complexity-		$O(b^m)$ or $O(b^m)$
space -	$O(b^*m)$	

m – maximum depth of the tree; b – legal moves

MiniMax is two player game, the player are referred to as **MAX**(the player) and **MIN**(the opponent).

MAX is the player trying to maximize its score and **MIN** is the opponent trying to minimize **MAX's** score.

Optimal strategy for MINIMAX: Design to find optimal strategy for max and find best move.

Generate the whole game tree to leaves. Apply evaluated function to leaves.

Back –up values from leaves toward the root.

- a MAX node compute the max of its child values.
- a MIN node compute the min of its child values.

4. When value reaches the root: Choose max value and the corresponding move.

Component: -Initial ,successor function, terminal state, utility function

Note: Higher utility value good for MAX and lower value bad for MAX

Alpha–beta pruning:

The minimax algorithm is a way of finding an optimal move in a two player game. *Alpha-beta pruning* is a way of finding the optimal minimax solution while avoiding searching sub-trees of moves which won't be selected. In the search tree for a two-player game, there are two kinds of nodes, nodes representing *your* moves and nodes representing *your opponent's* moves.

MAX nodes: The goal at a MAX node is to maximize the value of the sub-tree rooted at that node. To do this, a MAX node

MIN nodes: The goal at a MIN node is to minimize the value of the sub-tree rooted at that node. To do this, a MIN node chooses the child with the least (smallest) value, and that becomes the value of the MIN node.

Alpha-beta pruning gets its name from two bounds that are passed along during the calculation, which restrict the set of possible solutions based on the portion of the search tree

β - Beta is the *minimum upper bound* of possible solutions

α -Alpha is the *maximum lower bound* of possible solutions

Thus, when any new node is being considered as a possible path to the solution, it can only work if:

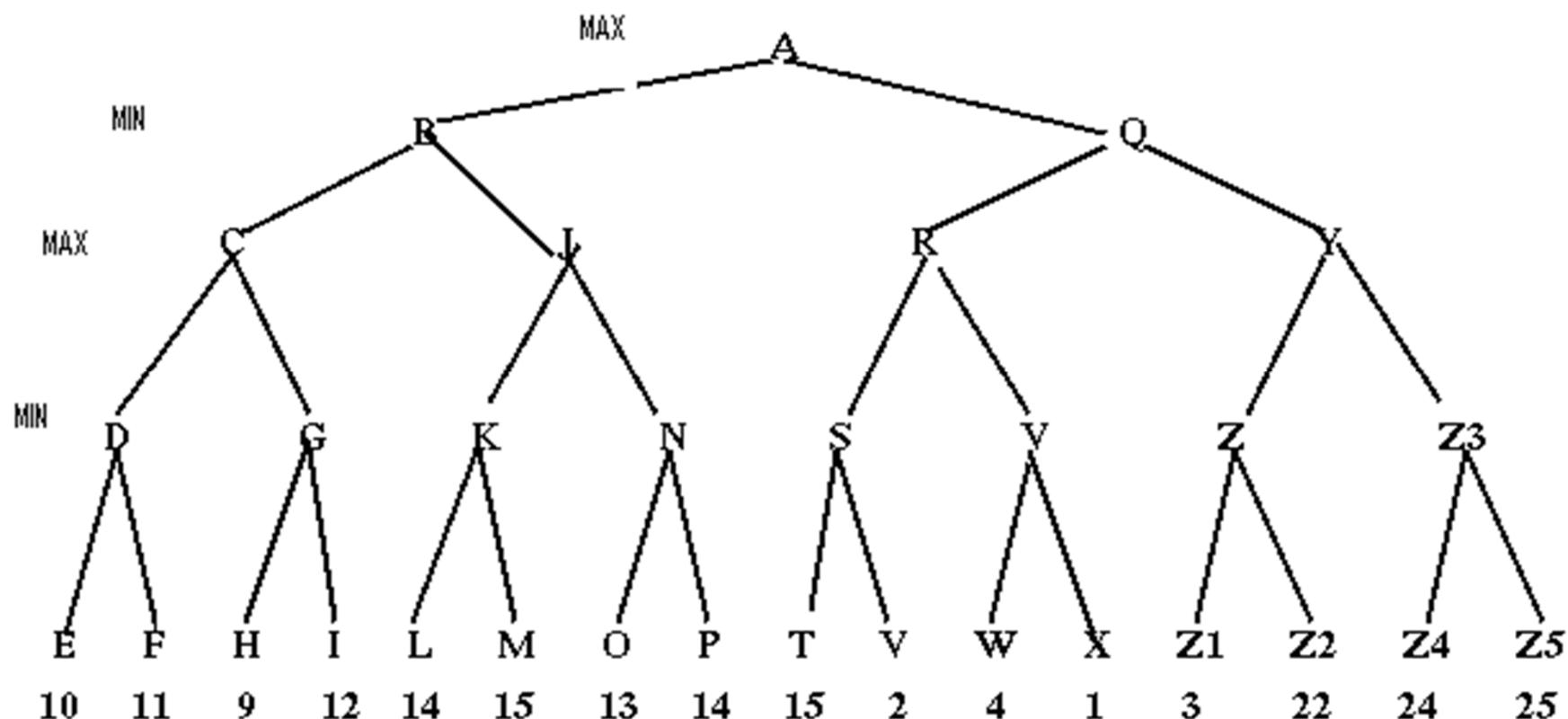
$$\alpha \leq N \leq \beta$$

where N is the current estimate of the value of the node.

Example: As for upper and lower bounds, all you know is that it's a number less than infinity and greater than negative infinity. Thus, here's what the initial situation looks like: Both are equivalent to



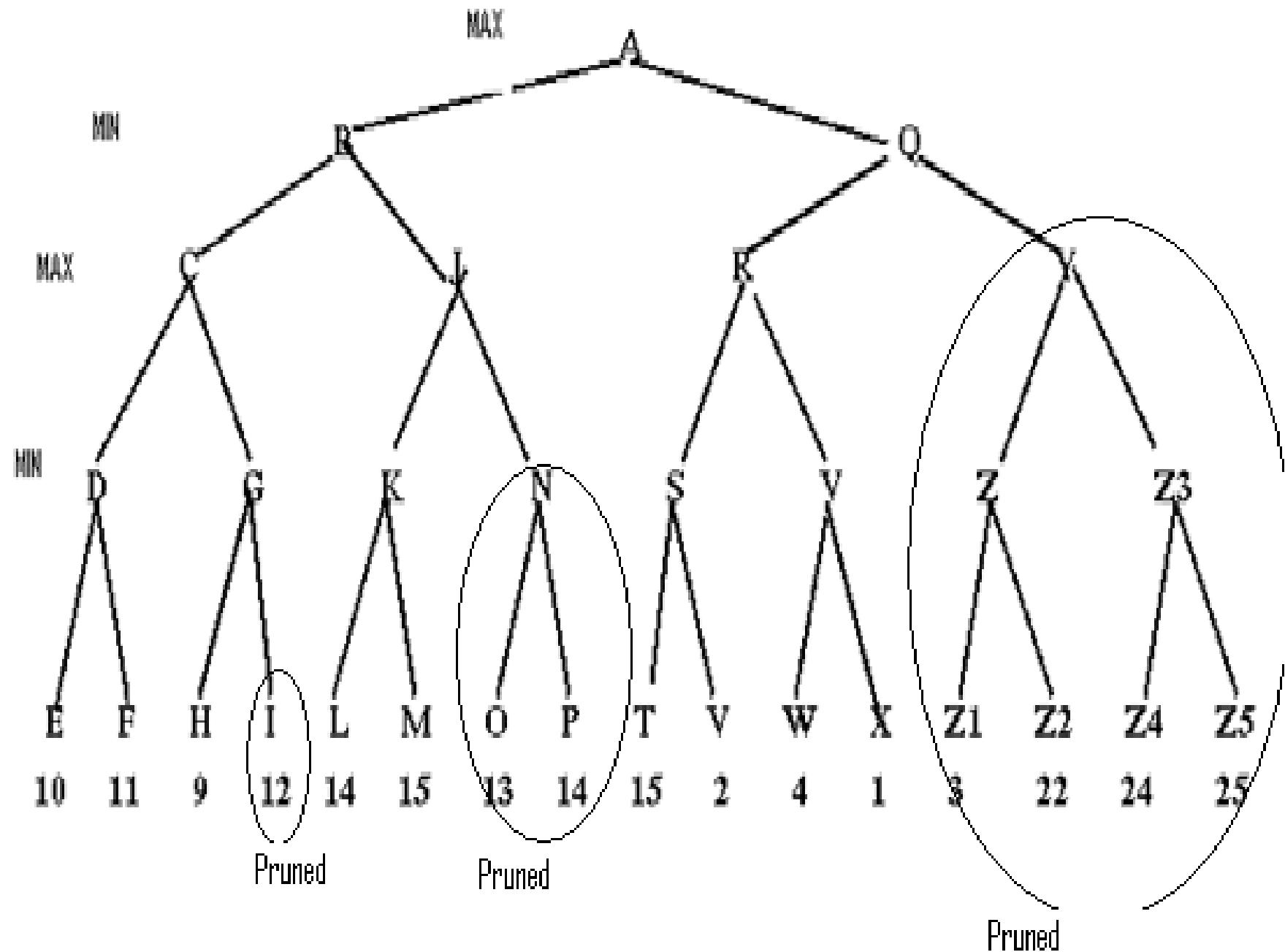
alpha beta pruning with the following problem:



Solution: Alpha—beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves. The general principle is this: consider a node n somewhere in the tree, such that Player has a choice of moving to that node. If Player has a better choice *in*, either at the parent node of n or at any choice point further up, then n *will never be reached in actual play*. So once we have found out enough about n (by examining some of its descendants) to reach this conclusion, we can prune it.

At last min level when E comes then $D \leq 10$, F comes as 11 so now D is 10. Now at MAX level it is clear that C will at least be 10. Now at MIN level H comes as 9 so it is confirmed that G will be less than equal to 9. Now at above MAX level 10 has already been achieved so why to go for a value which is less than 9 So I will be pruned.

- So C is confirmed as 10. So at above MIN level B will at most be 10. Now at lower min level when L comes as 14, K is at most 14, after confirming M as 15 K is finalized as 14. So now at next MAX level J is at least 14. But at above MIN level B has already got 10 then there is no need to explore N. So it is pruned.
- So B is confirmed as 10. Now A is at least 10. Next T comes as 15. So S is at most 15. After getting U as 2, it is confirmed as 2. So next level R is at least 2. Now W comes as 4 So V is at most 4, So R could be 4 so X will be checked. After getting X as 1 V is confirmed as 1. So R is confirmed as 2. It means Q is at most 2. So No need to explore Y. it will be pruned.



NEURAL NETWORKS

□ Neural network was inspired by the design and functioning of human brain and components.

□ *Definition:*

□ Information processing model that is inspired by the way biological nervous system (i.e) the brain, process information.

□ ANN is composed of large number of highly interconnected processing elements(neurons) working in unison to solve problems.

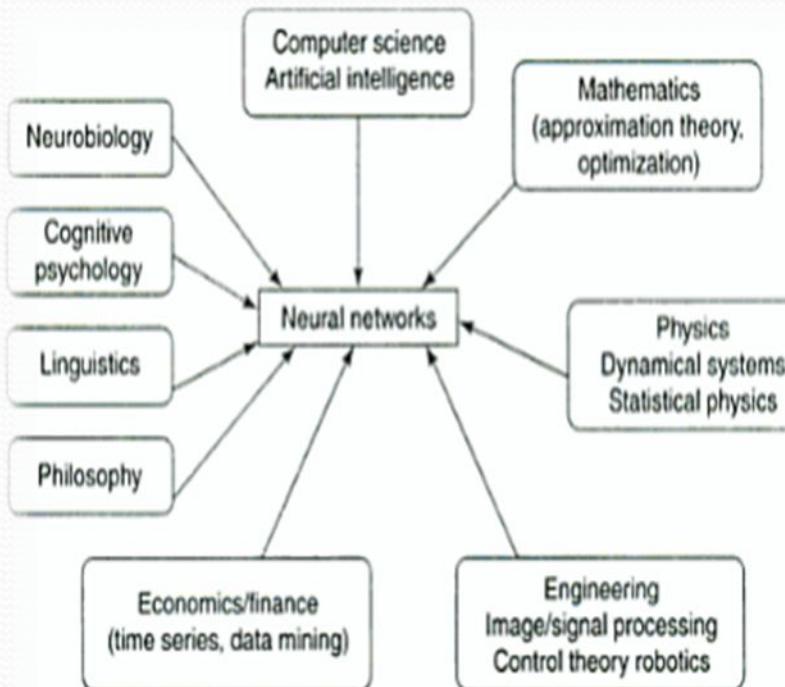
□ It is configured for special application such as pattern recognition and data classification through a learning process.

□ 85-90% accurate.

Advantages of Neural Networks

- A Neural Network can be an “*expert*” in analyzing the category of information given to it.
- Answers “what-if” questions
- Adaptive learning
 - Ability to learn how to do tasks based on the data given for training or initial experience.
- Self organization
 - Creates its own organization or representation of information it receives during learning time.
- Real time operation
 - Computations can be carried out in parallel.
- Fault tolerance via redundant information coding
 - Partial destruction of neural network cause degradation of performance.
 - In some cases, it can be retained even after major network damage.
- In future, it can also used to give spoken words as instructions for machine.

- This figure shows the multi disciplinary point of view of Neural Networks



Application Scope of Neural Networks

- Air traffic control
- Animal behavior
- Appraisal and valuation of property, etc.,
- Betting on horse races, stock markets
- Criminal sentencing
- Complex physical and chemical process
- Data mining, cleaning and validation
- Direct mail advertisers
- Echo patterns
- Economic modeling
- Employee hiring
- Expert consultatants
- Fraud detection
- Hand writing and typewriting
- Lake water levels
- Machinery controls
- Medical diagnosis
- Music composition
- Photos and finger prints
- Recipes and chemical formulation
- Traffic flows
- Weather prediction

Fuzzy Logic

- Lofti Zadeh, Professor at University of California.
- *An organized method for dealing with imprecise data*
- Fuzzy logic includes 0 and 1 as extreme cases of truth (or "the state of matters" or "fact") but also includes the various states of truth in between so that, for example, the result of a comparison between two things could be not "tall" or "short" but ".38 of tallness."
- Allows partial membership
- Implemented in small, embedded micro controllers to large , networked, multichannel PC or work station.
- Can be implemented in hardware, software or in both.
- It mimics how a person would make decisions.

Genetic algorithm

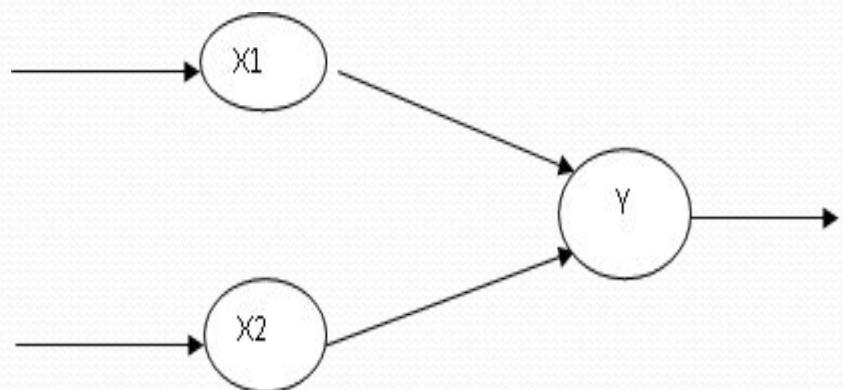
- How genes of parents combine to form those of their children.
- Create an initial population of individuals representing possible solutions to solve a problem
- Individual characters determine whether they are less or more fit to the population
- The more fit members will take high probability.
- It is very effective in ***finding optimal or near optimal solutions.***
- Generate and test strategy.
- Differ from normal optimization and search procedures in:
 - Work with coding of the parameter set
 - Work with multiple points
 - Search via sampling(a blind search)
 - Search using stochastic operators
- In business, scientific and engineering circles, etc.,

Artificial Neural Network : An Introduction

- Resembles the characteristic of biological neural network.
- **Nodes** – interconnected processing elements (units or neurons)
- Neuron is connected to other by a ***connection link***.
- Each connection link is associated with ***weight*** which has information about the input signal.
- ANN processing elements are called as ***neurons or artificial neurons*** , since they have the capability to model networks of original neurons as found in brain.
- Internal state of neuron is called ***activation or activity level*** of neuron, which is the function of the inputs the neurons receives.
- Neuron can send only one signal at a time.

Basic Operation of a Neural Net

- X1 and X2 – input neurons.
 - Y- output neuron
 - Weighted interconnection links- W1 and W2.
 - Net input calculation is :
-
- $Y_{in} = -x_1w_1 + x_2w_2$
 - Out $y = f(Y_{in})$

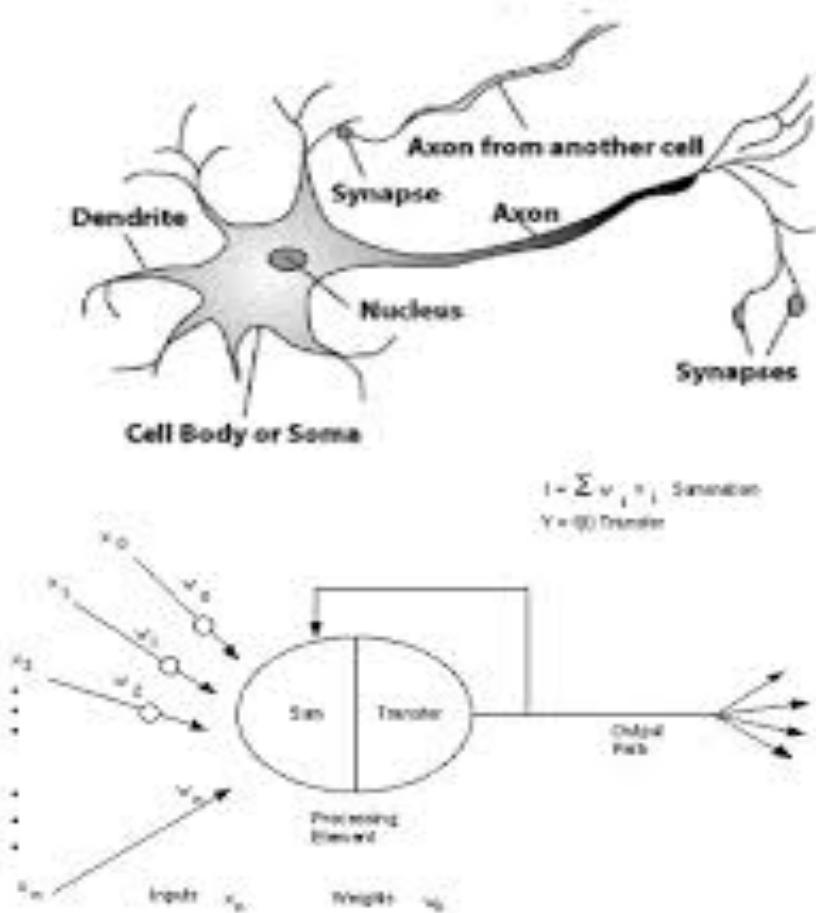


Contd....

- The function to be applied over the net input is called *activation function*.
- Weight involved in ANN is equal to the slope of linear straight line ($y=mx$).

Biological Neural Network

- Has three main parts
 - Soma or cell body-where cell nucleus is located
 - Dendrites-where the nerve is connected to the cell body
 - Axon-which carries the impulses of the neuron
- Electric impulse is passed between synapse and dendrites.
- Synapse- Axon split into strands and strands terminates into small bulb like organs called as synapse.



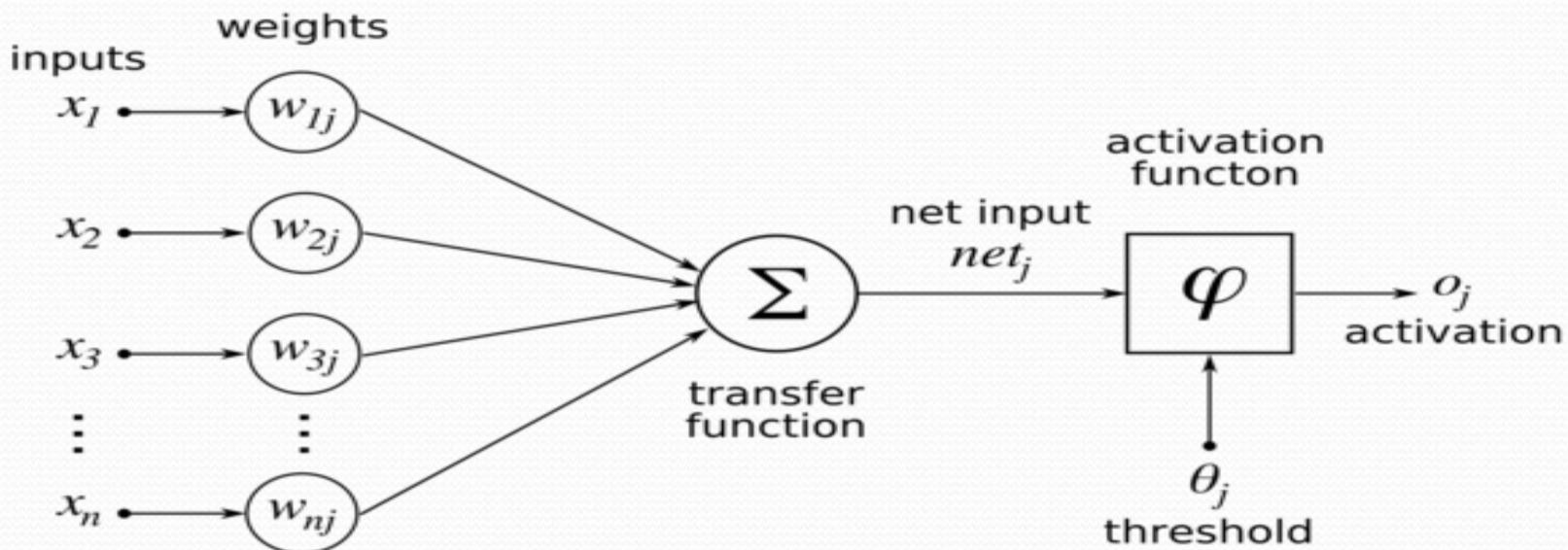
- It is a chemical process which results in increase /decrease in the electric potential inside the body of the receiving cell.
- If the electric potential reaches a threshold value, receiving cell fires & pulse / action potential of fixed strength and duration is sent through the axon to synaptic junction of the cell.
- After that, cell has to wait for a period called ***refractory period***.
- In this model net input is calculated by

$$Y_{in} = x_1w_1 + x_2w_2 + \dots + x_nw_n$$

$$\sum_{i=1}^n x_i w_i$$

Terminology Relation Between Biological And Artificial Neuron

Biological Neuron	Artificial Neuron
Cell	Neuron
Dendrites	Weights or interconnections
Soma	Net input
Axon	Output



Brain Vs computer

Term	Brain	Computer
Speed	Execution time is few milliseconds	Execution time is few nano seconds
Processing	Perform massive parallel operations simultaneously	Perform several parallel operations simultaneously. It is faster than the biological neuron
Size and complexity	Number of Neuron is 10^{11} and number of interconnections is 10^{15} . So complexity of brain is higher than computer	It depends on the chosen application and network designer.
Storage capacity	i) Information is stored in interconnections or in synapse strength. ii) New information is stored without destroying old one. iii) Sometimes fails to recollect information	i) Stored in continuous memory location. ii) Overloading may destroy older locations. iii) Can be easily retrieved

Contd....

Tolerance	<ul style="list-style-type: none">i) Fault tolerantii) Store and retrieve information even interconnections failsiii) Accept redundancies	<ul style="list-style-type: none">i) No fault toleranceii) Information corrupted if the network connections disconnected.iii) No redundancies
Control mechanism	Depends on active chemicals and neuron connections are strong or weak	CPU Control mechanism is very simple

Characteristics of ANN:

- Neurally implemented mathematical model
- Large number of processing elements called neurons exists here.
- Interconnections with weighted linkage hold informative knowledge.
- Input signals arrive at processing elements through connections and connecting weights.
- Processing elements can learn, recall and generalize from the given data.
- Computational power is determined by the collective behavior of neurons.
 - *ANN is a connection models, parallel distributed processing models, self-organizing systems, neuro-computing systems and neuro morphic system.*

Evolution of neural networks

Year	Neural network	Designer	Description
1943	McCulloch and Pitts neuron	McCulloch and Pitts	Arrangement of neurons is combination of logic gate. Unique feature is thresh hold
1949	Hebb network	Hebb	If two neurons are active, then their connection strengths should be increased.
1958,1959,1962,1988 ,1960	Perceptron Adaline	Frank Rosenblatt, Block, Minsky and Papert Widrow and Hoff	Weights are adjusted to reduce the difference between the net input to the output unit and the desired output

Contd....

1972	Kohonen self-organizing feature map	Kohonen	Inputs are clustered to obtain a fired output neuron.
1982, 1984, 1985, 1986, 1987	Hopfield network	John Hopfield and Tank	Based on fixed weights. Can act as associative memory nets
1986	Back propagation network	Rumelhart, Hinton and Williams	i) Multilayered ii) Error propagated backward from output to the hidden units

Contd..

1988	Counter propagation network	Grossberg	Similar to kohonen network
1987-1990	Adaptive resonance Theory(ART)	Carpenter and Grossberg	Designed for binary and analog inputs.
1988	Radial basis function network	Broomhead and Lowe	Resemble back propagation network , but activation function used is Gaussian function
1988	Neo cognitron	Fukushima	For character recogniton.

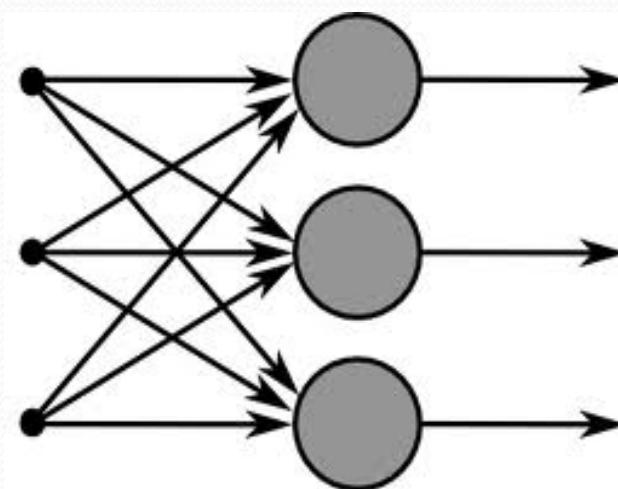
Basic models of ann

- Models are based on three entities
 - The model's synaptic interconnections.
 - The training or learning rules adopted for updating and adjusting the connection weights.
 - Their activation functions
- An ANN consists of a set of highly interconnected processing elements such that each processing element is found to be connected through weights to other processing elements or to itself.

- The *arrangement of neurons to form layers* and the *connection pattern formed within and between layers* is called the ***network architecture***.
- Five types:
 - Single layer feed forward network
 - Multilayer feed-forward network
 - Single node with its own feedback
 - Single-layer recurrent network
 - Multilayer recurrent network

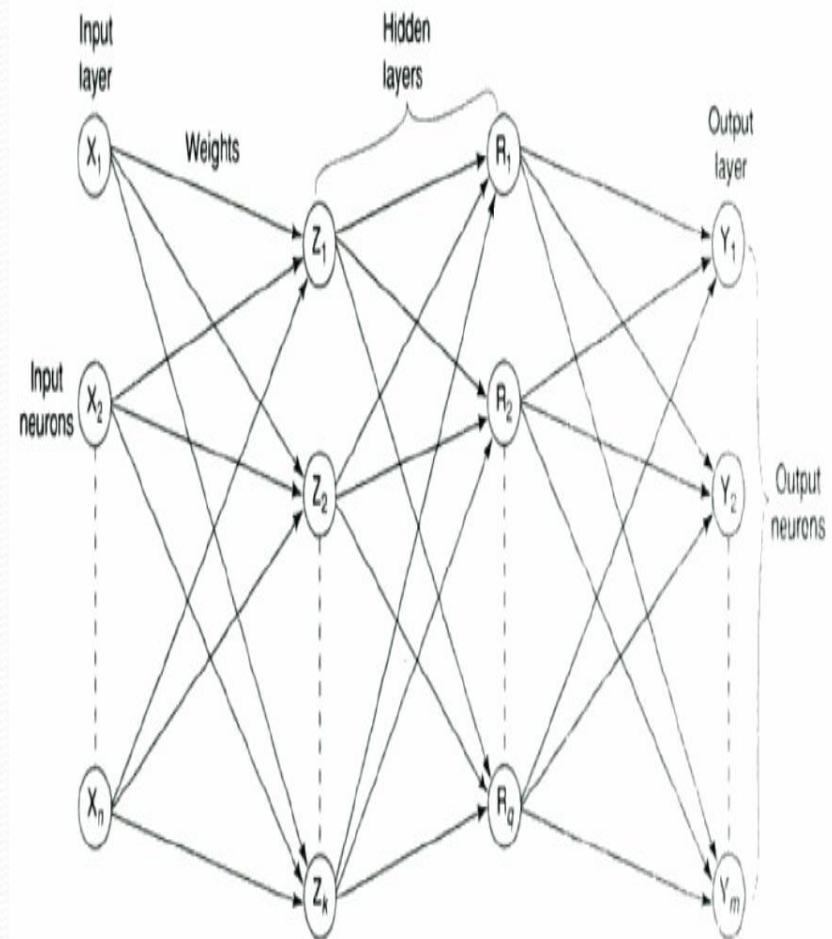
Single layer Feed- Forward Network

- Layer is formed by taking processing elements and combining it with other processing elements.
- Layer implies a stage , going stage by stage
- Input and output are linked with each other
- Inputs are connected to the processing nodes with various weights, resulting in series of outputs one per node.

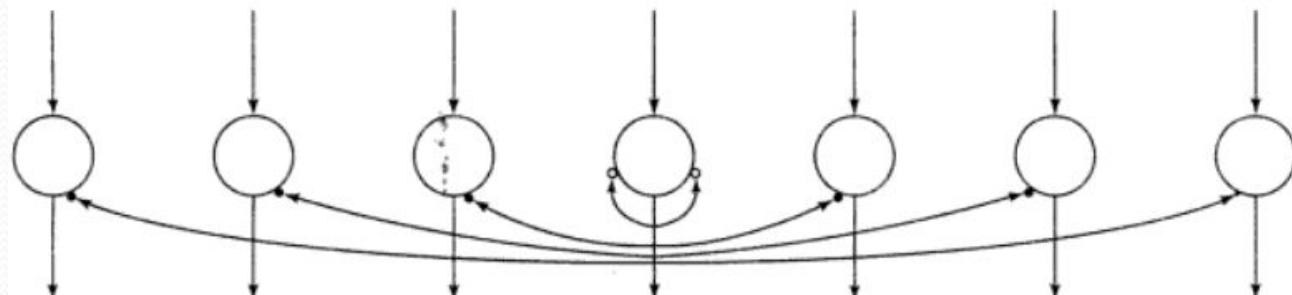
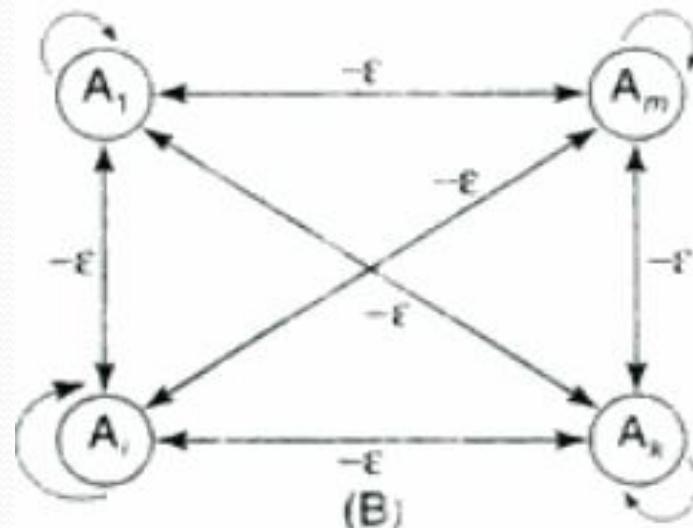


Multilayer feed-forward network

- Formed by the interconnection of several layers.
- Input layer receives input and buffers input signal.
- Output layer generated output.
- Layer between input and output is called *hidden layer*.
- Hidden layer is internal to the network.
- Zero to several hidden layers in a network.
- More the hidden layer, more is the complexity of network, but efficient output is produced.



- *Maxnet* –competitive interconnections having fixed weights.
- *On-center-off-surround/lateral inhibiton structure* – each processing neuron receives two different classes of inputs- “excitatory” input from nearby processing elements & “inhibitory” elements from more distantly located precessing elements. This type of interconnection is shown below



Feed back network

- If no neuron in the output layer is an input to a node in the same layer / proceeding layer – ***feed forward network.***
- If outputs are directed back as input to the processing elements in the same layer/proceeding layer –***feedback network.***
- If the output are directed back to the input of the same layer then it is ***lateral feedback.***
- ***Recurrent networks*** are networks with feedback networks with closed loop.
- Fig 2.8 (A) –simple recurrent neural network having a single neuron with feedback to itself.
- Fig 2.9 – single layer network with feedback from output can be directed to processing element itself or to other processing element/both.

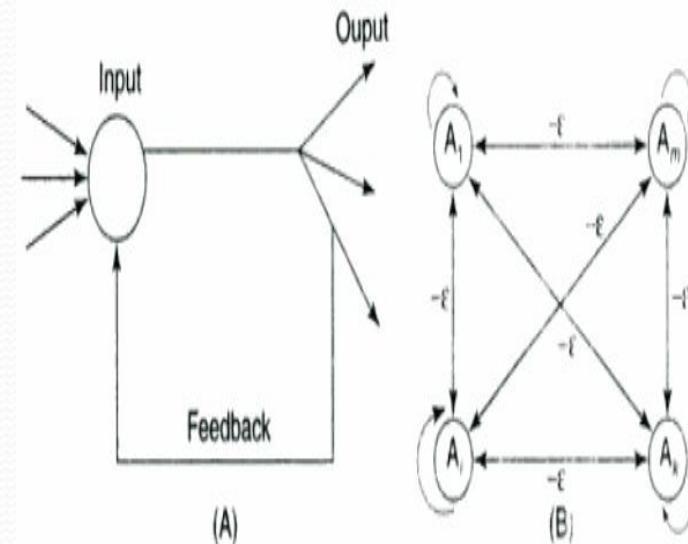


Figure 2.8 (A) Single node with own feedback. (B) Competitive nets.

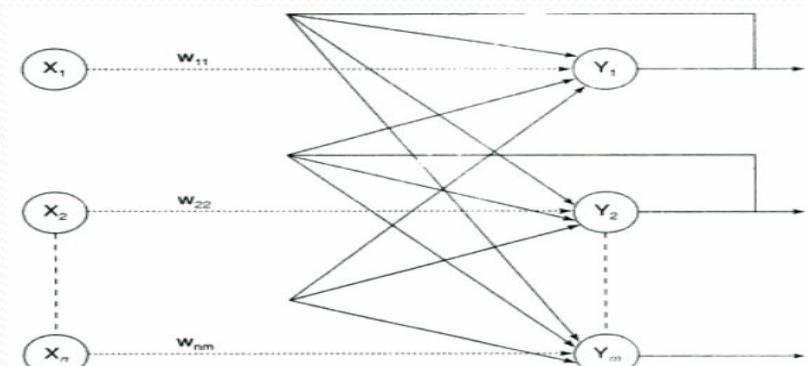
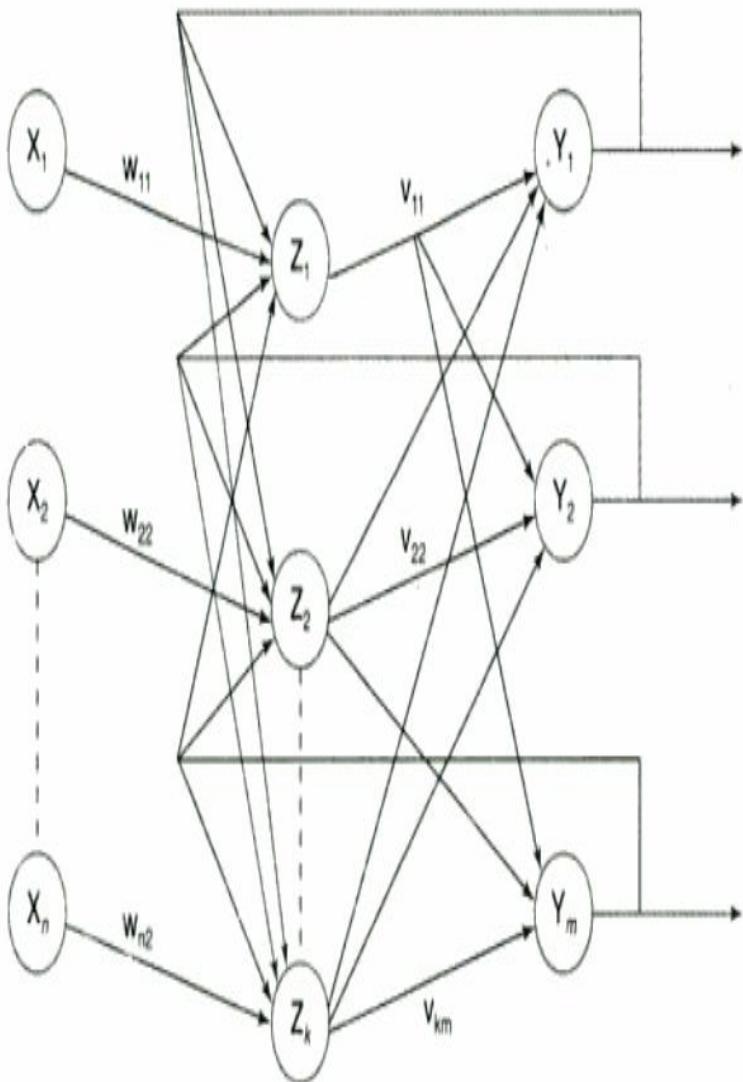


Figure 2.9 Single-layer recurrent network.



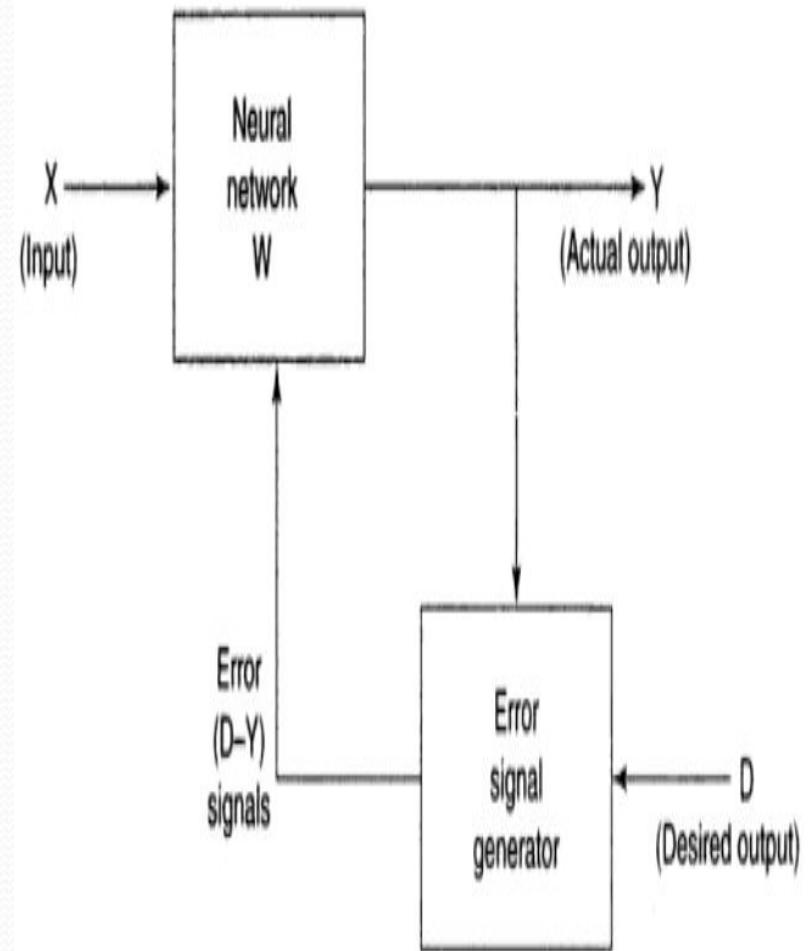
- Processing element output can be directed back to the nodes in the preceding layer, forming a ***multilayer recurrent network***.
- Processing element output can be directed to processing element itself or to other processing element in the same layer.

learning

- Two broad kinds of learning in ANNs is :
- i) parameter learning – updates connecting weights in a neural net.
- ii) Structure learning – focus on change in the network.
- Apart from these, learning in ANN is classified into three categories as
 - i) supervised learning
 - ii) unsupervised learning
 - III) reinforcement learning

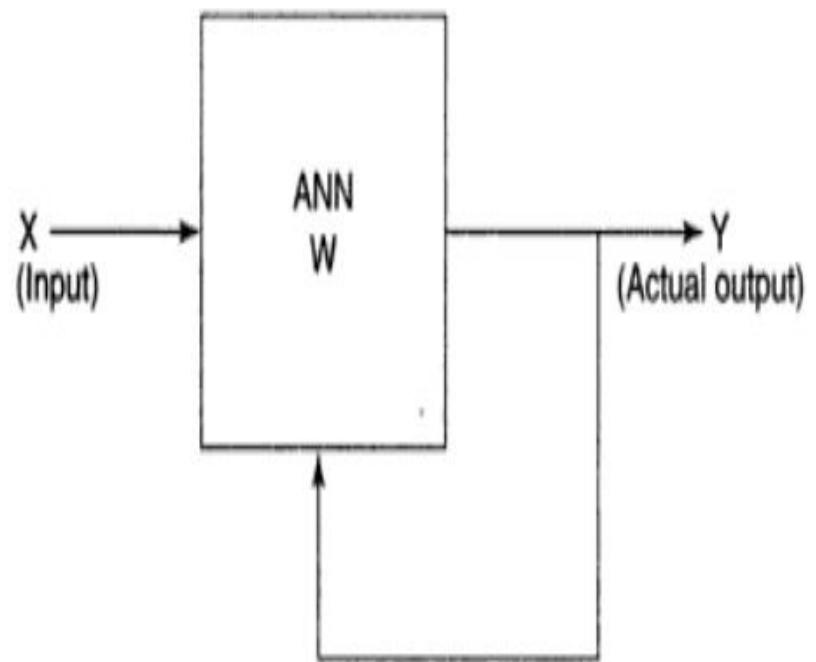
Supervised learning

- Learning with the help of a teacher.
- Example : learning process of a small child.
 - Child doesn't know read/write.
 - Their each & every action is supervised by a teacher
- In ANN, each input vector requires a corresponding target vector, which represents the desired output.
- The input vector along with target vector is called ***training pair***.
- The input vector results in output vector.
- The actual output vector is compared with desired output vector.
- If there is a difference means an error signal is generated by the network.
- It is used for adjustment of weights until actual output matches desired output.

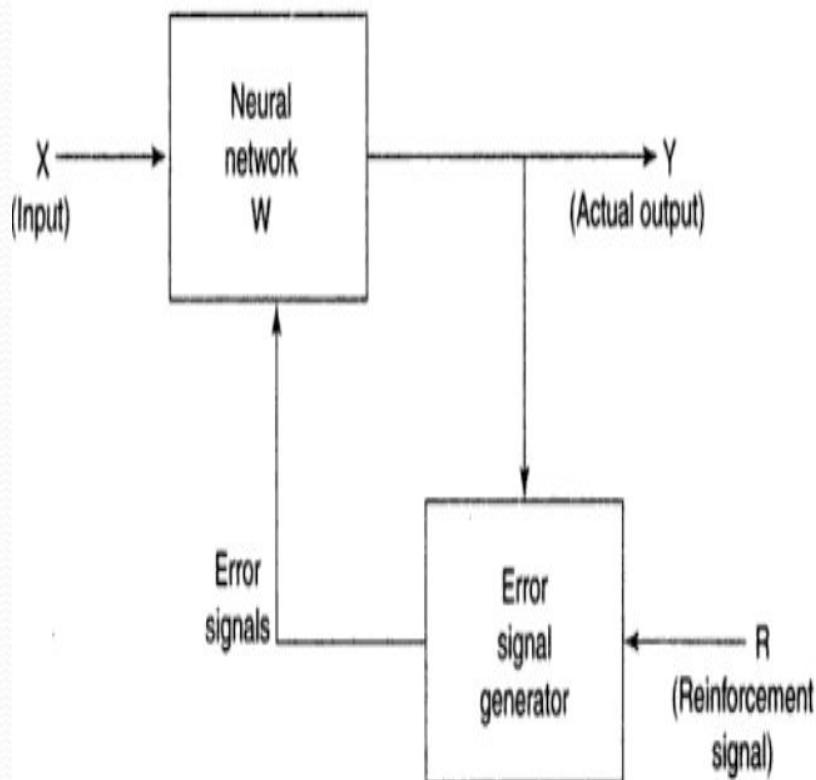


Unsupervised learning

- Learning is performed without the help of a teacher.
- Example: tadpole – learn to swim by itself.
- In ANN, during training process, network receives input patterns and organize it to form clusters.
- From the Fig. it is observed that no feedback is applied from environment to inform what output should be or whether they are correct.
- The network itself discover patterns, regularities, features/ categories from the input data and relations for the input data over the output.
- Exact clusters are formed by discovering similarities & dissimilarities so called as *self-organizing*.



Reinforcement learning



- Similar to supervised learning.
- Learning based on *critic information* is called *reinforcement learning* & the feedback sent is called *reinforcement signal*.
- The network receives some feedback from the environment.
- Feedback is only evaluative.
- The external reinforcement signals are processed in the critic signal generator, and the obtained critic signals are sent to the ANN for adjustment of weights properly to get critic feedback in future.

Activation functions

- To make work more efficient and for exact output, some force or activation is given.
- Like that, activation function is applied over the net input to calculate the output of an ANN.
- Information processing of processing element has two major parts: input and output.
- An integration function (f) is associated with input of processing element.
- Single Layer ANN provided with linear function and multilayer provided with non linear as liner function don't affect the output of multilayer ANN

- Several activation functions are there.

1. Identity function:

- it is a linear function which is defined as

$$f(x) = x \text{ for all } x$$

- The output is same as the input.

2. Binary step function

- it is defined as

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$

where θ represents thresh hold value.

It is used in single layer nets to convert the net input to an output that is binary. (0 or 1)

3. Bipolar step function:

- It is a linear function defined as

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ -1 & \text{if } x < \theta \end{cases}$$

- where θ represents threshold value.
- used in single layer nets to convert the net input to an output that is bipolar (+1 or -1).

4. Sigmoid function

- used in Back propagation nets

Two types:

a) *binary sigmoid function*

-logistic sigmoid function or unipolar sigmoid function.

-it is defined as

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

where λ – steepness parameter.

-The derivative of this function is

$f'(x) = \lambda f(x)[1-f(x)]$. The range of sigmoid function is 0 to 1.

b) Bipolar sigmoid function

$$f(x) = \frac{2}{1+e^{-\lambda x}} - 1 = \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}$$

where λ - steepness parameter and the sigmoid range is between -1 and +1.

The derivative of this function can be

$$f'(x) = \frac{\lambda}{2}(x)[1-f(x)]$$

The bipolar sigmoid function is closely related to hyperbolic tangent function, which is written as -

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$h(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Contd..

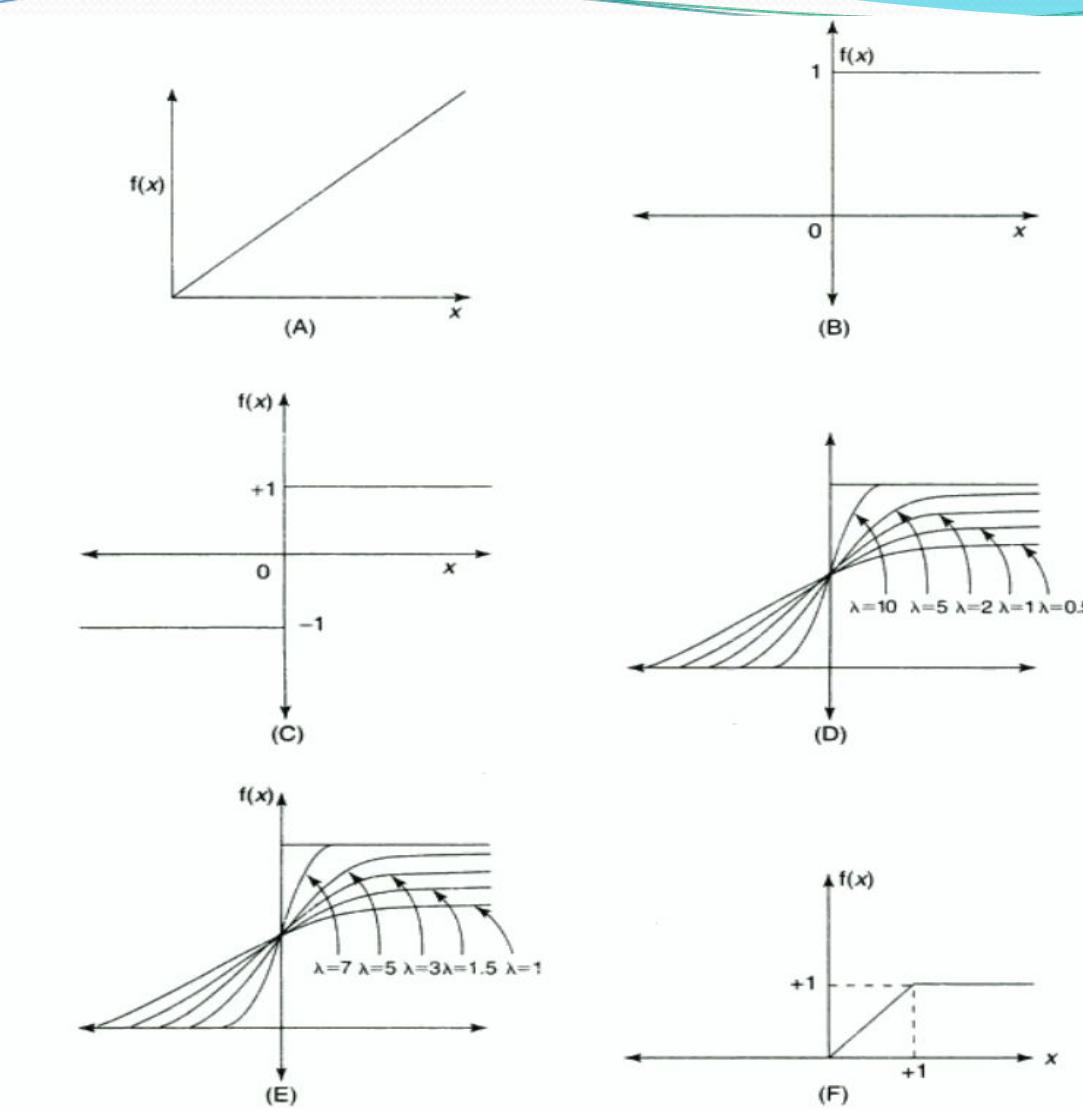
The derivative of the hyperbolic tangent function is

$$h'(x) = [1+h(x)][1-h(x)]$$

5. Ramp function

$$f(x) = \begin{cases} 1 & \text{if } x > 1 \\ x & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } x < 0 \end{cases}$$

The graphical representation of all these functions is given in the upcoming Figure



2-15 Depiction of activation functions: (A) identity function; (B) binary step function; (C) bipolar step function; (D) binary sigmoidal function; (E) bipolar sigmoidal function; (F) ramp function.

Important terminologies

● Weight

- The weight contain information about the input signal.
- It is used by the net to solve the problem.
- It is represented in terms of matrix & called as *connection matrix*.
- If weight matrix W contains all the elements of an ANN, then the set of all W matrices will determine the set of all possible information processing configuration.
- The ANN can be realized by finding an appropriate matrix W .
- Weight encode long-term memory (LTM) and the activation states of network encode short-term memory (STM) in a neural network.

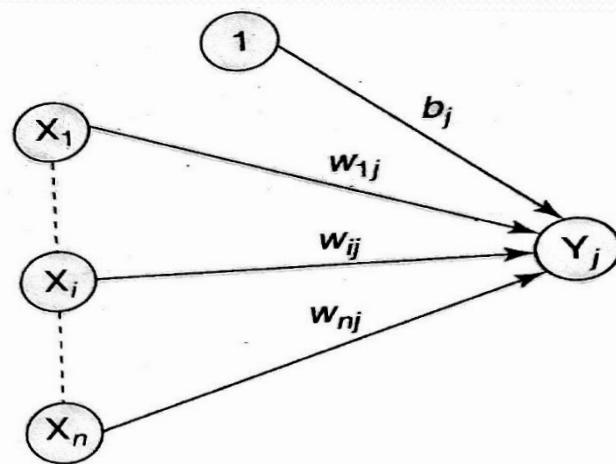


Figure 2-16 Simple net with bias.

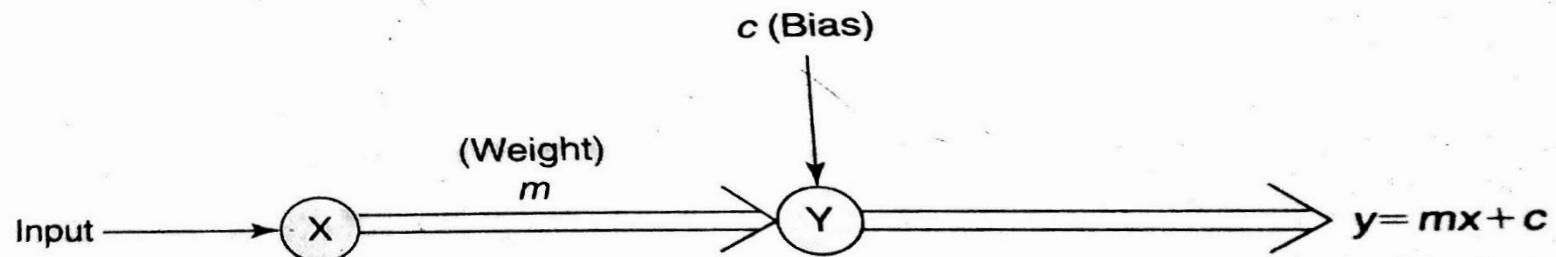


Figure 2-17 Block diagram for straight line.

Bias

- Bias has an impact in calculating net input.
- Bias is included by adding x_0 to the input vector x .
- The net output is calculated by

$$y_{inj} = \sum_{i=0}^n x_i w_{ij}$$

- The bias is of two types
 - Positive bias
 - Increase the net input
 - Negative bias
 - Decrease the net input

$$y_{inj} = b_j + \sum_{i=0}^n x_i w_{ij}$$

● Threshold

- It is a set value based upon which the final output is calculated.
- Used in activation function
- Calculated net input and threshold is compared to get the network output.
- The activation function of threshold is defined as

$$f(\text{net}) = \begin{cases} 1 & \text{if } \text{net} \geq \theta \\ -1 & \text{if } \text{net} < \theta \end{cases}$$

- where θ is the fixed threshold value

Contd..

- **Learning rate**
 - Denoted by α .
 - Control the amount of weight adjustment at each step of training.
 - The learning rate range from 0 to 1.
 - Determine the rate of learning at each step
- **Momentum Factor**
 - Convergence is made faster if a momentum factor is added to the weight updation process.
 - Done in back propagation network.
- **Vigilance parameter**
 - Denoted by ρ .
 - Used in Adaptive Resonance Theory (ART) network.
 - Used to control the degree of similarity.
 - Ranges from 0.7 to 1 to perform useful work in controlling the number of clusters.

The back-propagation algorithm is different from other networks in respect to the process by which the weights are calculated during the learning period of the network. The general difficulty with the multilayer perceptrons is calculating the weights of the hidden layers in an efficient way that would result in a very small or zero output error. When the hidden layers are increased the network training becomes more complex. To update weights, the error must be calculated. The error, which is the difference between the actual (calculated) and the desired (target) output, is easily measured at the output layer. It should be noted that at the hidden layers, there is no direct information of the error. Therefore, other techniques should be used to calculate an error at the hidden layer, which will cause minimization of the output error, and this is the ultimate goal.

The training of the BPN is done in three stages – the feed-forward of the input training pattern, the calculation and back-propagation of the error, and updation of weights. The testing of the BPN involves the computation of feed-forward phase only. There can be more than one hidden layer (more beneficial) but one hidden layer is sufficient. Even though the training is very slow, once the network is trained it can produce its outputs very rapidly.

3.5.2 Architecture

A back-propagation neural network is a multilayer, feed-forward neural network consisting of an input layer, a hidden layer and an output layer. The neurons present in the hidden and output layers have biases, which are the connections from the units whose activation is always 1. The bias terms also acts as weights. Figure 3-9 shows the architecture of a BPN, depicting only the direction of information flow for the feed-forward phase. During the back-propagation phase of learning, signals are sent in the reverse direction.

The inputs are sent to the BPN and the output obtained from the net could be either binary (0, 1) or bipolar (-1, +1). The activation function could be any function which increases monotonically and is also differentiable.

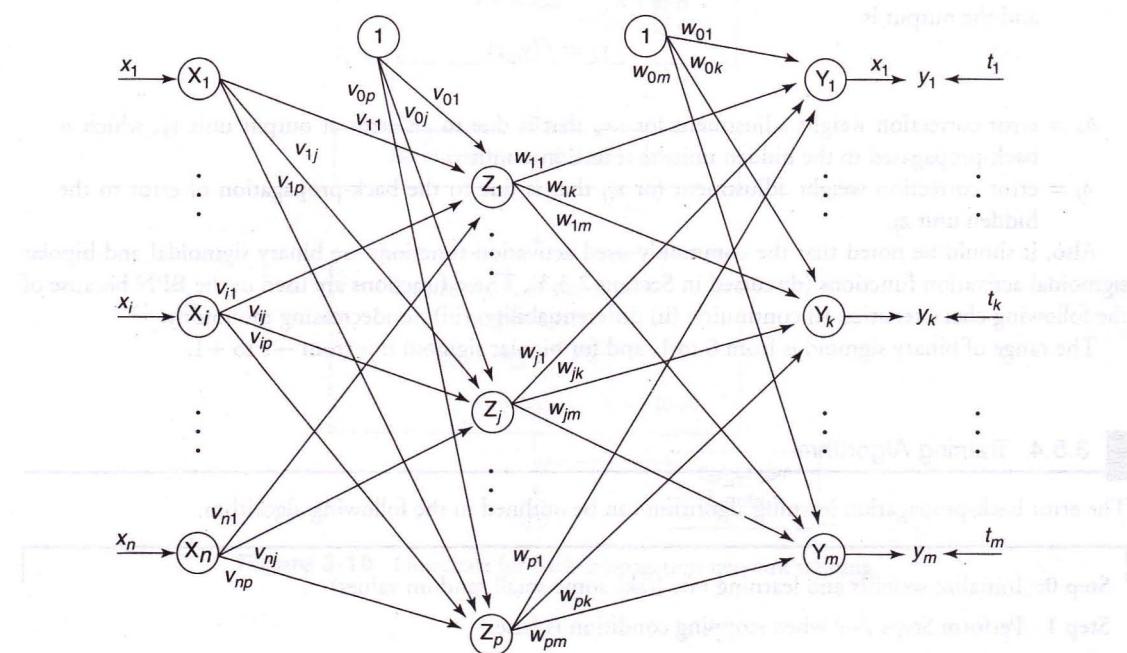


Figure 3-9 Architecture of a back-propagation network.

3.5.3 Flowchart for Training Process

The flowchart for the training process using a BPN is shown in Figure 3-10. The terminologies used in the flowchart and in the training algorithm are as follows:

x = input training vector ($x_1, \dots, x_i, \dots, x_n$)

t = target output vector ($t_1, \dots, t_k, \dots, t_m$)

α = learning rate parameter

x_i = input unit i . (Since the input layer uses identity activation function, the input and output signals here are same.)

v_{0j} = bias on j th hidden unit

w_{0k} = bias on k th output unit

z_j = hidden unit j . The net input to z_j is

$$z_{inj} = v_{0j} + \sum_{i=1}^n x_i v_{ij}$$

and the output is

$$z_j = f(z_{inj})$$

y_k = output unit k . The net input to y_k is

$$y_{ink} = w_{0k} + \sum_{j=1}^p z_j w_{jk}$$

and the output is

$$y_k = f(y_{ink})$$

δ_k = error correction weight adjustment for w_{jk} that is due to an error at output unit y_k , which is back-propagated to the hidden units that feed into unit y_k

δ_j = error correction weight adjustment for v_{ij} that is due to the back-propagation of error to the hidden unit z_j .

Also, it should be noted that the commonly used activation functions are binary sigmoidal and bipolar sigmoidal activation functions (discussed in Section 2.3.3). These functions are used in the BPN because of the following characteristics: (i) continuity; (ii) differentiability; (iii) nondecreasing monotony.

The range of binary sigmoid is from 0 to 1, and for bipolar sigmoid it is from -1 to +1.

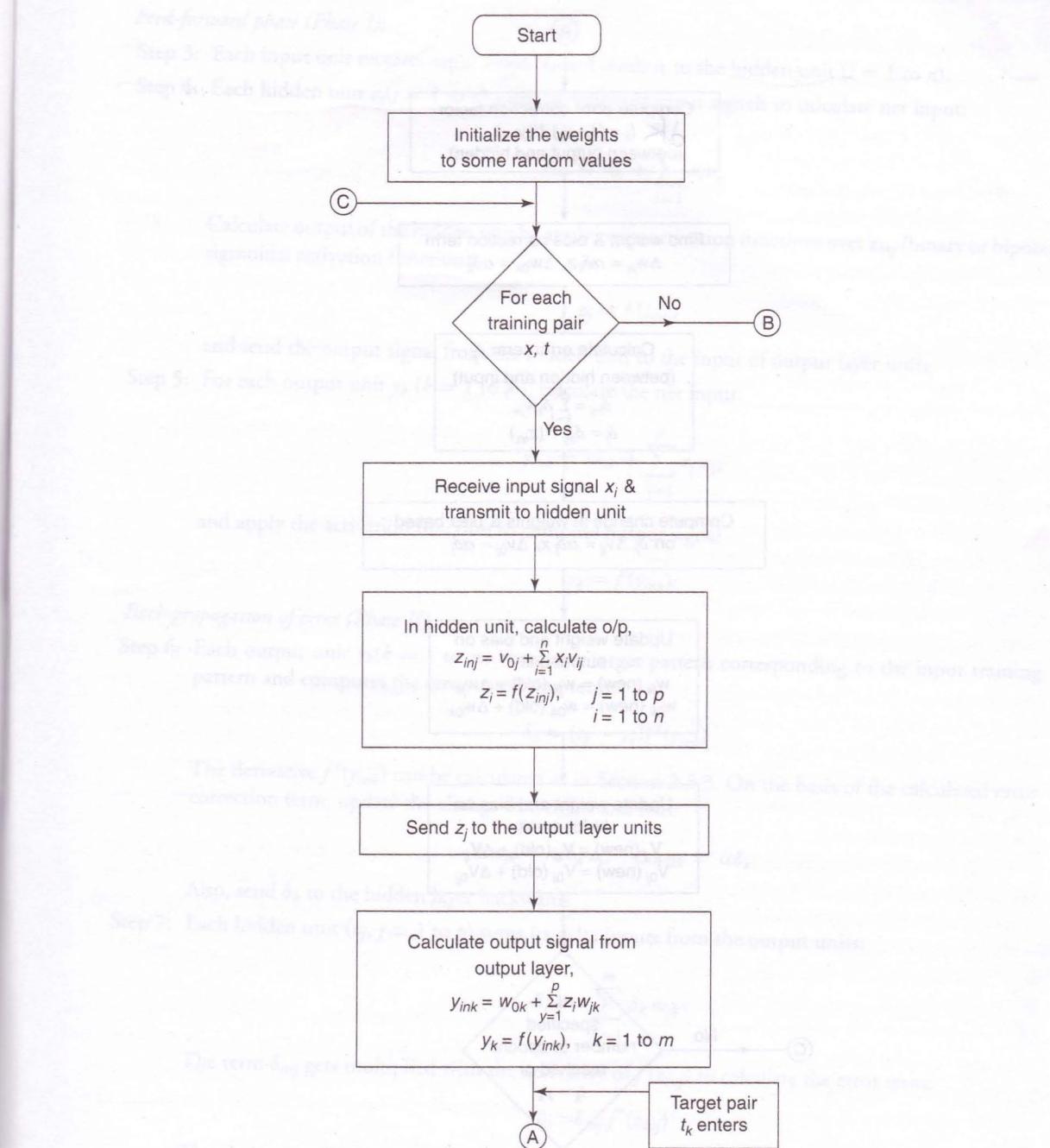
3.5.4 Training Algorithm

The error back-propagation learning algorithm can be outlined in the following algorithm:

Step 0: Initialize weights and learning rate (take some small random values).

Step 1: Perform Steps 2–9 when stopping condition is false.

Step 2: Perform Steps 3–8 for each training pair.

**Figure 3-10** Flowchart for back-propagation network training.

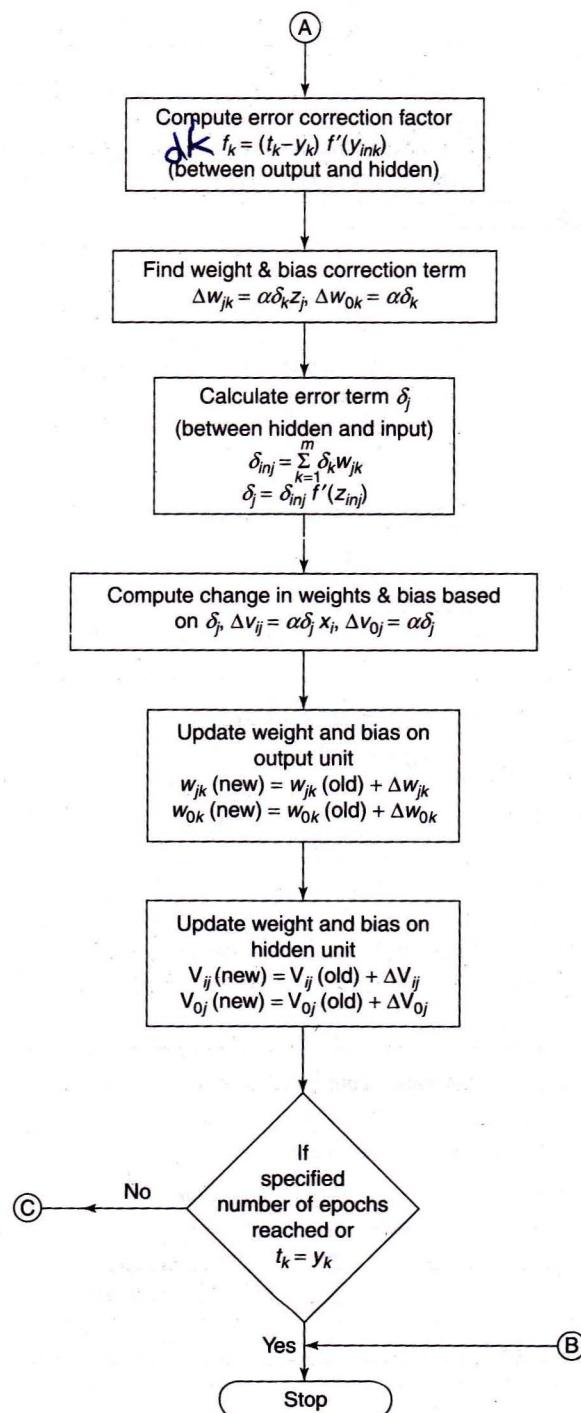


Figure 3-10 (Continued).

Feed-forward phase (Phase I):

Step 3: Each input unit receives input signal x_i and sends it to the hidden unit ($i = 1$ to n).

Step 4: Each hidden unit z_j ($j = 1$ to p) sums its weighted input signals to calculate net input:

$$z_{inj} = v_{0j} + \sum_{i=1}^n x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over z_{inj} (binary or bipolar sigmoidal activation function):

$$z_j = f(z_{inj})$$

and send the output signal from the hidden unit to the input of output layer units.

Step 5: For each output unit y_k ($k = 1$ to m), calculate the net input:

$$y_{ink} = w_{0k} + \sum_{j=1}^p z_j w_{jk}$$

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Back-propagation of error (Phase II):

Step 6: Each output unit y_k ($k = 1$ to m) receives a target pattern corresponding to the input training pattern and computes the error correction term:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

The derivative $f'(y_{ink})$ can be calculated as in Section 2.3.3. On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j; \quad \Delta w_{0k} = \alpha \delta_k$$

Also, send δ_k to the hidden layer backwards.

Step 7: Each hidden unit ($z_j, j = 1$ to p) sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^m \delta_k w_{jk}$$

The term δ_{inj} gets multiplied with the derivative of $f(z_{inj})$ to calculate the error term:

$$\delta_j = \delta_{inj} f'(z_{inj})$$

The derivative $f'(z_{inj})$ can be calculated as discussed in Section 2.3.3 depending on whether binary or bipolar sigmoidal function is used. On the basis of the calculated δ_j , update the change in weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i; \quad \Delta v_{0j} = \alpha \delta_j$$

Weight and bias updation (Phase III):

Step 8: Each output unit (y_k , $k = 1$ to m) updates the bias and weights:

$$\begin{aligned} w_{jk}(\text{new}) &= w_{jk}(\text{old}) + \Delta w_{jk} \\ w_{0k}(\text{new}) &= w_{0k}(\text{old}) + \Delta w_{0k} \end{aligned}$$

Each hidden unit (z_j , $j = 1$ to p) updates its bias and weights:

$$\begin{aligned} v_{ij}(\text{new}) &= v_{ij}(\text{old}) + \Delta v_{ij} \\ v_{0j}(\text{new}) &= v_{0j}(\text{old}) + \Delta v_{0j} \end{aligned}$$

Step 9: Check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output.

The above algorithm uses the incremental approach for updation of weights, i.e., the weights are being changed immediately after a training pattern is presented. There is another way of training called *batch-mode training*, where the weights are changed only after all the training patterns are presented. The effectiveness of two approaches depends on the problem, but batch-mode training requires additional local storage for each connection to maintain the immediate weight changes. When a BPN is used as a classifier, it is equivalent to the optimal Bayesian discriminant function for asymptotically large sets of statistically independent training patterns.

The problem in this case is whether the back-propagation learning algorithm can always converge and find proper weights for network even after enough learning. It will converge since it implements a gradient-descent on the error surface in the weight space, and this will roll down the error surface to the nearest minimum error and will stop. This becomes true only when the relation existing between the input and the output training patterns is deterministic and the error surface is deterministic. This is not the case in real world because the produced square-error surfaces are always at random. This is the stochastic nature of the back-propagation algorithm, which is purely based on the stochastic gradient-descent method. The BPN is a special case of stochastic approximation.

If the BPN algorithm converges at all, then it may get stuck with local minima and may be unable to find satisfactory solutions. The randomness of the algorithm helps it to get out of local minima. The error functions may have large number of global minima because of permutations of weights that keep the network input-output function unchanged. This causes the error surfaces to have numerous troughs.

3.5.5 Learning Factors of Back-Propagation Network

The training of a BPN is based on the choice of various parameters. Also, the convergence of the BPN is based on some important learning factors such as the initial weights, the learning rate, the updation rule, the size and nature of the training set, and the architecture (number of layers and number of neurons per layer).

3.5.5.1 Initial Weights

The ultimate solution may be affected by the initial weights of a multilayer feed-forward network. They are initialized at small random values. The choice of the initial weight determines how fast the network converges. The initial weights cannot be very high because the sigmoidal activation functions used here may get saturated

from the beginning itself and the system may be stuck at a local minima or at a very flat plateau at the starting point itself. One method of choosing the weight w_{ij} is choosing it in the range

$$\left[\frac{-3}{\sqrt{o_i}}, \frac{3}{\sqrt{o_i}} \right]$$

where o_i is the number of processing elements j that feed-forward to processing element i . The initialization can also be done by a method called Nyugen-Widrow initialization. This type of initialization leads to faster convergence of network. The concept here is based on the geometric analysis of the response of hidden neurons to a single input. The method is used for improving the learning ability of the hidden units. The random initialization of weights connecting input neurons to the hidden neurons is obtained by the equation

$$v_{ij}(\text{new}) = \gamma \frac{v_{ij}(\text{old})}{\|v_j(\text{old})\|}$$

where \bar{v}_j is the average weight calculated for all values of i , and the scale factor $\gamma = 0.7(P)^{1/n}$ ("n" is the number of input neurons and "P" is the number of hidden neurons).

3.5.5.2 Learning Rate α

The learning rate (α) affects the convergence of the BPN. A larger value of α may speed up the convergence but might result in overshooting, while a smaller value of α has vice-versa effect. The range of α from 10^{-3} to 10 has been used successfully for several back-propagation algorithmic experiments. Thus, a large learning rate leads to rapid learning but there is oscillation of weights, while the lower learning rate leads to slower learning.

3.5.5.3 Momentum Factor

The gradient descent is very slow if the learning rate α is small and oscillates widely if α is too large. One very efficient and commonly used method that allows a larger learning rate without oscillations is by adding a momentum factor to the normal gradient descent method.

The momentum factor is denoted by $\eta \in [0, 1]$ and the value of 0.9 is often used for the momentum factor. Also, this approach is more useful when some training data are very different from the majority of data. A momentum factor can be used with either pattern by pattern updating or batch-mode updating. In case of batch mode, it has the effect of complete averaging over the patterns. Even though the averaging is only partial in the pattern-by-pattern mode, it leaves some useful information for weight updation.

The weight updation formulas used here are

$$w_{jk}(t+1) = w_{jk}(t) + \underbrace{\alpha \delta_k z_j + \eta [w_{jk}(t) - w_{jk}(t-1)]}_{\Delta w_{jk}(t+1)}$$

and

$$v_{ij}(t+1) = v_{ij}(t) + \underbrace{\alpha \delta_j x_i + \eta [v_{ij}(t) - v_{ij}(t-1)]}_{\Delta v_{ij}(t+1)}$$

The momentum factor also helps in faster convergence.

4

Associative Memory Networks

Learning Objectives

- Gives details on associative memories.
- Discusses the training algorithm used for pattern association networks – Hebb rule and outer products rule.
- The architecture, flowchart for training process, training algorithm and testing algorithm of autoassociative, heteroassociative and bidirectional associative memory are discussed in detail.
- Variants of BAM – continuous BAM and discrete BAM are included.
- Hopfield network with its electrical model is described with training algorithm.
- Analysis of energy function was performed for BAM, discrete and continuous Hopfield networks.
- An overview is given on the iterative autoassociative network – linear autoassociator memory brain-in-the-box network and autoassociator with threshold unit.
- Also temporal associative memory is discussed in brief.

4.1 Introduction

An associative memory network can store a set of patterns as memories. When the associative memory is being presented with a key pattern, it responds by producing one of the stored patterns, which closely resembles or relates to the key pattern. Thus, the recall is through association of the key pattern, with the help of information memorized. These types of memories are also called as *content-addressable memories* (CAM) in contrast to that of traditional *address-addressable memories* in digital computers where stored pattern (in bytes) is recalled by its address. It is also a matrix memory as in RAM/ROM. The CAM can also be viewed as associating data to address, i.e., for every data in the memory there is a corresponding unique address. Also, it can be viewed as data correlator. Here input data is correlated with that of the stored data in the CAM. It should be noted that the stored patterns must be unique, i.e., different patterns in each location. If the same pattern exists in more than one location in the CAM, then, even though the correlation is correct, the address is noted to be ambiguous. The basic structure of CAM is given in Figure 4-1.

Associative memory makes a parallel search within a stored data file. The concept behind this search is to output any one or all stored items which match the given search argument and to retrieve the stored data either completely or partially.

Two types of associative memories can be differentiated. They are *autoassociative memory* and *heteroassociative memory*. Both these nets are single-layer nets in which the weights are determined in a manner that the net stores a set of pattern associations. Each of this association is an input–output vector pair, say, *s.t.* If each of the output vectors is same as the input vectors with which it is associated, then the net is said to

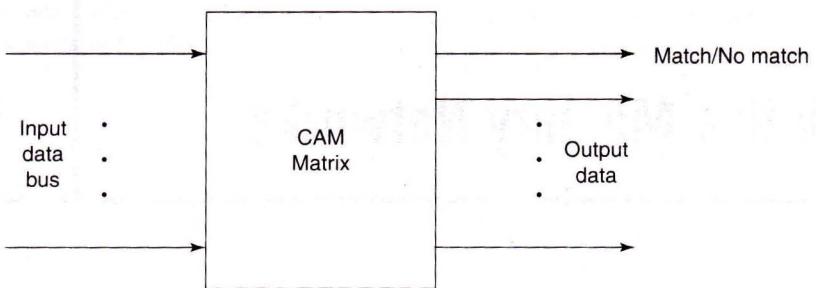


Figure 4-1 CAM architecture.

be autoassociative memory net. On the other hand, if the output vectors are different from the input vectors then the net is said to be heteroassociative memory net.

If there exist vectors, say, $x = (x_1, x_2, \dots, x_n)^T$ and $x' = (x'_1, x'_2, \dots, x'_n)^T$, then the hamming distance (HD) is defined as the number of mismatched components of x and x' vectors, i.e.,

$$\text{HD}(x, x') = \begin{cases} \sum_{i=1}^n |x_i - x'_i| & \text{if } x_i, x'_i \in [0, 1] \\ \frac{1}{2} \sum_{i=1}^n |x_i - x'_i| & \text{if } x_i, x'_i \in [-1, 1] \end{cases}$$

The architecture of an associative net may be either feed-forward or iterative (recurrent). As is already known, in a feed-forward net the information flows from the input units to the output units; on the other hand, in a recurrent neural net, there are connections among the units to form a closed-loop structure. In the forthcoming sections, we will discuss the training algorithms used for pattern association and various types of association nets in detail.

4.2 Training Algorithms for Pattern Association

There are two algorithms developed for training of pattern association nets. These are discussed below.

4.2.1 Hebb Rule

The Hebb rule is widely used for finding the weights of an associative memory neural net. The training vector pairs here are denoted as $s:t$. The flowchart for the training algorithm of pattern association is as shown in Figure 4-2. The weights are updated until there is no weight change. The algorithmic steps followed are given below:

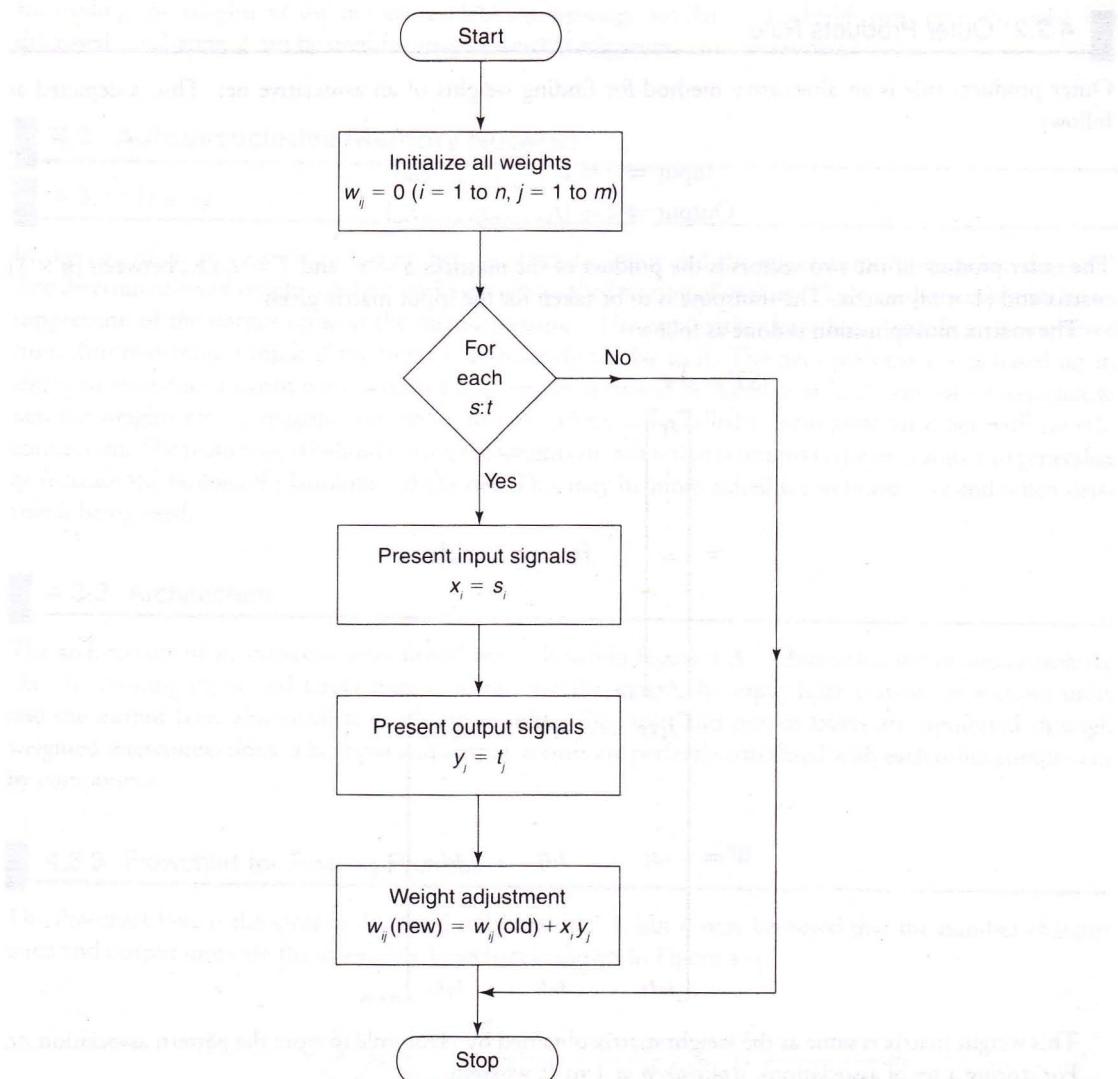
Step 0: Set all the initial weights to zero, i.e.,

$$w_{ij} = 0 \quad (i = 1 \text{ to } n, j = 1 \text{ to } m)$$

Step 1: For each training target input output vector pairs $s:t$, perform Steps 2–4.

Step 2: Activate the input layer units to current training input,

$$x_i = s_i \quad (\text{for } i = 1 \text{ to } n)$$

**Figure 4-2** Flowchart for Hebb rule.

Step 3: Activate the output layer units to current target output,

$$y_j = t_j \quad (\text{for } j = 1 \text{ to } m)$$

Step 4: Start the weight adjustment

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j \quad (\text{for } i = 1 \text{ to } n, j = 1 \text{ to } m)$$

This algorithm is used for the calculation of the weights of the associative nets. Also, it can be used with patterns that are being represented as either binary or bipolar vectors.

4.2.2 Outer Products Rule

Outer products rule is an alternative method for finding weights of an associative net. This is depicted as follows:

$$\text{Input} \Rightarrow s = (s_1, \dots, s_i, \dots, s_n)$$

$$\text{Output} \Rightarrow t = (t_1, \dots, t_j, \dots, t_m)$$

The outer product of the two vectors is the product of the matrices $S = s^T$ and $T = t$, i.e., between $[n \times 1]$ matrix and $[1 \times m]$ matrix. The transpose is to be taken for the input matrix given.

The matrix multiplication is done as follows:

$$ST = s^T t$$

$$= \begin{bmatrix} s_1 \\ \vdots \\ s_i \\ \vdots \\ s_n \end{bmatrix}_{n \times 1} [t_1 \dots t_j \dots t_m]_{1 \times m}$$

$$W = \begin{bmatrix} s_1 t_1 & \dots & s_1 t_j & \dots & s_1 t_m \\ \vdots & & \vdots & & \vdots \\ s_i t_1 & \dots & s_i t_j & \dots & s_i t_m \\ \vdots & & \vdots & & \vdots \\ s_n t_1 & \dots & s_n t_j & \dots & s_n t_m \end{bmatrix}_{n \times m}$$

This weight matrix is same as the weight matrix obtained by Hebb rule to store the pattern association *s.t.* For storing a set of associations, $s(p):t(p)$, $p = 1$ to P , wherein,

$$s(p) = (s_1(p), \dots, s_i(p), \dots, s_n(p))$$

$$t(p) = (t_1(p), \dots, t_j(p), \dots, t_m(p))$$

the weight matrix $W = \{w_{ij}\}$ can be given as

$$w_{ij} = \sum_{p=1}^P s_i^T(p) t_j(p)$$

This can also be rewritten as

$$W = \sum_{p=1}^P s^T(p) t(p)$$

The above five steps complete the algorithmic process. In Step 4, the weight updation formula can also be given in vector form as

$$w(\text{new}) = w(\text{old}) + xy$$

Here the change in weight can be expressed as

$$\Delta w = xy$$

As a result,

$$w(\text{new}) = w(\text{old}) + \Delta w$$

The Hebb rule can be used for pattern association, pattern categorization, pattern classification and over a range of other areas.

2.8 Summary

In this chapter we have discussed the basics of an ANN and its growth. A detailed comparison between biological neuron and artificial neuron has been included to enable the reader understand the basic difference between them. An ANN is constructed with few basic building blocks. The building blocks are based on the models of artificial neurons and the topology of few basic structures. Concepts of supervised learning, unsupervised learning and reinforcement learning are briefly included in this chapter. Various activation functions and different types of layered connections are also considered here. The basic terminologies of ANN are discussed with their typical values. A brief description on McCulloch-Pitts neuron model is provided. The concept of linear separability is discussed and illustrated with suitable examples. Details are provided for the effective training of a Hebb network.

2.9 Solved Problems

1. For the network shown in Figure 1, calculate the net input to the output neuron.

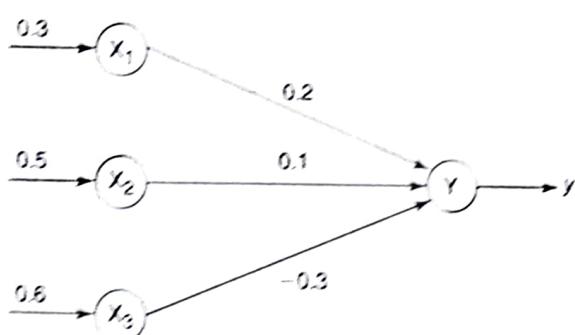


Figure 1 Neural net.

Solution: The given neural net consists of three input neurons and one output neuron. The inputs and

$$\begin{aligned} [x_1, x_2, x_3] &= [0.3, 0.5, 0.6] \\ [w_1, w_2, w_3] &= [0.2, 0.1, -0.3] \end{aligned}$$

The net input can be calculated as

$$\begin{aligned} y_{in} &= x_1 w_1 + x_2 w_2 + x_3 w_3 \\ &= 0.3 \times 0.2 + 0.5 \times 0.1 + 0.6 \times (-0.3) \\ &= 0.06 + 0.05 - 0.18 = -0.07 \end{aligned}$$

2. Calculate the net input for the network shown in Figure 2 with bias included in the network.

Solution: The given net consists of two input neurons, a bias and an output neuron. The inputs are

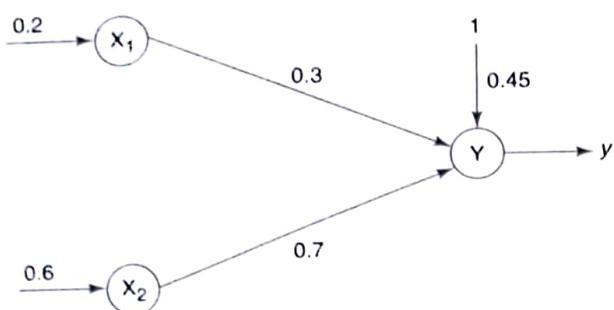


Figure 2 Simple neural net.

$[x_1, x_2] = [0.2, 0.6]$ and the weights are $[w_1, w_2] = [0.3, 0.7]$. Since the bias is included $b = 0.45$ and bias input x_0 is equal to 1, the net input is calculated as

$$\begin{aligned}y_{in} &= b + x_1 w_1 + x_2 w_2 \\&= 0.45 + 0.2 \times 0.3 + 0.6 \times 0.7 \\&= 0.45 + 0.06 + 0.42 = 0.93\end{aligned}$$

Therefore $y_{in} = 0.93$ is the net input.

3. Obtain the output of the neuron Y for the network shown in Figure 3 using activation functions as: (i) binary sigmoidal and (ii) bipolar sigmoidal.

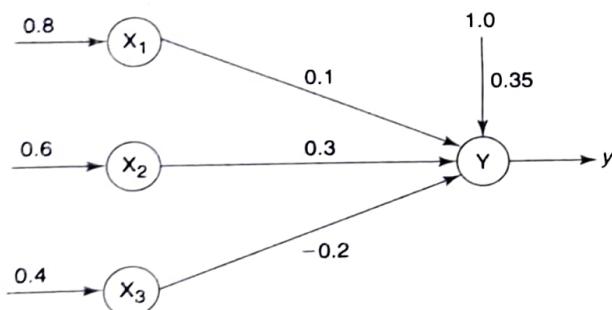


Figure 3 Neural net.

Solution: The given network has three input neurons with bias and one output neuron. These form a single-layer network. The inputs are given as $[x_1, x_2, x_3] = [0.8, 0.6, 0.4]$ and the weights are $[w_1, w_2, w_3] = [0.1, 0.3, -0.2]$ with bias $b = 0.35$ (its input is always 1).

The net input to the output neuron is

$$y_{in} = b + \sum_{i=1}^n x_i w_i$$

$$\begin{aligned}[n &= 3, \text{ because only} \\&3 \text{ input neurons are given}] \\&= b + x_1 w_1 + x_2 w_2 + x_3 w_3 \\&= 0.35 + 0.8 \times 0.1 + 0.6 \times 0.3 \\&\quad + 0.4 \times (-0.2) \\&= 0.35 + 0.08 + 0.18 - 0.08 = 0.53\end{aligned}$$

- (i) For binary sigmoidal activation function,

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-0.53}} = 0.625$$

- (ii) For bipolar sigmoidal activation function,

$$\begin{aligned}y &= f(y_{in}) = \frac{2}{1 + e^{-y_{in}}} - 1 = \frac{2}{1 + e^{-0.53}} - 1 \\&= 0.259\end{aligned}$$

4. Implement AND function using McCulloch–Pitts neuron (take binary data).

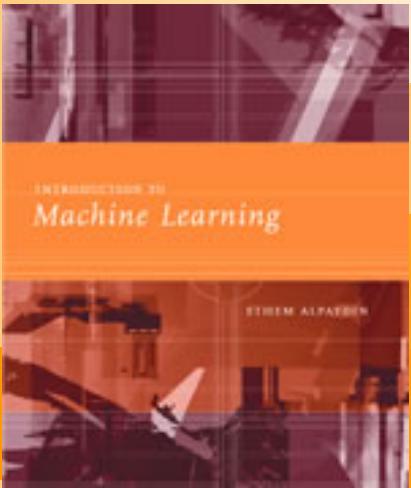
Solution: Consider the truth table for AND function (Table 1).

Table 1

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

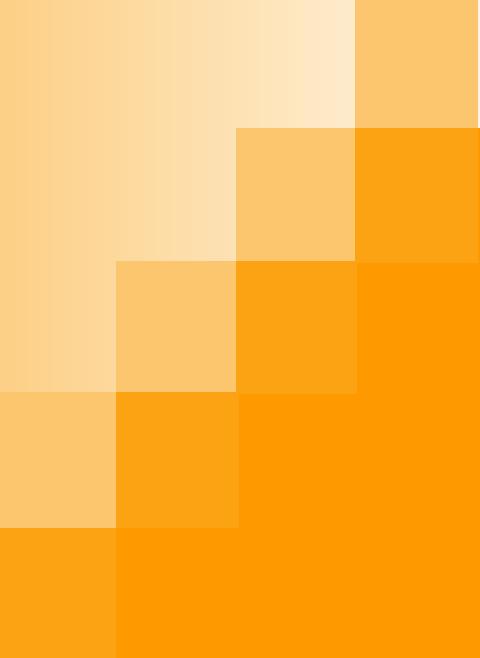
In McCulloch–Pitts neuron, only analysis is being performed. Hence, assume the weights be $w_1 = 1$ and $w_2 = 1$. The network architecture is shown in Figure 4. With these assumed weights, the net input is calculated for four inputs: For inputs

$$\begin{aligned}(1, 1), \quad y_{in} &= x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2 \\(1, 0), \quad y_{in} &= x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1 \\(0, 1), \quad y_{in} &= x_1 w_1 + x_2 w_2 = 0 \times 1 + 1 \times 1 = 1 \\(0, 0), \quad y_{in} &= x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0\end{aligned}$$



INTRODUCTION TO

Machine Learning



CHAPTER 1:

Introduction

Why “Learn” ?

- Machine learning is programming computers to optimize a performance criterion using example data or past experience.
- There is no need to “learn” to calculate payroll
- Learning is used when:
 - Human expertise does not exist (navigating on Mars),
 - Humans are unable to explain their expertise (speech recognition)
 - Solution changes in time (routing on a computer network)
 - Solution needs to be adapted to particular cases (user biometrics)

What We Talk About When We Talk About “Learning”

- Learning general models from a data of particular examples
- Data is cheap and abundant (data warehouses, data marts); knowledge is expensive and scarce.
- Example in retail: Customer transactions to consumer behavior:

People who bought “Da Vinci Code” also bought “The Five People You Meet in Heaven” (www.amazon.com)
- Build a model that is *a good and useful approximation* to the data.

Data Mining

- **Retail:** Market basket analysis, Customer relationship management (CRM)
- **Finance:** Credit scoring, fraud detection
- **Manufacturing:** Optimization, troubleshooting
- **Medicine:** Medical diagnosis
- **Telecommunications:** Quality of service optimization
- **Bioinformatics:** Motifs, alignment
- **Web mining:** Search engines
- ...

What is Machine Learning?

- Optimize a performance criterion using example data or past experience.
- Role of Statistics: Inference from a sample
- Role of Computer science: Efficient algorithms to
 - Solve the optimization problem
 - Representing and evaluating the model for inference

Applications

- Association
- Supervised Learning
 - Classification
 - Regression
- Unsupervised Learning
- Reinforcement Learning

Learning Associations

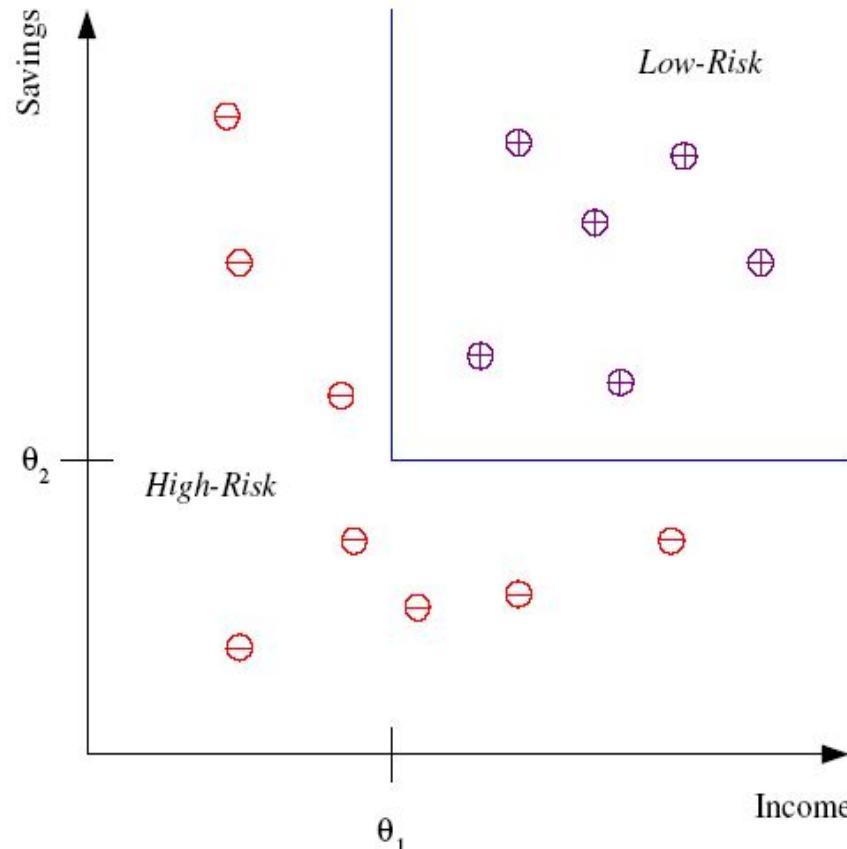
- Basket analysis:

$P(Y | X)$ probability that somebody who buys X also buys Y where X and Y are products/services.

Example: $P(\text{chips} | \text{beer}) = 0.7$

Classification

- Example: Credit scoring
- Differentiating between **low-risk** and **high-risk** customers from their *income* and *savings*



Discriminant: IF $\text{income} > \theta_1$ AND $\text{savings} > \theta_2$
THEN **low-risk** ELSE **high-risk**

Classification: Applications

- Aka Pattern recognition
- Face recognition: Pose, lighting, occlusion (glasses, beard), make-up, hair style
- Character recognition: Different handwriting styles.
- Speech recognition: Temporal dependency.
 - Use of a dictionary or the syntax of the language.
 - Sensor fusion: Combine multiple modalities; eg, visual (lip image) and acoustic for speech
- Medical diagnosis: From symptoms to illnesses
- ...

Face Recognition

Training examples of a person



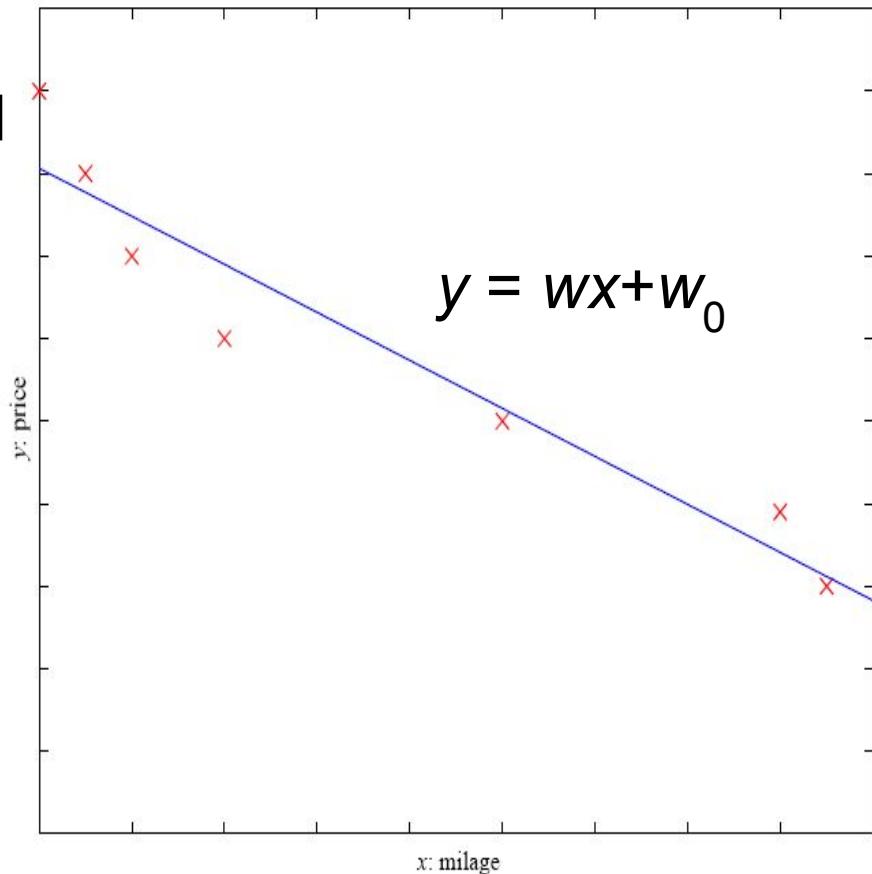
Test images



AT&T Laboratories, Cambridge UK
<http://www.uk.research.att.com/facedatabase.html>

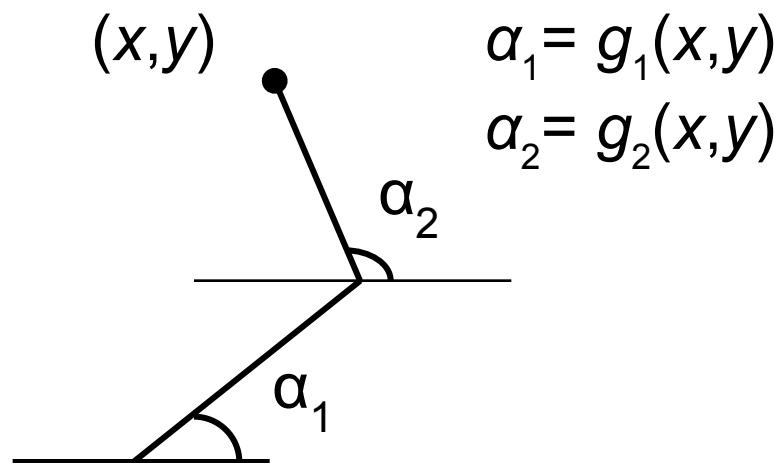
Regression

- Example: Price of a used car
- x : car attributes
 y : price
 $y = g(x | \theta)$
 $g(\cdot)$ model,
 θ parameters

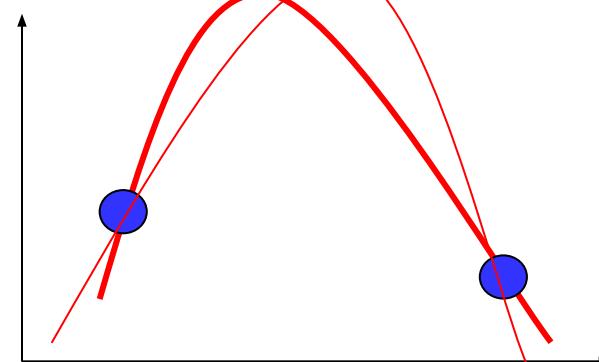


Regression Applications

- Navigating a car: Angle of the steering wheel (CMU NavLab)
- Kinematics of a robot arm



- Response surface design



Supervised learning

- Supervised learning is an approach to creating artificial intelligence (**AI**), where a computer algorithm is trained on input data that has been labeled for a particular output. The model is trained until it can detect the underlying patterns and relationships between the input data and the output labels, enabling it to yield accurate labeling results when presented with never-before-seen data.

Supervised Learning:

- Naïve Base Classifier, ,
- Classifying with k-Nearest
- Neighbour classifier,
- Decision Tree classifier,
- Naive Bayes classifier.

Supervised Learning: Uses

- **Prediction of future cases:** Use the rule to predict the output for future inputs
- **Knowledge extraction:** The rule is easy to understand
- **Compression:** The rule is simpler than the data it explains
- **Outlier detection:** Exceptions that are not covered by the rule, e.g., fraud

Unsupervised learning

- the algorithm is given unlabeled data as a training set. Unlike in supervised learning, there are no correct output values; the algorithm determines the patterns and similarities within the data, as opposed to relating it to some external measurement.
- In other words, algorithms are able to function freely in order to learn more about the data and find interesting or unexpected findings that human beings weren't looking for.

Unsupervised Learning

- k-means clustering,
- Apriori algorithm
- Reinforcement learning

Unsupervised Learning

- Learning “what normally happens”
- No output
- Clustering: Grouping similar instances
- Example applications
 - Customer segmentation in CRM
 - Image compression: Color quantization
 - Bioinformatics: Learning motifs

Reinforcement Learning

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.

Reinforcement Learning

- Learning a policy: A **sequence** of outputs
- No supervised output but delayed reward
- Credit assignment problem
- Game playing
- Robot in a maze
- Multiple agents, partial observability, ...



Introduction To ML



AGENDA

- Machine Learning basics,
- Applications of ML,
- Data Mining Vs Machine Learning vs Big Data Analytics.



A Few Quotes

- “A breakthrough in machine learning would be worth ten Microsofts” (**Bill Gates, Chairman, Microsoft**)
- “Machine learning is the next Internet” (**Tony Tether, Director, DARPA**)
- Machine learning is the hot new thing” (**John Hennessy, President, Stanford**)
- “Web rankings today are mostly a matter of machine learning” (**Prabhakar Raghavan, Dir. Research, Yahoo**)
- “Machine learning is going to result in a real revolution” (**Greg Papadopoulos, CTO, Sun**)
- “Machine learning is today’s discontinuity” (**Jerry Yang, CEO, Yahoo**)



So What Is Machine Learning?

- Automating automation
- Getting computers to program themselves
- Writing software is the bottleneck
- Let the data do the work instead!

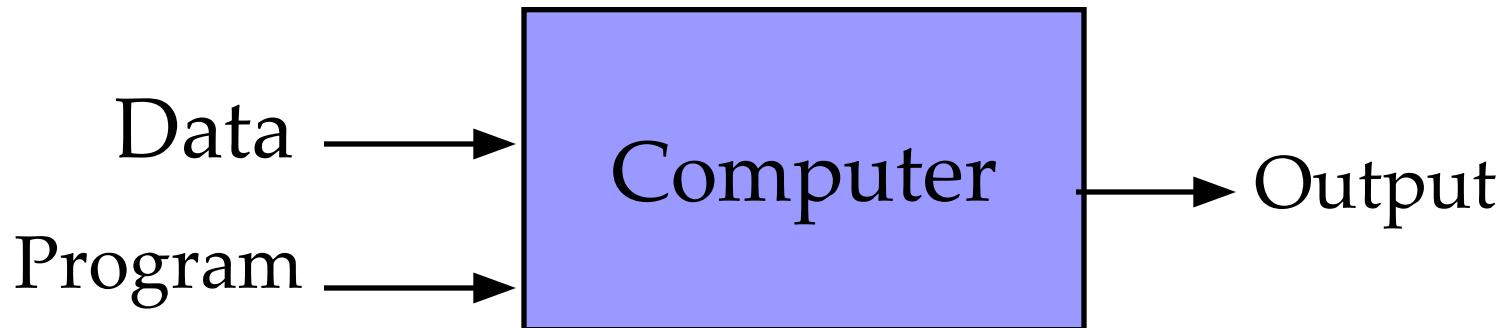


Machine Learning

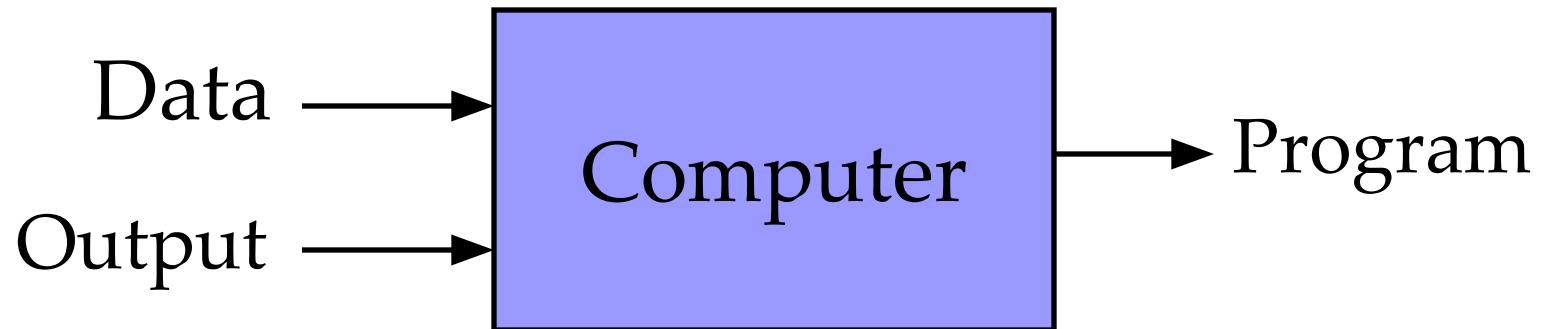
- Machine Learning (Mitchell 1997)
 - Learn from past experiences
 - Improve the performances of intelligent programs
- Definitions (Mitchell 1997)
 - A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at the tasks improves with the experiences



Traditional Programming



Machine Learning





Why Machine Learning

- Recent progress in algorithms and theory
- Growing flood of online data
- Computational power is available
- Budding industry
- **Three niches for machine learning:**
 - Data mining : using historical data to improve decisions
 - Software applications we can't program by hand
 - Self customizing programs { Newsreader that learns user interests



Why “Learn”?

- Machine learning is programming computers to optimize a performance criterion using example data or past experience.
- There is no need to “learn” to calculate payroll
- Learning is used when:
 - Human expertise does not exist (navigating on Mars),
 - Humans are unable to explain their expertise (speech recognition)
 - Solution changes in time (routing on a computer network)
 - Solution needs to be adapted to particular cases (user biometrics)



What We Talk About When We Talk About “Learning”

- Learning general models from a data of particular examples
- Data is cheap and abundant (data warehouses, data marts); knowledge is expensive and scarce.
- Example in retail: Customer transactions to consumer behavior:

People who bought “Da Vinci Code” also bought “The Five People You Meet in Heaven” (www.amazon.com)
- Build a model that is *a good and useful approximation* to the data.



What is the Learning Problem?

- Learning = Improving with experience at some task
 - Improve over task T ,
 - with respect to performance measure P ,
 - based on experience E.

- E.g., Learn to play checkers T : Play checkers P : % of games won in world tournament E: opportunity to play against self



Type of Training Experience

- Direct or indirect?
- Teacher or not?
- A problem: is training experience representative of performance goal?



Learning to Play Checkers

- T : Play checkers
- P : Percent of games won in world tournament
- What experience?
- What exactly should be learned?
- How shall it be represented?
- What specific algorithm to learn it?



Data Mining/KDD

Definition := “KDD is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data” (Fayyad)

Applications:

- **Retail:** Market basket analysis, Customer relationship management (CRM)
- **Finance:** Credit scoring, fraud detection
- **Manufacturing:** Optimization, troubleshooting
- **Medicine:** Medical diagnosis
- **Telecommunications:** Quality of service optimization
- **Bioinformatics:** Motifs, alignment
- **Web mining:** Search engines



What is Machine Learning?

- Machine Learning
 - Study of algorithms that
 - improve their performance
 - at some task
 - with experience
- Optimize a performance criterion using example data or past experience.
- Role of Statistics: Inference from a sample
- Role of Computer science: Efficient algorithms to
 - Solve the optimization problem
 - Representing and evaluating the model for inference



Growth of Machine Learning

- Machine learning is preferred approach to
 - Speech recognition, Natural language processing
 - Computer vision
 - Medical outcomes analysis
 - Robot control
 - Computational biology
- This trend is accelerating
 - Improved machine learning algorithms
 - Improved data capture, networking, faster computers
 - Software too complex to write by hand
 - New sensors / IO devices
 - Demand for self-customization to user, environment
 - It turns out to be difficult to extract knowledge from human experts→*failure of expert systems in the 1980's.*



Types of Learning

1. Supervised (inductive) learning

- Training data includes desired outputs

2. Unsupervised learning

- Training data does not include desired outputs

3. Semi-supervised learning

- Training data includes a few desired outputs

4. Rote learning

- One to One association

5. Reinforcement learning

- Rewards from sequence of actions



An example application

- An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.
- **A decision is needed:** whether to put a new patient in an intensive-care unit.
- Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.
- **Problem:** to predict **high-risk** patients and discriminate them from **low-risk** patients.



Another application

- A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,
 - age
 - Marital status
 - annual salary
 - outstanding debts
 - credit rating etc.
- **Problem:** to decide whether an application should be approved, or to classify applications into two categories, **approved** and **not approved**.



Machine learning and our focus

- Like human learning from past experiences.
- A computer does not have “experiences”.
- A computer system learns from data, which represent some “past experiences” of an application domain.

- Our focus: learn a target function that can be used to predict the values of a discrete class attribute, e.g., approve or not-approved, and high-risk or low risk.
- The task is commonly called: Supervised learning, classification, or inductive learning.



The data and the goal

- **Data:** A set of data records (also called examples, instances or cases) described by
 - k attributes: $A_1, A_2, \dots A_k$.
 - a class: Each example is labelled with a pre-defined class.
- **Goal:** To learn a classification model from the data that can be used to predict the classes of new (future, or test) cases/instances.



An example: data (loan application)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No



An example: the learning task

- Learn a classification model from the data
- Use the model to classify future loan applications into
 - Yes (approved) and
 - No (not approved)
- What is the class for following case/instance?

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?



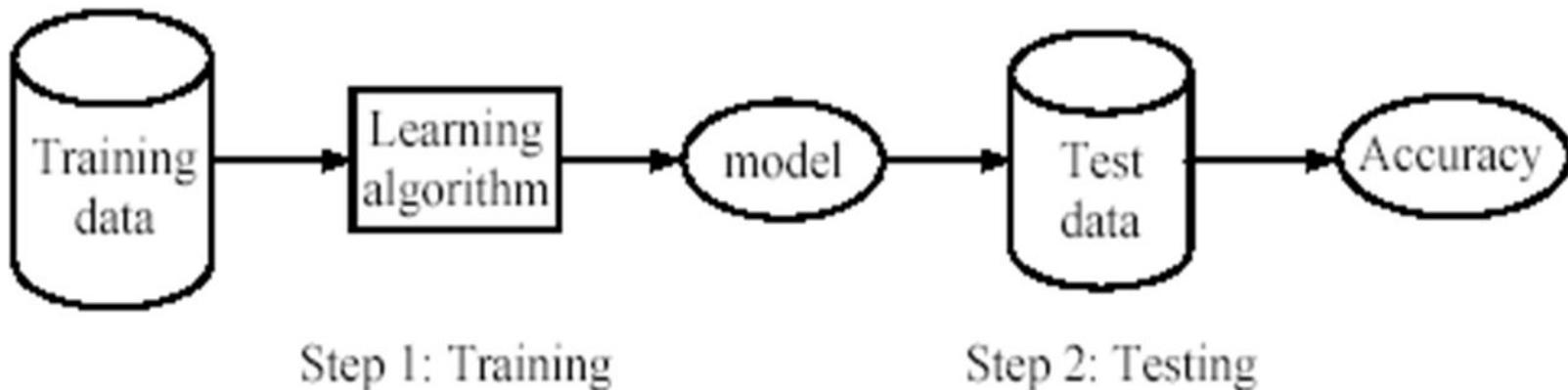
Supervised vs. unsupervised Learning

- **Supervised learning:** classification is seen as supervised learning from examples.
 - **Supervision:** The data (observations, measurements, etc.) are labeled with predefined classes. It is like that a “teacher” gives the classes (**supervision**).
 - Test data are classified into these classes too.
- **Unsupervised learning (clustering)**
 - **Class labels of the data are unknown**
 - Given a set of data, the task is to establish the existence of classes or clusters in the data

Supervised learning process: two steps

- **Learning (training):** Learn a model using the **training data**
- **Testing:** Test the model using **unseen test data** to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$





Supervised learning categories and techniques

- **Linear classifier** (numerical functions)
- **Parametric** (Probabilistic functions)
 - Naïve Bayes, Gaussian discriminant analysis (GDA), Hidden Markov models (HMM), Probabilistic graphical models
- **Non-parametric** (Instance-based functions)
 - K -nearest neighbors, Kernel regression, Kernel density estimation, Local regression
- **Non-metric** (Symbolic functions)
 - Classification and regression tree (CART), decision tree
- **Aggregation**
 - Bagging (bootstrap + aggregation), Adaboost, Random forest



Unsupervised learning

- The data have no target attribute.
 - We want to explore the data to find some intrinsic structures in them.
- **Unsupervised Learning:** Learning useful structure *without* labeled classes, optimization criterion, feedback signal, or any other information beyond the raw data



Examples:

- Find natural groupings of Xs (X=human languages, stocks, gene sequences, animal species,...)→ Prelude to discovery of underlying properties
- Summarize the news for the past month→ Cluster first, then report centroids.
- Sequence extrapolation: E.g. Predict cancer incidence next decade; predict rise in antibiotic-resistant bacteria



Unsupervised learning categories and techniques

- Unsupervised learning categories and techniques
 - **Clustering**
 - K-means clustering
 - Spectral clustering
 - **Density Estimation**
 - Gaussian mixture model (GMM)
 - Graphical models
 - **Dimensionality reduction**
 - Principal component analysis (PCA)
 - Factor analysis



Rote learning

- **Rote learning** – One-to-one mapping from inputs to stored representation. “Learning by memorization.”
Association-based storage and retrieval.



Reinforcement learning

- In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem.
- After a set of trial-and error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward.



Example

- Let us say we want to build a machine that learns to play chess.
- In this case we cannot use a supervised learner for two reasons. First, it is very costly to have a teacher that will take us through many games and indicate us the best move for each position.
- Second, in many cases, there is no such thing as the best move; the goodness of a move depends on the moves that follow.
- A single move does not count; a sequence of moves is good if after playing them we win the game. The only feedback is at the end of the game when we win or lose the game.



The agent interacts with an environment

- There is decision maker, called the *agent*, that is placed in an *environment*.
- In chess, the game-player is the decision maker and the environment is the board.





Learning agent

- Once an action is chosen and taken, the state changes.
- The solution to the task requires a sequence of actions, and we get feedback, in the form of a *reward* rarely.
- The reward defines the problem and is necessary if we want a *learning* agent.
- The learning agent learns the best sequence of actions to solve a problem where “best” is quantified as the sequence of actions that has the maximum cumulative reward. Such is the setting of *reinforcement learning*.



Reinforcement learning

- Reinforcement learning is different from the Other learning methods.
- It is called “learning with a critic,” as opposed to learning with a teacher which we have in supervised learning.
- *A critic differs from a teacher in that it does not tell us what to do* but only how well we have been doing in the past; the critic never informs in advance.



Reinforcement learning

- Though reinforcement learning algorithms are slower than supervised learning algorithms, it is clear that they have a wider variety of application and have the potential to construct better learning machines (Ballard 1997).
- They do not need any supervision, and this may actually be better since then they are not biased by the teacher.
- The field of reinforcement learning is developing rapidly.



Applications

- Association Analysis
- Supervised Learning
 - Classification
 - Regression/Prediction
- Unsupervised Learning
- Reinforcement Learning



Learning Associations

- Basket analysis:

$P(Y | X)$ probability that somebody who buys X also buys Y where X and Y are products/services.

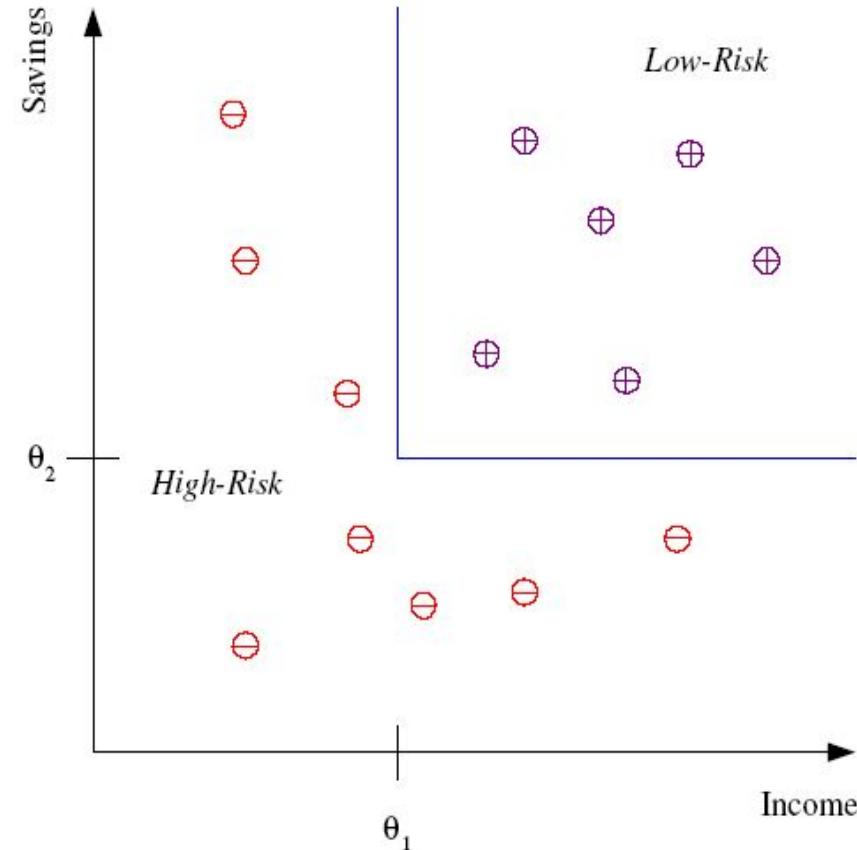
Example: $P(\text{chips} | \text{beer}) = 0.7$

Market-Basket transactions

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Classification

- Example: Credit scoring
- Differentiating between **low-risk** and **high-risk** customers from their *income* and *savings*



Discriminant: IF $\text{income} > \theta_1$ AND $\text{savings} > \theta_2$
THEN **low-risk** ELSE **high-risk**

Model



Classification: Applications

- Aka Pattern recognition
- Face recognition: Pose, lighting, occlusion (glasses, beard), make-up, hair style
- Character recognition: Different handwriting styles.
- Speech recognition: Temporal dependency.
 - Use of a dictionary or the syntax of the language.
 - Sensor fusion: Combine multiple modalities; eg, visual (lip image) and acoustic for speech
- Medical diagnosis: From symptoms to illnesses
- Web Advertising: Predict if a user clicks on an ad on the Internet.



Face Recognition

Training examples of a person



Test images

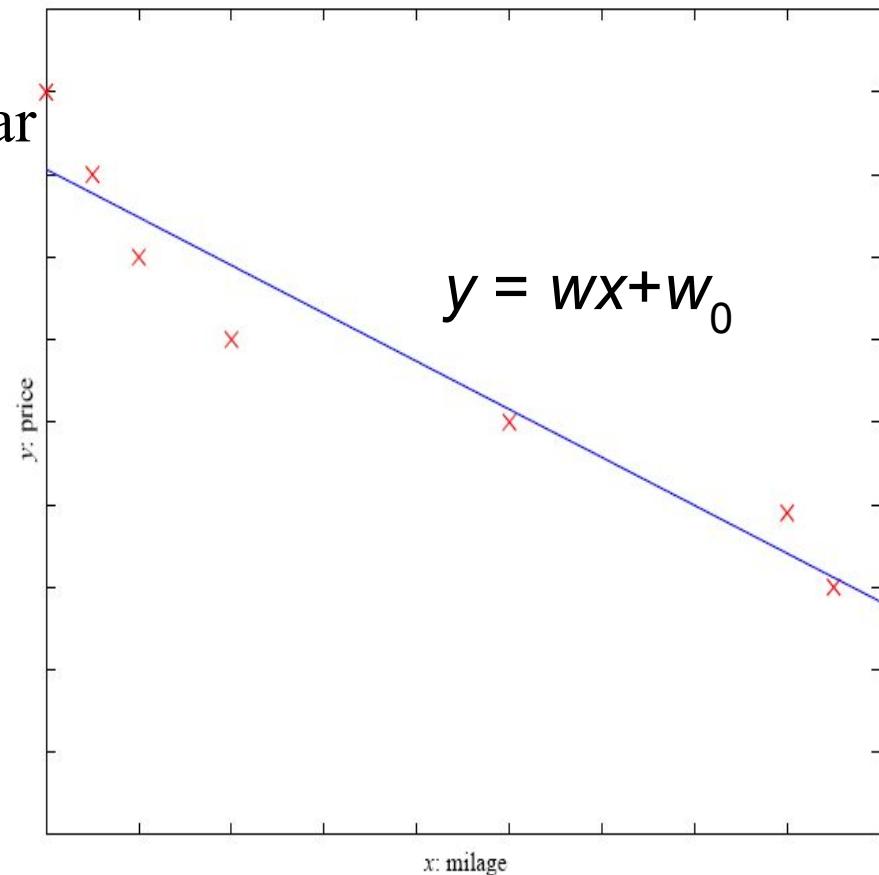


AT&T Laboratories, Cambridge UK
<http://www.uk.research.att.com/facedatabase.html>



Prediction: Regression

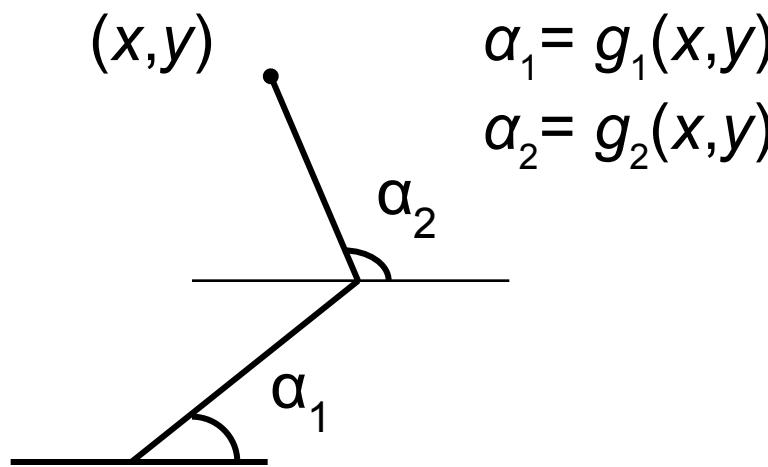
- Example: Price of a used car
- x : car attributes
- y : price
- $y = g(x | \theta)$
- $g(\cdot)$ model,
- θ parameters





Regression Applications

- Navigating a car: Angle of the steering wheel
- Kinematics of a robot arm





Supervised Learning: Uses

Example: decision trees tools that create rules

- Prediction of future cases: Use the rule to predict the output for future inputs
- Knowledge extraction: The rule is easy to understand
- Compression: The rule is simpler than the data it explains
- Outlier detection: Exceptions that are not covered by the rule, e.g., fraud



Unsupervised Learning

- Learning “what normally happens”
- No output
- Clustering: Grouping similar instances
- Other applications: Summarization, Association Analysis
- Example applications
 - Customer segmentation in CRM
 - Image compression: Color quantization
 - Bioinformatics: Learning motifs



Reinforcement Learning

- Policies: what actions should an agent take in a particular situation
- Utility estimation: how good is a state (\rightarrow used by policy)
- No supervised output but delayed reward
- Credit assignment problem (what was responsible for the outcome)
- Applications: Game playing, Robot in a maze, Multiple agents, partial observability,



Some successful applications of machine learning

- Learning to recognize spoken words.
- Learning to drive an autonomous vehicle.
- Learning to classify new astronomical structures.
- Learning to play world-class backgammon.



Symbolic vs. Statistical Learning

- **Symbolic Learning:**
 - underlying learning strategies such as rote learning, learning by being told, learning by analogy, learning from examples and learning from discovery.
 - Examples: DTrees
- **Statistical Learning: modeling of the behavior**
 - Sometimes called statistical inference, connections tic learning
 - Example: Artificial neuronal networks, bayesian learning



Designing a Learning System

- **learning system:** A computer program is said to learn from *experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*
- i.e. a checkers learning problem
- T: playing checkers
- P: percent of games won against opponents
- E: playing practice games against itself



Steps for Designing a Learning System

Choosing the Training Experience

Choosing the Target Function

Choosing a Representation for the Target Function .

Choosing a Function Approximation Algorithm

ESTIMATING TRAINING VALUES

ADJUSTING THE WEIGHTS

The Final Design



Designing a Learning System

- Consider designing a program to learn to play checkers in order to illustrate some of the basic design issues and approaches to machine learning
- **Choosing the training experience**



1. Choosing the training experience

1. The type of training experience available can have a significant impact on success or failure of the learner.
 - One key attribute : the training experience provides direct or indirect feedback.
2. A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples.
 - the learner might rely on the teacher to select informative board states and to provide the correct move for each.



1. Choosing the training experience

3. A third important attribute of the training experience is how well it represents the distribution of examples over which the final system performance P must be measured.
 - The learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion.
 - In order to complete the design of the learning system,
 - 1. the exact type of knowledge to be learned
 - 2. a representation for this target knowledge
 - 3. a learning mechanism



2. Choosing the Target Function

1. Determine what type of knowledge will be learned
 - Most obvious form is a function, that chooses the best move for any given board state
 - i.e., $\text{ChooseMove} : \mathcal{B} \rightarrow \mathcal{M}$
2. sometimes evaluation functions are easier to learn
 - i.e., $V : \mathcal{B} \rightarrow \mathcal{R}$
3. Assigns higher values to better boards states by recursively generating the successor states and evaluating them until the game is over.
4. **⇒ function approximation is sufficient (operational definition)**



2. Choosing the Target Function

- Let us therefore define the target value
- $V(b)$ for an arbitrary board state b in B , as follows:
 - 1. if b is a final board state that is won, then $V(b) = 100$
 - 2. if b is a final board state that is lost, then $V(b) = -100$
 - 3. if b is a final board state that is drawn, then $V(b) = 0$
- if b is not a final state in the game, then $V(b) = V(b')$, where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.



3. Choosing a Representation for the Target Function

- The ideal target function V , then must choose a representation that the learning program will use to describe the function c that it will learn.
- the function V will be calculated as a linear combination of the following board features:
 - x_1 : the number of black pieces on the board
 - x_2 : the number of red pieces on the board
 - x_3 : the number of black kings on the board
 - x_4 : *the number of red kings on the board*
 - x_5 : *the number of black pieces threatened by red*
 - X_6 : *the number of red pieces threatened by black*



4. Choosing a Function Approximation Algorithm

- In order to learn the target function V we require a set of training examples, each describing a specific board state b and the training value $V_{\text{train}}(b)$ for b .
- estimate
- we require training examples that assign specific scores to specific board state.
- Adjust
- All that remains is to specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{\text{train}}(b))\}$



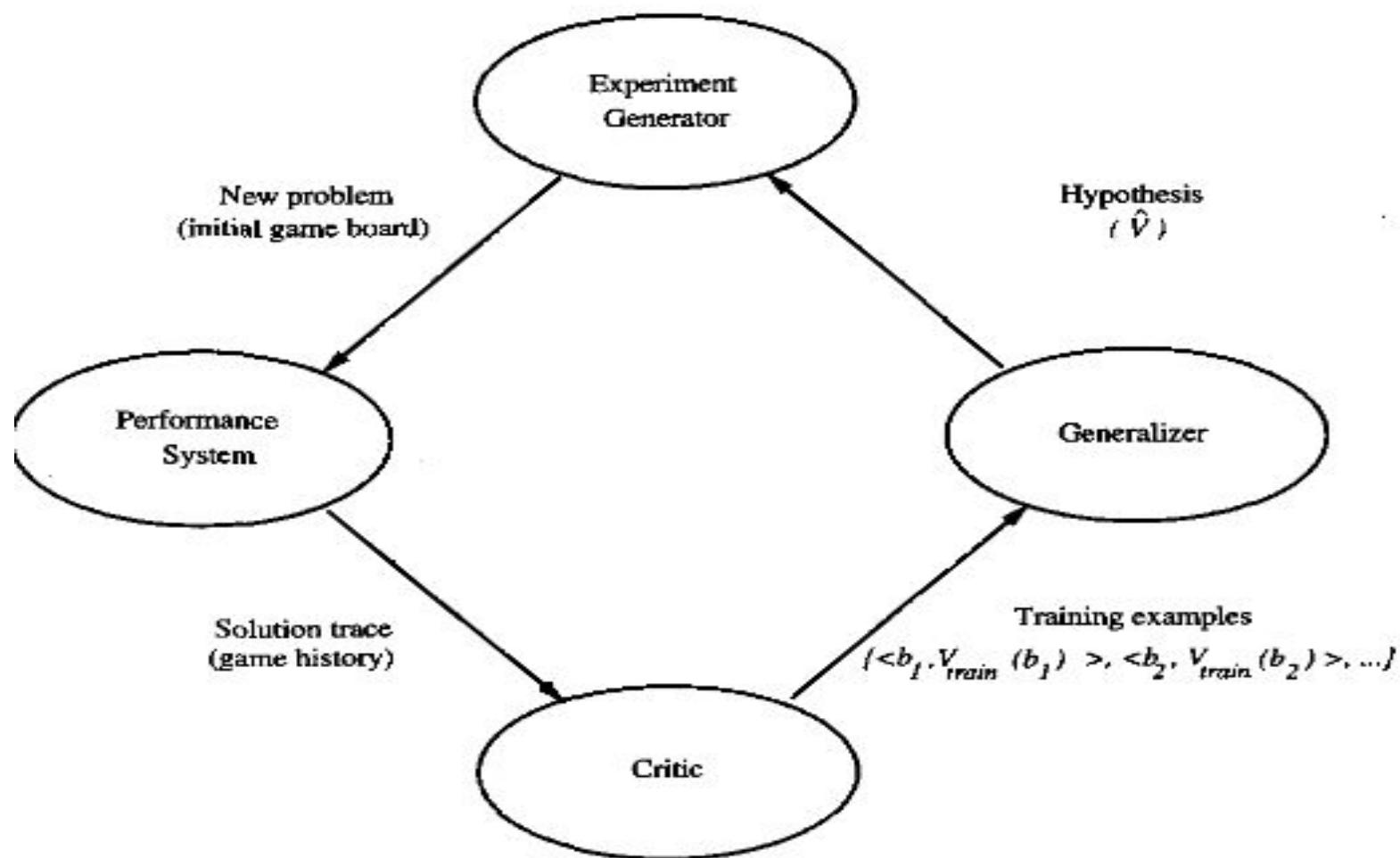
Final design

- **The Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s).
 - It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.
- **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function.

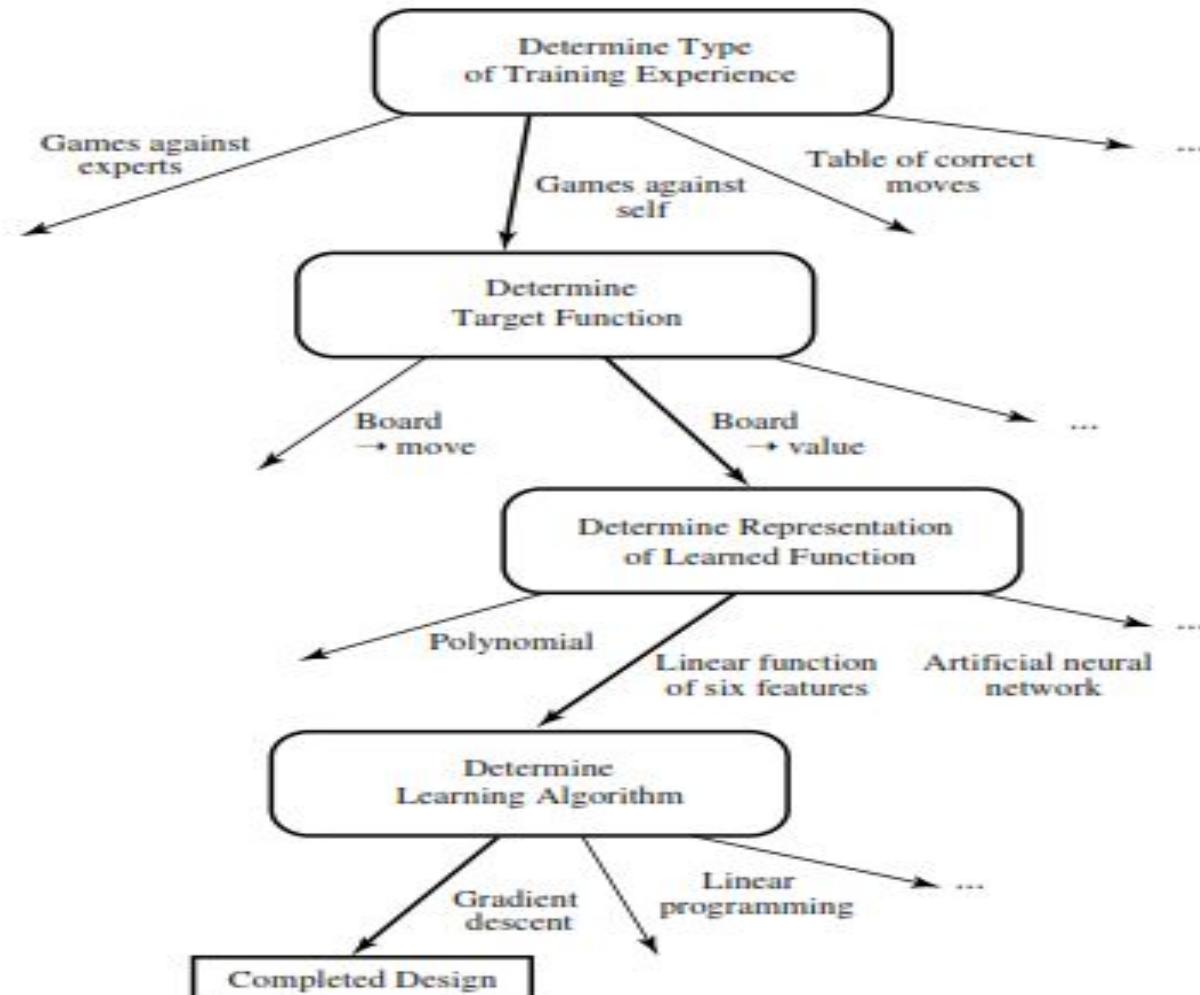


- **The Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function.
- **The Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem

5. Final Design



Designing a Learning System





PERSPECTIVES AND ISSUES IN MACHINE LEARNING

- One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner.



Issues in Machine Learning

- What algorithms exist for learning general target functions from specific training examples?
- How much training data is sufficient?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples?
- What is the best strategy for choosing a useful next training experience?
- What is the best way to reduce the learning task to one or more function approximation problems?
- How can the learner automatically alter **its representation to improve its** ability to represent and learn the target function?

What is a Means End Analysis?

Means End Analysis (MEA) is a problem-solving technique that has been used since the fifties of the last century to stimulate **creativity**. Means End Analysis is also a way of looking at the **organisational planning**, and helps in achieving the **end-goals**.

With Means End Analysis, it is possible to control the entire process of problem solving. It starts from a predetermined goal, in which actions are chosen that lead to that goal.

Each action that is executed leads to the next action; everything is connected together in order to reach the end-goal. In the meantime however, problems may arise. It is often hard to determine where exactly the crux is.

With the help of Means End Analysis, both forward and backward research can be done to determine where the stagnation is occurring. This enables the larger parts of a problem to be solved first, to subsequently return to the smaller problems afterwards.

Intermediate steps

In order for Means End Analysis to be effective, it is advisable to get all relevant **actions and intermediate steps** leading to the goal in the picture, making them detectable.

Additionally, it is handy to be capable of tracking (small) changes, and to **measure** the differences between the actual and desired state of the individual actions.

If this doesn't happen, there is a significant risk that a mistake or change will have further consequences across the series of actions following it, making it harder and harder to intervene.

Goals

Every organisation works with **goals** that need to be met.

Depending on the goal a short term (a week or a month), mid-long term (a year), and a long term (multiple years) are determined. It is nice both for the organisation and for the employees when these goals are successfully met.

By making an analysis of the means and the intermediate actions with the help of Means End Analysis beforehand, it is easier to focus and not lose your way. It is a fact that goals don't just achieve themselves. Based on careful **planning**, action should be undertaken.

Without planning there's a significant chance for the organisation to head in the wrong direction, deviating from its pre-determined goal.

Means End Analysis example

To successfully execute Means End Analysis it is advisable to think from large to small; the eventual goal needs to be split into smaller sub-goals, making it overseable for all parties that are going to work towards on achieving it.

When a commercial electronic business has the end-goal to reach a turnover of 15 million euro's within a year, that is a noble thought. It means that all actions in that year will be geared towards meeting that 15 million euro limit.

However, it will only work when it becomes clear what has to be done to meet that turnover of 15 million. With the help of Means End Analysis, the end goal is split into a few smaller goals, which will contribute to the 15 million turnover:

- A specific product, for example the newest smartphone, needs to be sold aggressively;
- A minimal selling price is set, which dealers also must comply with;
- Aside from the newest smartphone, there are some related products that will be go to market as well.

Means End Analysis : Executable steps

Regardless of the splitting into smaller sub-goals, it will still not be possible for the organisation to achieve a turnover of 15 million. The search for even smaller, more specific steps, aids in them to achieving the end-goal.

These sub-sub-goals are translated into executable steps that are deployed by the organisation and used to achieve the original goal of a turnover of 15 million. In case there is stagnation of a problem somewhere, it becomes much easier to find the problem and fix that part of the process. Prior sub-goals are elaborated upon below:

- A specific **marketing plan** is developed for the smartphone to give publicity to the new product, especially via social media;
- **New applications** will be developed by the electronic business to be sold as a by-product;
- A special discount is offered to students when they can prove that they are, in fact, registered at an institute of higher education;
- An advertisement will be placed in door-to-door newspapers, whereby a coupon can be used to obtain a substantial trade-in discount for the old mobile phone.

Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity. Constraint satisfaction depends on three components, namely:

- **X:** It is a set of variables.
- **D:** It is a set of domains where the variables reside. There is a specific domain for each variable.
- **C:** It is a set of constraints which are followed by the set of variables.

In constraint satisfaction, domains are the spaces where the variables reside, following the problem specific constraints. These are the three main elements of a constraint satisfaction technique. The constraint value consists of a pair of **{scope, rel}**. The **scope** is a tuple of variables which participate in the constraint

and **rel** is a relation which includes a list of values which the variables can take to satisfy the constraints of the problem.

Solving Constraint Satisfaction Problems

The requirements to solve a constraint satisfaction problem (CSP) is:

- A state-space
- The notion of the solution.

A state in state-space is defined by assigning values to some or all variables such as

{X1=v1, X2=v2, and so on...}.

An assignment of values to a variable can be done in three ways:

- **Consistent or Legal Assignment:** An assignment which does not violate any constraint or rule is called Consistent or legal assignment.
- **Complete Assignment:** An assignment where every variable is assigned with a value, and the solution to the CSP remains consistent. Such assignment is known as Complete assignment.
- **Partial Assignment:** An assignment which assigns values to some of the variables only. Such type of assignments are called Partial assignments.

Types of Domains in CSP

There are following two types of domains which are used by the variables :

- **Discrete Domain:** It is an infinite domain which can have one state for multiple variables. **For example**, a start state can be allocated infinite times for each variable.
- **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

Constraint Types in CSP

With respect to the variables, basically there are following types of constraints:

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.

- **Binary Constraints:** It is the constraint type which relates two variables. A value x_2 will contain a value which lies between x_1 and x_3 .
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

Some special types of solution algorithms are used to solve the following types of constraints:

- **Linear Constraints:** These type of constraints are commonly used in linear programming where each variable containing an integer value exists in linear form only.
- **Non-linear Constraints:** These type of constraints are used in non-linear programming where each variable (an integer value) exists in a non-linear form.

Note: A special constraint which works in real-world is known as **Preference constraint**.

Constraint Propagation

In local state-spaces, the choice is only one, i.e., to search for a solution. But in CSP, we have two choices either:

- We can search for a solution or
- We can perform a special type of inference called **constraint propagation**.

Constraint propagation is a special type of inference which helps in reducing the legal number of values for the variables. The idea behind constraint propagation is **local consistency**.

In local consistency, variables are treated as **nodes**, and each binary constraint is treated as an **arc** in the given problem. **There are following local consistencies which are discussed below:**

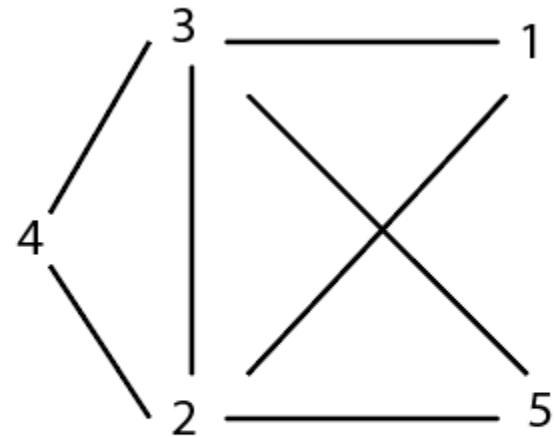
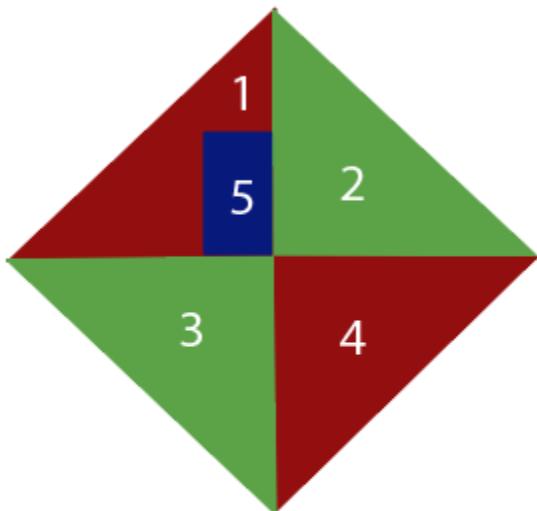
- **Node Consistency:** A single variable is said to be node consistent if all the values in the variable's domain satisfy the unary constraints on the variables.

- **Arc Consistency:** A variable is arc consistent if every value in its domain satisfies the binary constraints of the variables.
- **Path Consistency:** When the evaluation of a set of two variable with respect to a third variable can be extended over another variable, satisfying all the binary constraints. It is similar to arc consistency.
- **k-consistency:** This type of consistency is used to define the notion of stronger forms of propagation. Here, we examine the k-consistency of the variables.

CSP Problems

Constraint satisfaction includes those problems which contains some constraints while solving the problem. CSP includes the following problems:

- **Graph Coloring:** The problem where the constraint is that no adjacent sides can have the same color.



Graph Coloring

- **Sudoku Playing:** The gameplay where the constraint is that no number from 0-9 can be repeated in the same row or column.

SUDOKU

4							5	9
2	6		5				3	
			9	2				
		2		6			1	
		3	8	1	9	7		
7			3		5			
			3	4				
3				6		2	7	
5	9						6	

Puzzle

4	1	7	6	8	3	2	5	9
2	6	9	5	7	1	8	3	4
3	8	5	4	9	2	6	7	1
8	4	2	7	6	5	9	1	3
6	5	3	8	1	9	7	4	2
9	7	1	2	3	4	5	6	8
7	2	6	3	4	8	1	9	5
1	3	8	9	5	6	4	2	7
5	9	4	1	2	7	3	8	6

Solution

- **n-queen problem:** In n-queen problem, the constraint is that no queen should be placed either diagonally, in the same row or column.

Note: The n-queen problem is already discussed in Problem-solving in AI section.

- **Crossword:** In crossword problem, the constraint is that there should be the correct formation of the words, and it should be meaningful.



- **Latin square Problem:** In this game, the task is to search the pattern which is occurring several times in the game. They may be shuffled but will contain the same digits.

1	1	1		1	1	1	1
1	2	3	4	1	2	3	4
1	3	4	2	1	5	4	3
1	4	2	3	1	4	3	5
1	2	4	3	1	3	2	5

Latin Squence Problem

- **Cryptarithmetic Problem:** This problem has one most important constraint that is, we cannot assign a different digit to the same character. All digits should contain a unique alphabet.

Problem Reduction

the divide and conquer strategy, a solution to a problem can be obtained by decomposing it into smaller sub-problems. Each of this sub-problem can then be solved to get its sub solution. These sub solutions can then be recombined to get a solution as a whole. That is called is Problem Reduction. This method generates arc which is called as AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved for an arc to point to a solution.

Algorithm

1. Initialize the graph to the starting node.
2. Loop until the starting node is labelled SOLVED or until its cost goes above FUTILITY:
 - (i) Traverse the graph, starting at the initial node and following the current best path and accumulate the set of nodes that are on that path and have not yet been expanded.
 - (ii) Pick one of these unexpanded nodes and expand it. If there are no successors, assign FUTILITY as the value of this node. Otherwise, add its successors to the graph and for each of them compute $f'(n)$. If $f'(n)$ of any node is 0, mark that node as SOLVED.
 - (iii) Change the $f'(n)$ estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backwards through the graph. If any node contains a successor arc whose descendants are all solved, label the node itself as SOLVED.

Principal Component Analysis | Dimension Reduction

Pattern Recognition

Dimension Reduction-

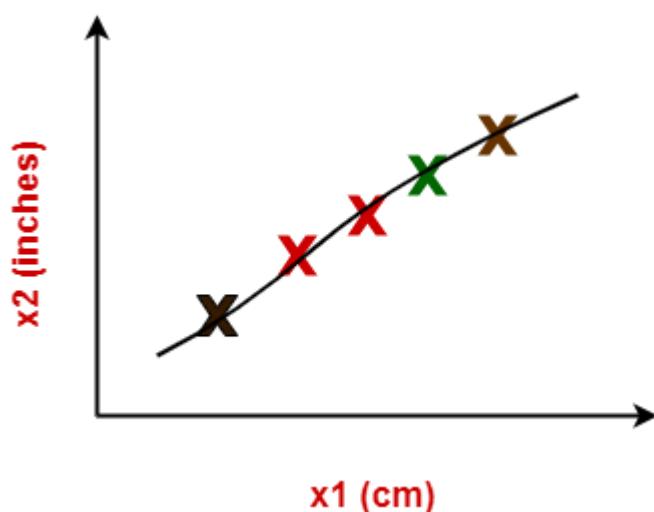
In pattern recognition, Dimension Reduction is defined as-

- It is a process of converting a data set having vast dimensions into a data set with lesser dimensions.
- It ensures that the converted data set conveys similar information concisely.

Example-

Consider the following example-

- The following graph shows two dimensions x_1 and x_2 .
- x_1 represents the measurement of several objects in cm.
- x_2 represents the measurement of several objects in inches.

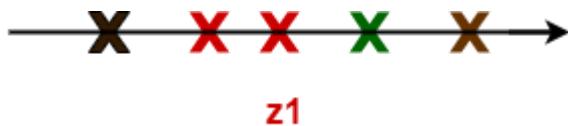


In machine learning,

- Using both these dimensions convey similar information.
- Also, they introduce a lot of noise in the system.
- So, it is better to use just one dimension.

Using dimension reduction techniques-

- We convert the dimensions of data from 2 dimensions (x_1 and x_2) to 1 dimension (z_1).
- It makes the data relatively easier to explain.



Benefits-

Dimension reduction offers several benefits such as-

- It compresses the data and thus reduces the storage space requirements.
- It reduces the time required for computation since less dimensions require less computation.
- It eliminates the redundant features.
- It improves the model performance.

Dimension Reduction Techniques-

The two popular and well-known dimension reduction techniques are-



1. Principal Component Analysis (PCA)
2. Fisher Linear Discriminant Analysis (LDA)

Principal Component Analysis-

- Principal Component Analysis is a well-known dimension reduction technique.

- It transforms the variables into a new set of variables called as principal components.
- These principal components are linear combination of original variables and are orthogonal.
- The first principal component accounts for most of the possible variation of original data.
- The second principal component does its best to capture the variance in the data.
- There can be only two principal components for a two-dimensional data set.

PCA Algorithm-

The steps involved in PCA Algorithm are as follows-

Step-01: Get data.

Step-02: Compute the mean vector (μ).

Step-03: Subtract mean from the given data.

Step-04: Calculate the covariance matrix.

Step-05: Calculate the eigen vectors and eigen values of the covariance matrix.

Step-06: Choosing components and forming a feature vector.

Step-07: Deriving the new data set.

PRACTICE PROBLEMS BASED ON PRINCIPAL COMPONENT ANALYSIS-

Problem-01:

Given data = { 2, 3, 4, 5, 6, 7 ; 1, 5, 3, 6, 7, 8 }.

Compute the principal component using PCA Algorithm.

OR

Consider the two dimensional patterns (2, 1), (3, 5), (4, 3), (5, 6), (6, 7), (7, 8).

Compute the principal component using PCA Algorithm.

OR

Compute the principal component of following data-

CLASS 1

$$X = 2, 3, 4$$

$$Y = 1, 5, 3$$

CLASS 2

$$X = 5, 6, 7$$

$$Y = 6, 7, 8$$

Solution-

We use the above discussed PCA Algorithm-

Step-01:

Get data.

The given feature vectors are-

- $x_1 = (2, 1)$
- $x_2 = (3, 5)$
- $x_3 = (4, 3)$
- $x_4 = (5, 6)$
- $x_5 = (6, 7)$
- $x_6 = (7, 8)$

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \begin{bmatrix} 3 \\ 5 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix}$$

Step-02:

Calculate the mean vector (μ).

Mean vector (μ)

$$= ((2 + 3 + 4 + 5 + 6 + 7) / 6, (1 + 5 + 3 + 6 + 7 + 8) / 6)$$
$$= (4.5, 5)$$

Thus,

Mean vector (μ) =
$$\begin{bmatrix} 4.5 \\ 5 \end{bmatrix}$$

Step-03:

Subtract mean vector (μ) from the given feature vectors.

- $x_1 - \mu = (2 - 4.5, 1 - 5) = (-2.5, -4)$
- $x_2 - \mu = (3 - 4.5, 5 - 5) = (-1.5, 0)$
- $x_3 - \mu = (4 - 4.5, 3 - 5) = (-0.5, -2)$
- $x_4 - \mu = (5 - 4.5, 6 - 5) = (0.5, 1)$
- $x_5 - \mu = (6 - 4.5, 7 - 5) = (1.5, 2)$
- $x_6 - \mu = (7 - 4.5, 8 - 5) = (2.5, 3)$

Feature vectors (x_i) after subtracting mean vector (μ) are-

$$\begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \begin{bmatrix} -1.5 \\ 0 \end{bmatrix} \begin{bmatrix} -0.5 \\ -2 \end{bmatrix} \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \begin{bmatrix} 1.5 \\ 2 \end{bmatrix} \begin{bmatrix} 2.5 \\ 3 \end{bmatrix}$$

Step-04:

Calculate the covariance matrix.

Covariance matrix is given by-

$$\text{Covariance Matrix} = \frac{\sum (x_i - \mu)(x_i - \mu)^t}{n}$$

Now,

$$m_1 = (x_1 - \mu)(x_1 - \mu)^t = \begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \begin{bmatrix} -2.5 & -4 \end{bmatrix} = \begin{bmatrix} 6.25 & 10 \\ 10 & 16 \end{bmatrix}$$

$$m_2 = (x_2 - \mu)(x_2 - \mu)^t = \begin{bmatrix} -1.5 \\ 0 \end{bmatrix} \begin{bmatrix} -1.5 & 0 \end{bmatrix} = \begin{bmatrix} 2.25 & 0 \\ 0 & 0 \end{bmatrix}$$

$$m_3 = (x_3 - \mu)(x_3 - \mu)^t = \begin{bmatrix} -0.5 \\ -2 \end{bmatrix} \begin{bmatrix} -0.5 & -2 \end{bmatrix} = \begin{bmatrix} 0.25 & 1 \\ 1 & 4 \end{bmatrix}$$

$$m_4 = (x_4 - \mu)(x_4 - \mu)^t = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \begin{bmatrix} 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$m_5 = (x_5 - \mu)(x_5 - \mu)^t = \begin{bmatrix} 1.5 \\ 2 \end{bmatrix} \begin{bmatrix} 1.5 & 2 \end{bmatrix} = \begin{bmatrix} 2.25 & 3 \\ 3 & 4 \end{bmatrix}$$

$$m_6 = (x_6 - \mu)(x_6 - \mu)^t = \begin{bmatrix} 2.5 \\ 3 \end{bmatrix} \begin{bmatrix} 2.5 & 3 \end{bmatrix} = \begin{bmatrix} 6.25 & 7.5 \\ 7.5 & 9 \end{bmatrix}$$

Now,

Covariance matrix

$$= (m_1 + m_2 + m_3 + m_4 + m_5 + m_6) / 6$$

On adding the above matrices and dividing by 6, we get-

$$\text{Covariance Matrix} = \frac{1}{6} \begin{bmatrix} 17.5 & 22 \\ 22 & 34 \end{bmatrix}$$

$$\text{Covariance Matrix} = \begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix}$$

Step-05:

Calculate the eigen values and eigen vectors of the covariance matrix.

λ is an eigen value for a matrix M if it is a solution of the characteristic equation $|M - \lambda I| = 0$.

So, we have-

$$\begin{vmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{vmatrix} - \begin{vmatrix} \lambda & 0 \\ 0 & \lambda \end{vmatrix} = 0$$

$$\begin{vmatrix} 2.92 - \lambda & 3.67 \\ 3.67 & 5.67 - \lambda \end{vmatrix} = 0$$

From here,

$$(2.92 - \lambda)(5.67 - \lambda) - (3.67 \times 3.67) = 0$$

$$16.56 - 2.92\lambda - 5.67\lambda + \lambda^2 - 13.47 = 0$$

$$\lambda^2 - 8.59\lambda + 3.09 = 0$$

Solving this quadratic equation, we get $\lambda = 8.22, 0.38$

Thus, two eigen values are $\lambda_1 = 8.22$ and $\lambda_2 = 0.38$.

Clearly, the second eigen value is very small compared to the first eigen value.

So, the second eigen vector can be left out.

Eigen vector corresponding to the greatest eigen value is the principal component for the given data set.

So. we find the eigen vector corresponding to eigen value λ_1 .

We use the following equation to find the eigen vector-

$$MX = \lambda X$$

where-

- M = Covariance Matrix
- X = Eigen vector
- λ = Eigen value

Substituting the values in the above equation, we get-

$$\begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = 8.22 \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

Solving these, we get-

$$2.92X_1 + 3.67X_2 = 8.22X_1$$

$$3.67X_1 + 5.67X_2 = 8.22X_2$$

On simplification, we get-

$$5.3X_1 = 3.67X_2 \dots\dots\dots(1)$$

$$3.67X_1 = 2.55X_2 \dots\dots\dots(2)$$

From (1) and (2), $X_1 = 0.69X_2$

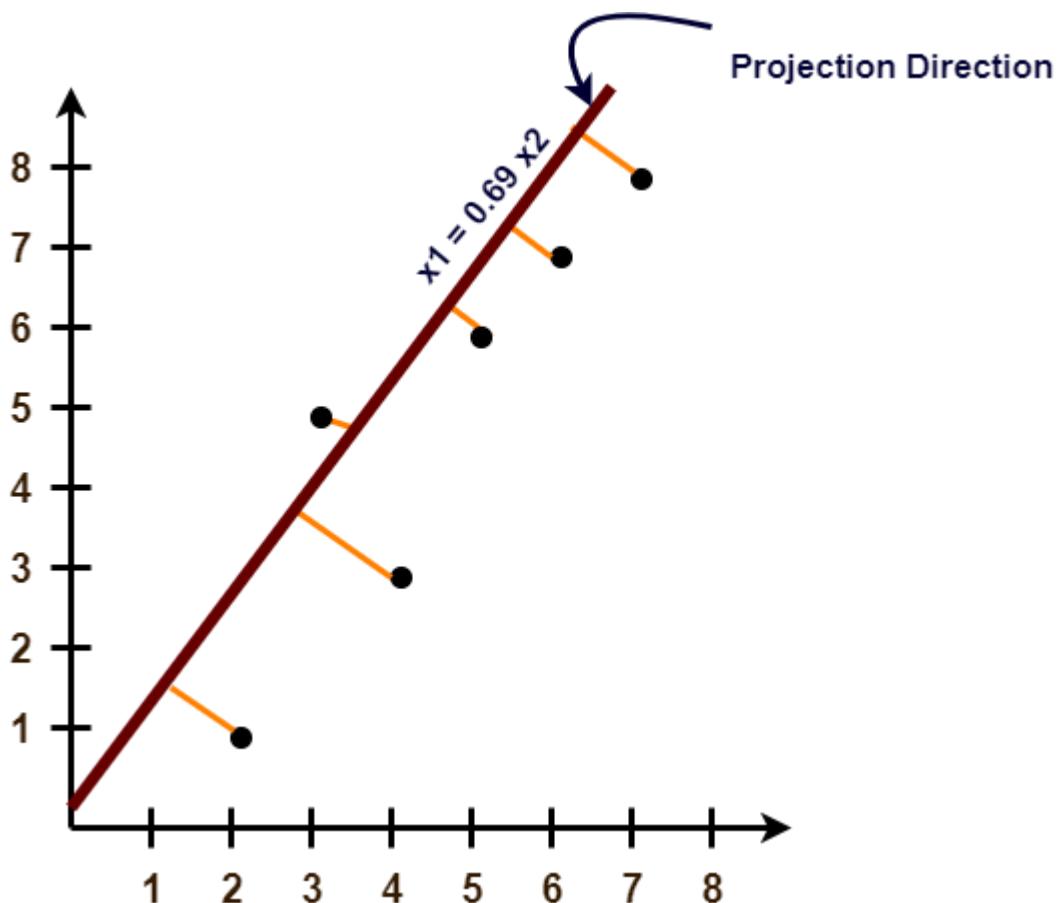
From (2), the eigen vector is-

Eigen Vector : $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$

Thus, principal component for the given data set is-

Principal Component : $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$

Lastly, we project the data points onto the new subspace as-



Problem-02:

Use PCA Algorithm to transform the pattern (2, 1) onto the eigen vector in the previous question.

Solution-

The given feature vector is (2, 1).

Given Feature Vector :
$$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

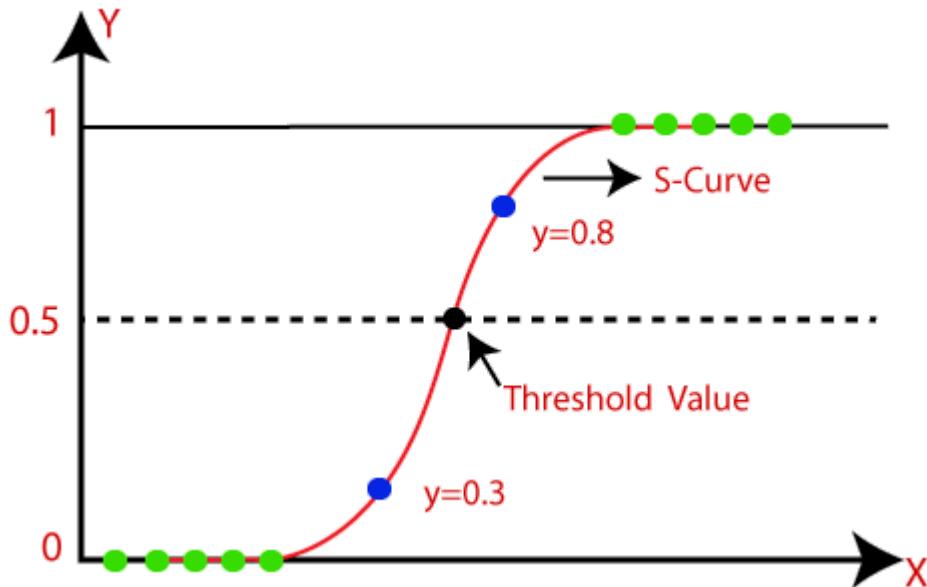
The feature vector gets transformed to

= Transpose of Eigen vector x (Feature Vector – Mean Vector)

$$\begin{aligned} &= \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}^T \times \left(\begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 4.5 \\ 5 \end{bmatrix} \right) \\ &= \begin{bmatrix} 2.55 & 3.67 \end{bmatrix} \times \begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \\ &= -21.055 \end{aligned}$$

Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1.**
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems.**
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Note: Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by $(1-y)$:

$$\frac{y}{1-y}; 0 \text{ for } y=0, \text{ and infinity for } y=1$$

- But we need range between $-\infty$ to $+\infty$, then take logarithm of the equation it will become:

$$\log \left[\frac{y}{1-y} \right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Python Implementation of Logistic Regression (Binomial)

To understand the implementation of Logistic Regression in Python, we will use the below example:

Competitive questions on Structures in HindiKeep Watching

Example: There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

For this problem, we will build a Machine Learning model using the Logistic regression algorithm. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	13000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

- Data Pre-processing step
- Fitting Logistic Regression to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

1. Data Pre-processing step: In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

1. #Data Pre-procesing Step

```

2. # importing libraries
3. import numpy as nm
4. import matplotlib.pyplot as mtp
5. import pandas as pd
6.
7. #importing datasets
8. data_set= pd.read_csv('user_data.csv')

```

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

1. #Extracting Independent and dependent Variable
2. x= data_set.iloc[:, [2,3]].values

3. `y= data_set.iloc[:, 4].values`

In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4. The output will be:

The image shows two side-by-side data visualization windows. The left window is titled "x - NumPy array" and displays a 2D table with two columns labeled 0 and 1. The first column contains ages (e.g., 19, 35, 26, 27, 19, 27, 27, 32, 25, 35, 26, 26, 20) and the second column contains salaries (e.g., 19000, 20000, 43000, 57000, 76000, 58000, 84000, 150000, 33000, 65000, 80000, 52000, 86000). The right window is titled "y - NumPy array" and displays a 1D array of 13 elements, all of which are 0 except for element 7 which is 1.

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000
10	26	80000
11	26	52000
12	20	86000

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Now we will split the dataset into a training set and test set. Below is the code for it:

1. # Splitting the dataset into training and test set.
2. from sklearn.model_selection import train_test_split
- 3.

```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0 )
```

The output for this is given below:

For

test

set:

The image shows two side-by-side data visualization windows. The left window is titled "x_test - NumPy array" and displays a 13x2 grid of numerical values. The right window is titled "y_test - NumPy array" and displays a 13x1 grid of categorical values represented by red and blue squares. Both windows have "Format", "Resize", and "Background color" buttons at the bottom.

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

For training set:

The image shows two side-by-side data visualization windows. The left window is titled "x_test - NumPy array" and displays a 13x2 grid of numerical values. The right window is titled "y_test - NumPy array" and displays a 13x1 grid of categorical values represented by red and blue squares. Both windows have "Format", "Resize", and "Background color" buttons at the bottom.

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

1. #feature Scaling
2. from sklearn.preprocessing import StandardScaler
3. st_x= StandardScaler()
4. x_train= st_x.fit_transform(x_train)
5. x_test= st_x.transform(x_test)

The scaled output is given below:

The image shows two side-by-side data visualization windows. Both windows have a title bar labeled 'x_test - NumPy array' and 'x_train - NumPy array' respectively. Each window contains a table with 13 rows and 2 columns. The first column is labeled '0' and the second column is labeled '1'. The data values are color-coded: pink for negative values and blue for positive values. The 'x_test' window has a vertical scroll bar on the right. At the bottom of each window are buttons for 'Format', 'Resize', and 'Background color' (with a checked checkbox), and a 'Save and Close' button.

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0	1
0	0.581649	-0.886707
1	-0.606738	1.46174
2	-0.0125441	-0.567782
3	-0.606738	1.89663
4	1.37391	-1.40858
5	1.47294	0.997847
6	0.0864882	-0.799728
7	-0.0125441	-0.248858
8	-0.210609	-0.567782
9	-0.210609	-0.190872
10	-0.309641	-1.29261
11	-0.309641	-0.567782
12	0.383585	0.0990599

2. Fitting Logistic Regression to the Training set:

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

1. #Fitting Logistic Regression to the training set
2. from sklearn.linear_model **import** LogisticRegression
3. classifier= LogisticRegression(random_state=0)
4. classifier.fit(x_train, y_train)

Output: By executing the above code, we will get the below output:

Out[5]:

1. LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
2. intercept_scaling=1, l1_ratio=None, max_iter=100,
3. multi_class='warn', n_jobs=None, penalty='l2',
4. random_state=0, solver='warn', tol=0.0001, verbose=0,
5. warm_start=False)

Hence our model is well fitted to the training set.

3. Predicting the Test Result

Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)

In the above code, we have created a y_pred vector to predict the test set result.

Output: By executing the above code, a new vector (y_pred) will be created under the variable explorer option. It can be seen as:



The above output image shows the corresponding predicted users who want to purchase or not purchase the car.

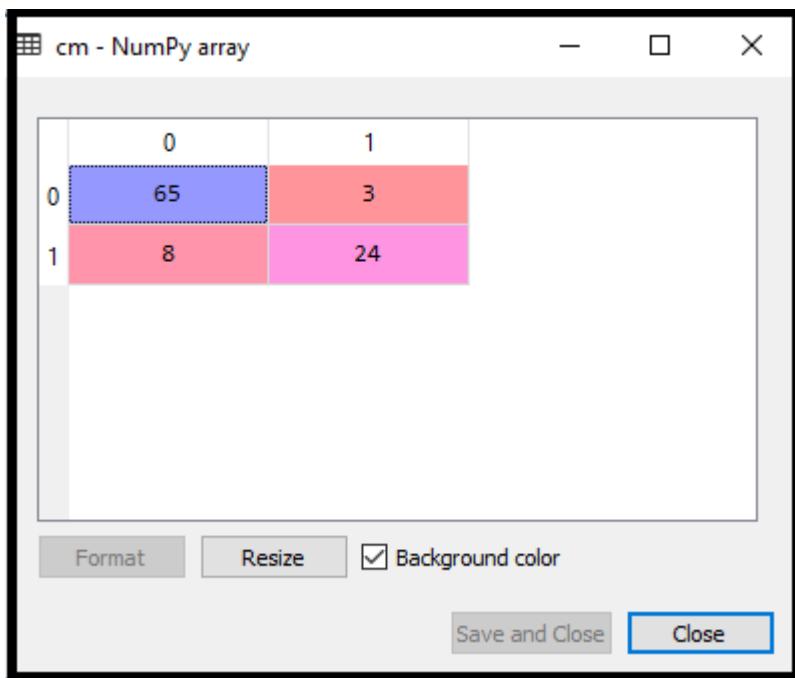
4. Test Accuracy of the result

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the `confusion_matrix` function of the `sklearn` library. After importing the function, we will call it using a new variable `cm`. The function takes two parameters, mainly `y_true`(the actual values) and `y_pred` (the targeted value return by the classifier). Below is the code for it:

1. #Creating the Confusion matrix
2. from `sklearn.metrics` **import** `confusion_matrix`
3. `cm= confusion_matrix()`

Output:

By executing the above code, a new confusion matrix will be created. Consider the below image:



We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that $65+24= 89$ (Correct Output) and $8+3= 11$ (Incorrect Output).

5. Visualizing the training set result

Finally, we will visualize the training set result. To visualize the result, we will use **ListedColormap** class of matplotlib library. Below is the code for it:

1. #Visualizing the training set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_train, y_train
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01), nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
- 6.

7. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape), alpha = 0.75, cmap = ListedColormap(['purple','green']))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c = ListedColormap(['purple', 'green'])(i), label = j)

```

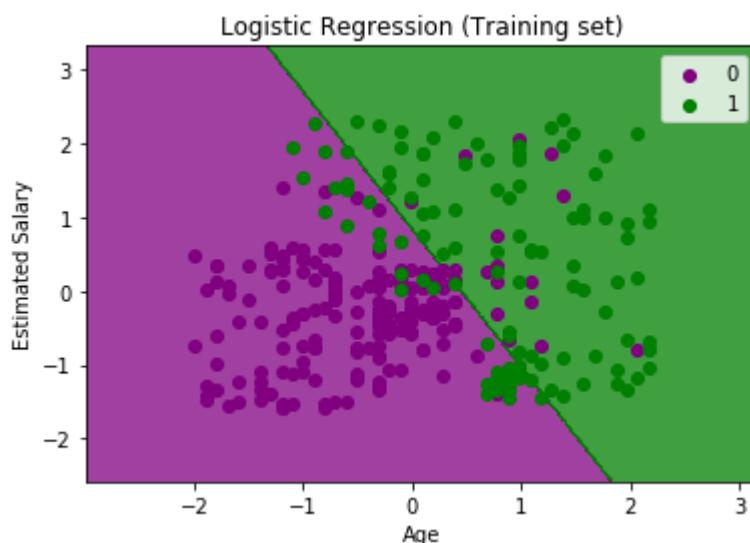
13. mtp.title("Logistic Regression (Training set)")
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

In the above code, we have imported the **ListedColormap** class of Matplotlib library to create the colormap for visualizing the result. We have created two new variables **x_set** and **y_set** to replace **x_train** and **y_train**. After that, we have used the **nm.meshgrid** command to create a rectangular grid, which has a range of -1(minimum) to 1 (maximum). The pixel points we have taken are of 0.01 resolution.

To create a filled contour, we have used **mtp.contourf** command, it will create regions of provided colors (purple and green). In this function, we have passed the **classifier.predict** to show the predicted data points predicted by the classifier.

Output: By executing the above code, we will get the below output:



The graph can be explained in the below points:

- In the above graph, we can see that there are some **Green points** within the green region and **Purple points** within the purple region.
- All these data points are the observation points from the training set, which shows the result for purchased variables.
- This graph is made by using two independent variables i.e., **Age on the x-axis** and **Estimated salary on the y-axis**.
- The **purple point observations** are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car.

- The **green point observations** are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.
- We can also estimate from the graph that the users who are younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.
- But there are some purple points in the green region (Buying the car) and some green points in the purple region(Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.

The goal of the classifier:

We have successfully visualized the training set result for the logistic regression, and our goal for this classification is to divide the users who purchased the SUV car and who did not purchase the car. So from the output graph, we can clearly see the two regions (Purple and Green) with the observation points. The Purple region is for those users who didn't buy the car, and Green Region is for those users who purchased the car.

Linear Classifier:

As we can see from the graph, the classifier is a Straight line or linear in nature as we have used the Linear model for Logistic Regression. In further topics, we will learn for non-linear Classifiers.

Visualizing the test set result:

Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set). The code for the test set will remain same as above except that here we will use **x_test** and **y_test** instead of **x_train** and **y_train**. Below is the code for it:

1. #Visulaizing the test set result
2. from matplotlib.colors **import** ListedColormap
3. **x_set, y_set = x_test, y_test**
4. **x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),**
5. **nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))**
- 6.

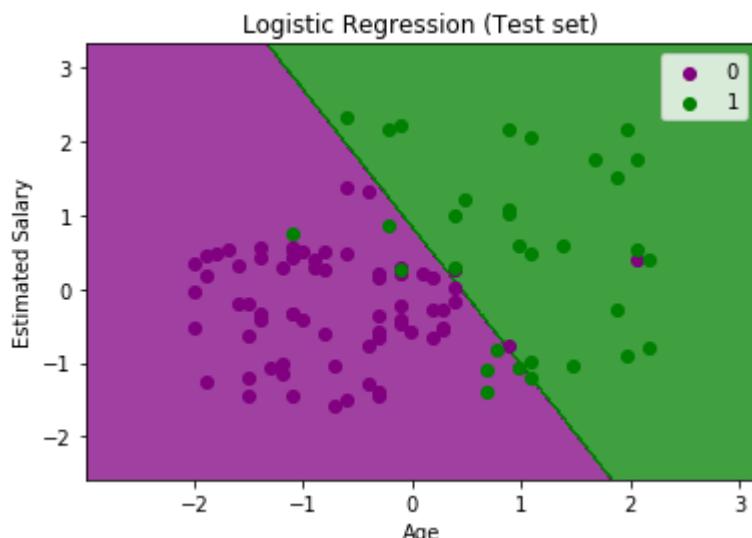
```
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
```

```

7. alpha = 0.75, cmap = ListedColormap(['purple','green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                  c = ListedColormap(['purple', 'green'])(i), label = j)
13. mtp.title('Logistic Regression (Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

Output:



The above graph shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).

Hence our model is pretty good and ready to make new predictions for this classification problem.

What is Naive Bayes algorithm?

It is a classification technique based on [Bayes' Theorem](#) with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability $P(c|x)$ from $P(c)$, $P(x)$ and $P(x|c)$. Look at the equation below:

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

Likelihood Class Prior Probability
↓ ↓
Posterior Probability Predictor Prior Probability

$$P(c|\mathbf{x}) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Above,

- $P(c|x)$ is the posterior probability of *class* (*c, target*) given *predictor* (*x, attributes*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

How Naive Bayes algorithm works?

Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table			
Weather	No	Yes	
Overcast		4	=4/14 0.29
Rainy	3	2	=5/14 0.36
Sunny	2	3	=5/14 0.36
All	5	9	
	=5/14	=9/14	
	0.36	0.64	

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem: Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

$$P(\text{Yes} | \text{Sunny}) = P(\text{ Sunny} | \text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

Here we have $P(\text{Sunny} | \text{Yes}) = 3/9 = 0.33$, $P(\text{Sunny}) = 5/14 = 0.36$, $P(\text{Yes}) = 9/14 = 0.64$

Now, $P(\text{Yes} | \text{Sunny}) = 0.33 * 0.64 / 0.36 = 0.60$, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

What are the Pros and Cons of Naive Bayes?

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from `predict_proba` are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

4 Applications of Naive Bayes Algorithms

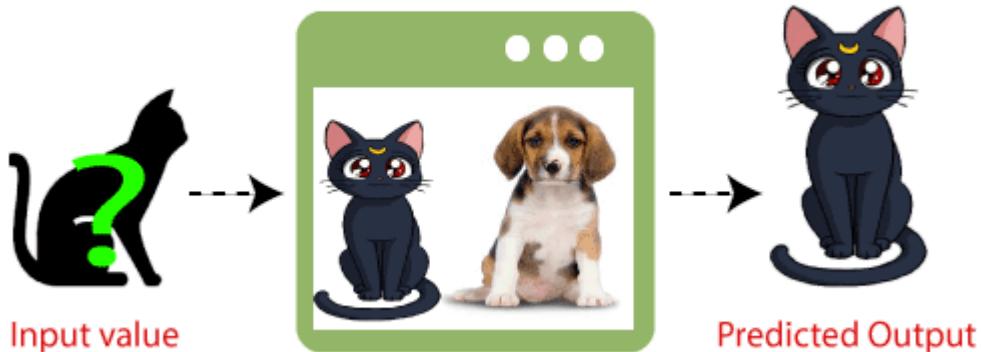
- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment

Analysis (in social media analysis, to identify positive and negative customer sentiments)

- **Recommendation System:** Naive Bayes Classifier and [Collaborative Filtering](#) together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

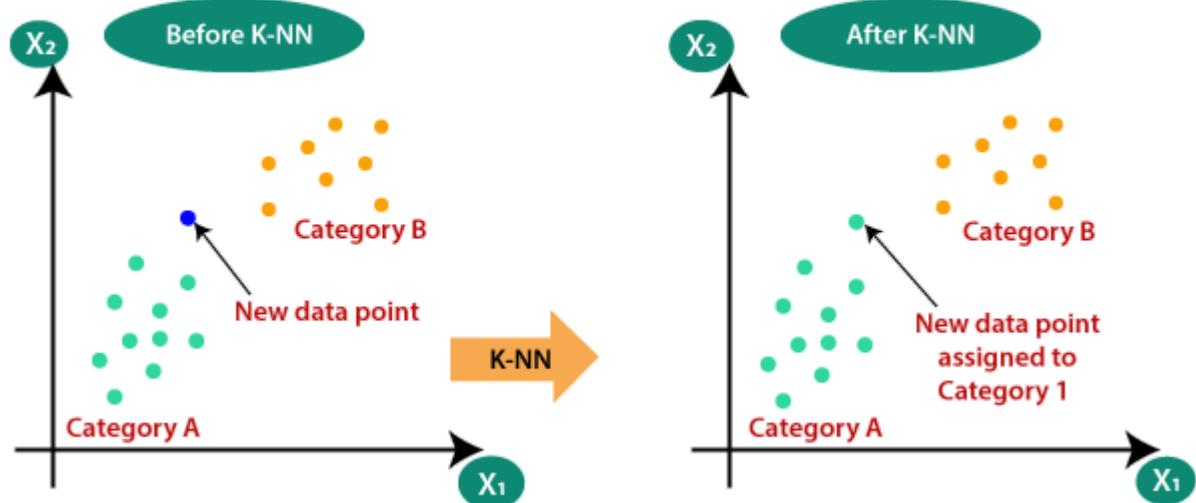
- category.

KNN Classifier



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x_1 , so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



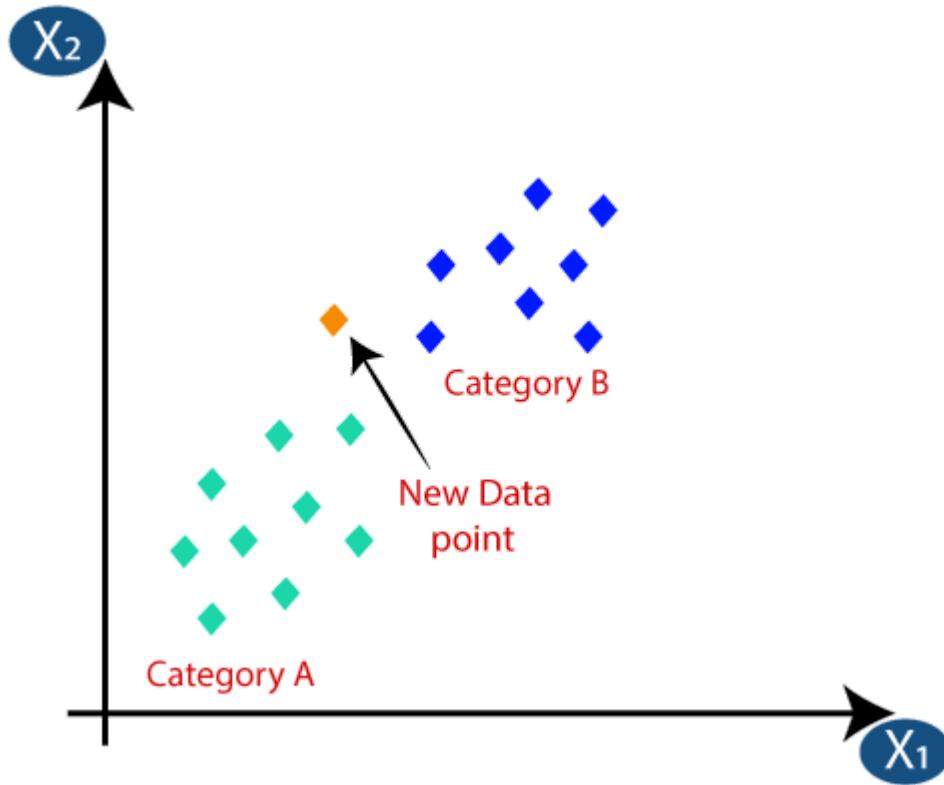
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

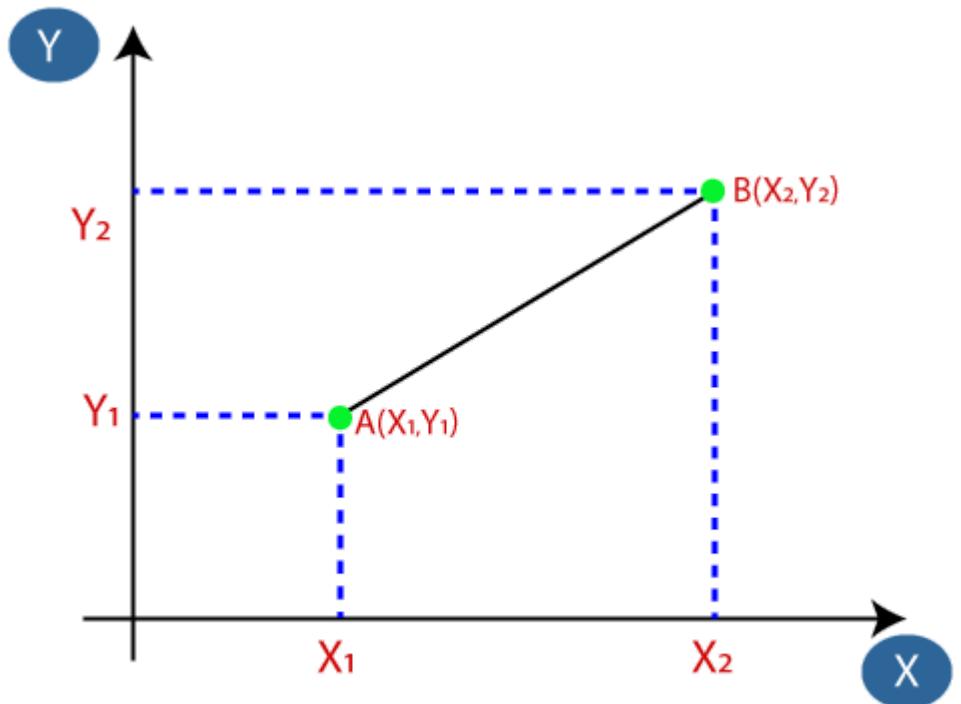
- Step-1: Select the number K of the neighbors

- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the $k=5$.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



$$\text{Euclidean Distance between } A_1 \text{ and } B_2 = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

Stay

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Python implementation of the KNN algorithm

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model. Below is the problem description:

Problem for K-NN Algorithm: There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains

lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps to implement the K-NN algorithm:

- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

1. # importing libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp

```
4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')
8.
9. #Extracting Independent and dependent Variable
10.x= data_set.iloc[:, [2,3]].values
11.y= data_set.iloc[:, 4].values
12.
13.# Splitting the dataset into training and test set.
14.from sklearn.model_selection import train_test_split
15.

x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0
)
16.
17.#feature Scaling
18.from sklearn.preprocessing import StandardScaler
19.st_x= StandardScaler()
20.x_train= st_x.fit_transform(x_train)
21.x_test= st_x.transform(x_test)
```

By executing the above code, our dataset is imported to our program and well pre-processed. After feature scaling our test dataset will look like:

x_test - NumPy array

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

y_test - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

From the above output image, we can see that our data is successfully scaled.

- **Fitting K-NN classifier to the Training data:**

Now we will fit the K-NN classifier to the training data. To do this we will import the **KNeighborsClassifier** class of **Sklearn Neighbors** library. After importing the class, we will create the **Classifier** object of the class. The Parameter of this class will be

- **n_neighbors:** To define the required neighbors of the algorithm. Usually, it takes 5.
- **metric='minkowski':** This is the default parameter and it decides the distance between the points.
- **p=2:** It is equivalent to the standard Euclidean metric.

And then we will fit the classifier to the training data. Below is the code for it:

1. #Fitting K-NN classifier to the training set
2. from sklearn.neighbors import KNeighborsClassifier
3. classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
4. classifier.fit(x_train, y_train)

Output: By executing the above code, we will get the output as:

```
Out[10]:  
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,  
                     weights='uniform')
```

- o **Predicting the Test Result:** To predict the test set result, we will create a **y_pred** vector as we did in Logistic Regression. Below is the code for it:

1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)

Output:

The output for the above code will be:

y_pred - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0
12	0

Format Resize Background color

Save and Close Close

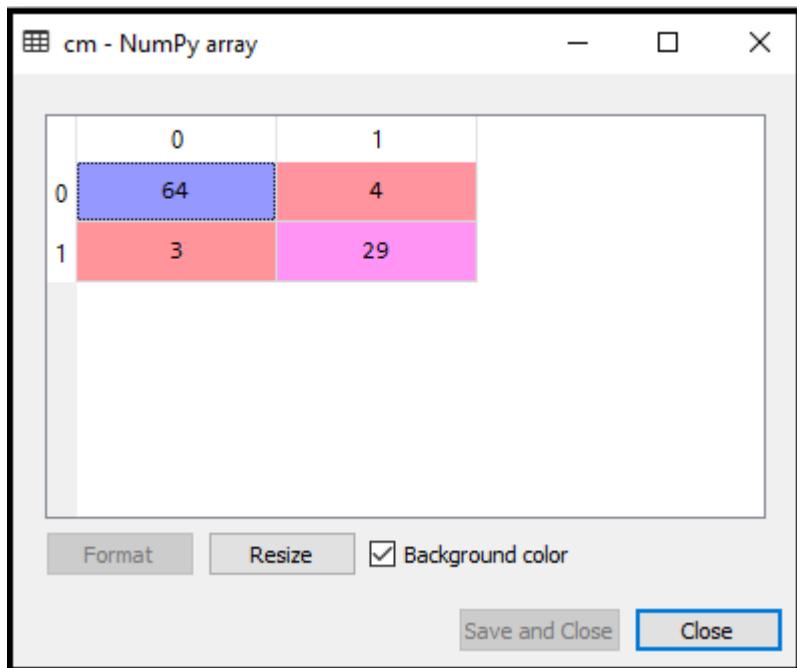
- **Creating the Confusion Matrix:**

Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics import confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)

In above code, we have imported the confusion_matrix function and called it using the variable cm.

Output: By executing the above code, we will get the matrix as below:



In the above image, we can see there are $64+29= 93$ correct predictions and $3+4= 7$ incorrect predictions, whereas, in Logistic Regression, there were 11 incorrect predictions. So we can say that the performance of the model is improved by using the K-NN algorithm.

- **Visualizing the Training set result:**

Now, we will visualize the training set result for K-NN model. The code will remain same as we did in Logistic Regression, except the name of the graph.

Below is the code for it:

1. #Visulaizing the trianing set result
 2. from matplotlib.colors **import** ListedColormap
 3. x_set, y_set = x_train, y_train
 4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01), nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
 5. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape), alpha = 0.75, cmap = ListedColormap(['red','green']))
 - 6.
- ```
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
```

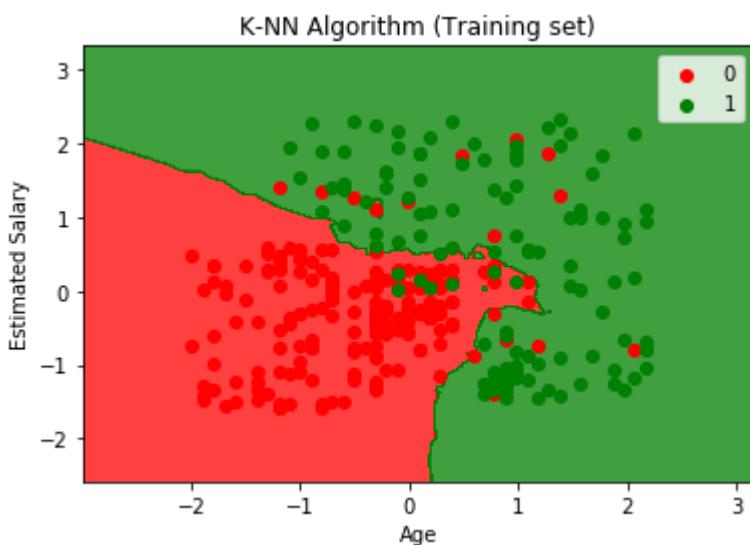
```

11. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12. c = ListedColormap(('red', 'green'))(i), label = j)
13.mtp.title('K-NN Algorithm (Training set)')
14.mtp.xlabel('Age')
15.mtp.ylabel('Estimated Salary')
16.mtp.legend()
17.mtp.show()

```

**Output:**

By executing the above code, we will get the below graph:



The output graph is different from the graph which we have occurred in Logistic Regression. It can be understood in the below points:

- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.
- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.

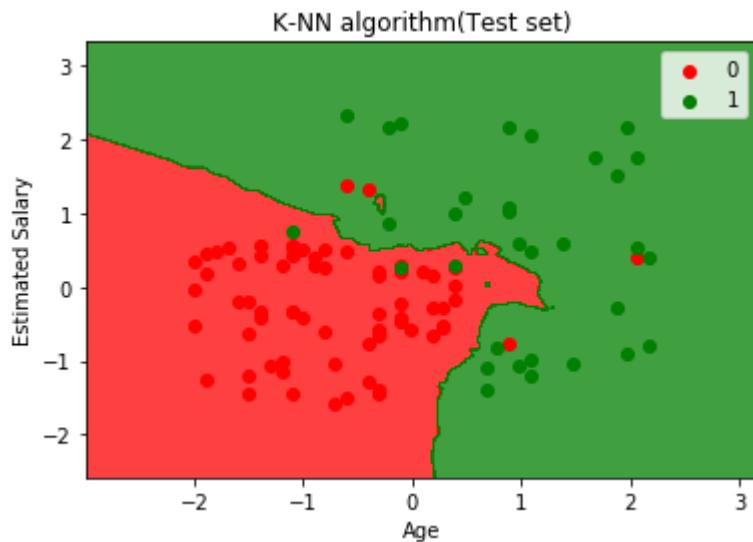
- The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.
- Hence our model is well trained.
- **Visualizing the Test set result:**

After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x\_train and y\_train** will be replaced by **x\_test and y\_test**.

Below is the code for it:

1. #Visualizing the test set result
2. from matplotlib.colors **import** ListedColormap
3. x\_set, y\_set = x\_test, y\_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01), nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))
5. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape), alpha = 0.75, cmap = ListedColormap(['red', 'green']))
6. mtp.xlim(x1.min(), x1.max())
7. mtp.ylim(x2.min(), x2.max())
8. **for** i, j in enumerate(nm.unique(y\_set)):
9. mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1], c = ListedColormap(['red', 'green'])(i), label = j)
10. mtp.title('K-NN algorithm(Test set)')
11. mtp.xlabel('Age')
12. mtp.ylabel('Estimated Salary')
13. mtp.legend()
14. mtp.show()

**Output:**



The above graph is showing the output for the test data set. As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.

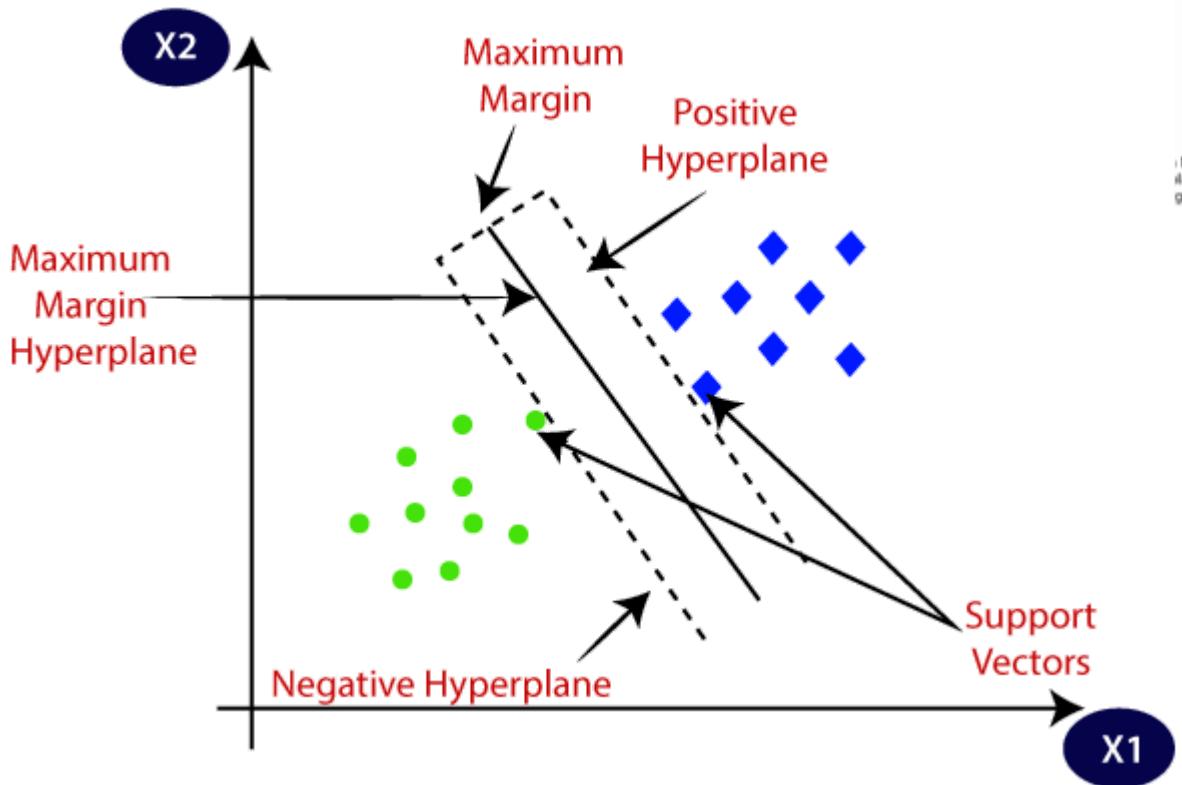
However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).

# Support Vector Machine Algorithm

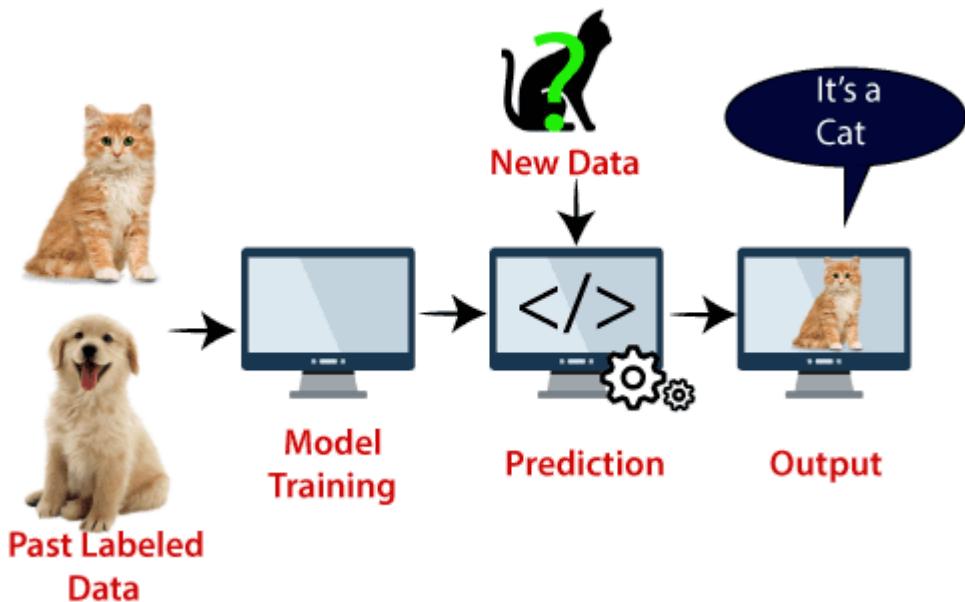
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



**Example:** SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization, etc.**

## Types of SVM

**SVM can be of two types:**

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

## Hyperplane and Support Vectors in the SVM algorithm:

**Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

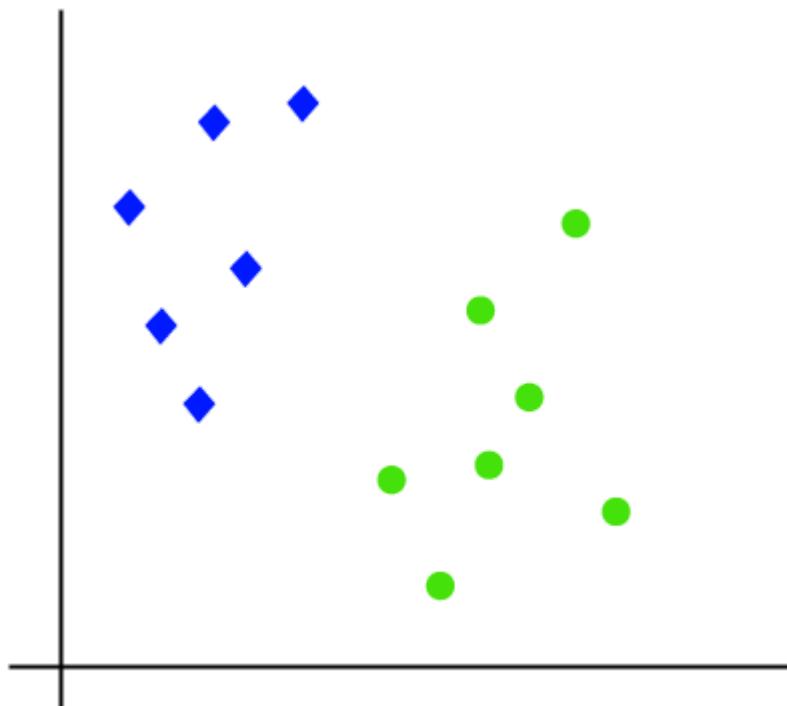
### **Support Vectors:**

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

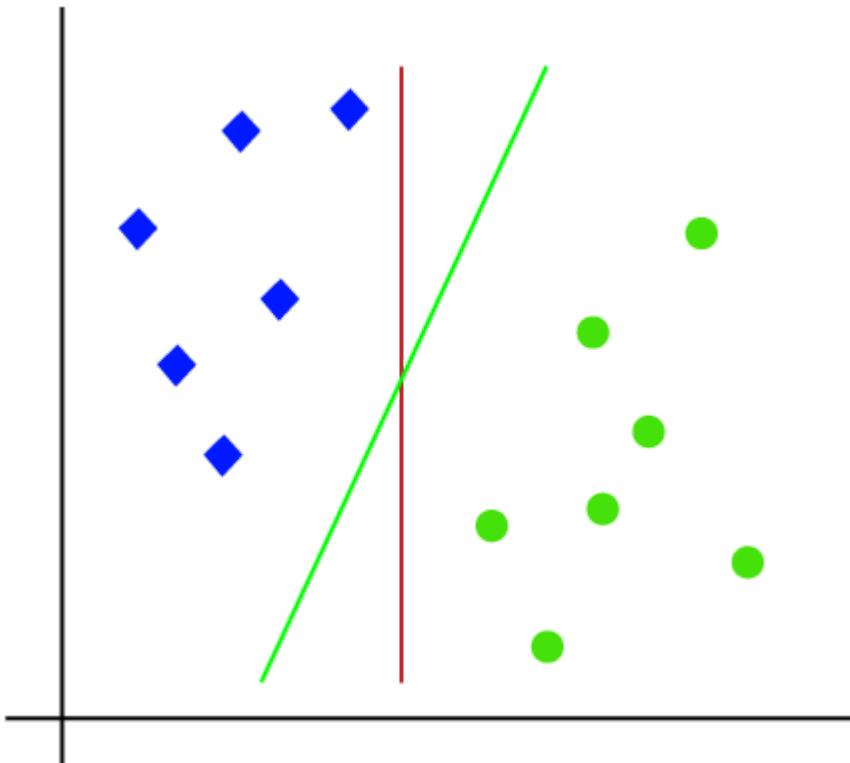
## **How does SVM works?**

### **Linear SVM:**

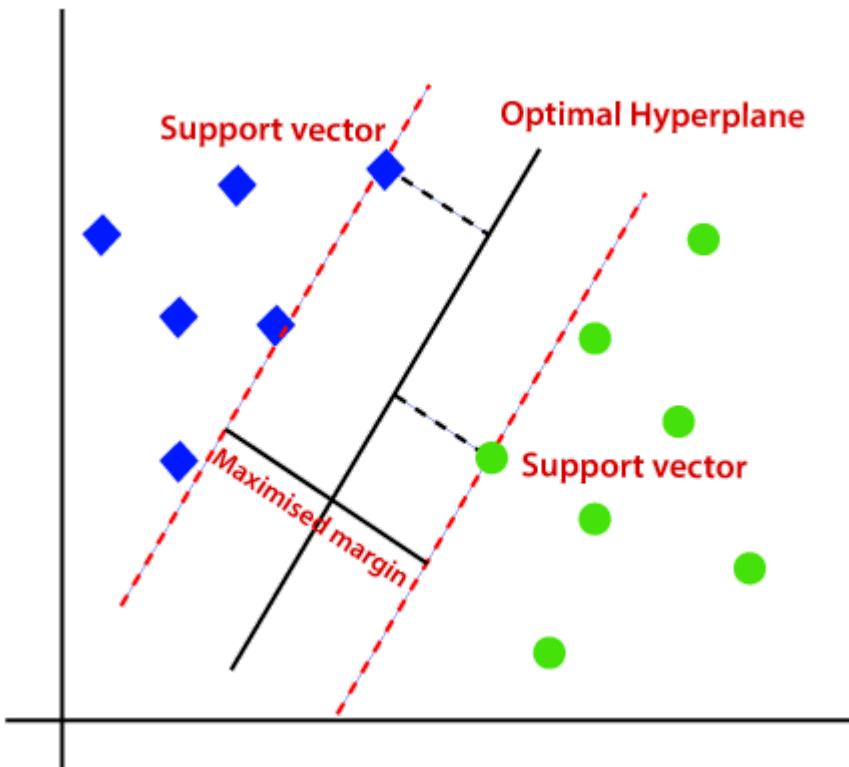
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features  $x_1$  and  $x_2$ . We want a classifier that can classify the pair( $x_1$ ,  $x_2$ ) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

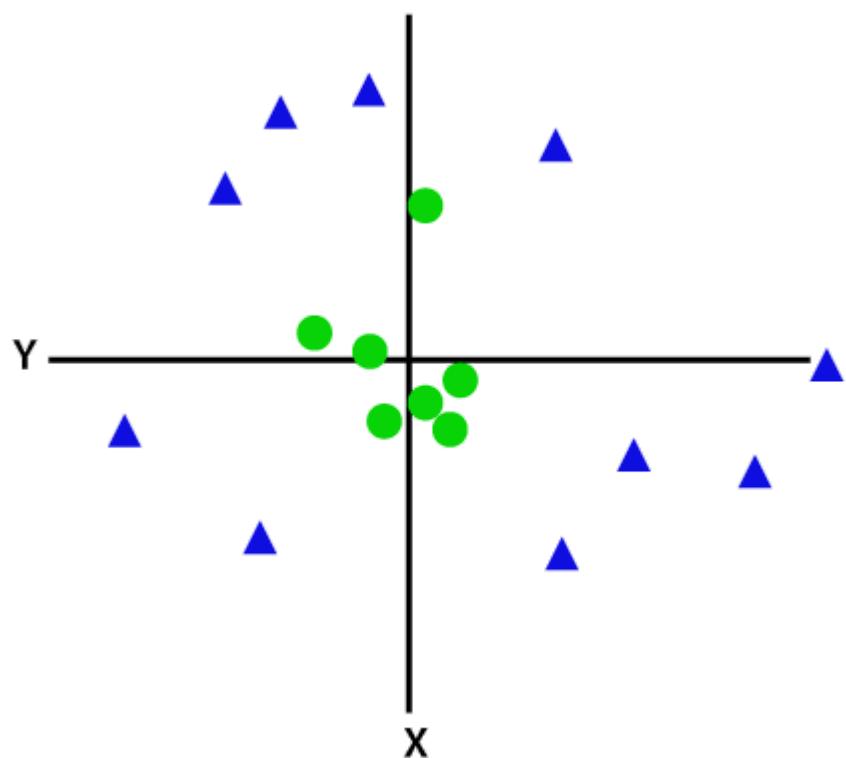


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



### Non-Linear SVM:

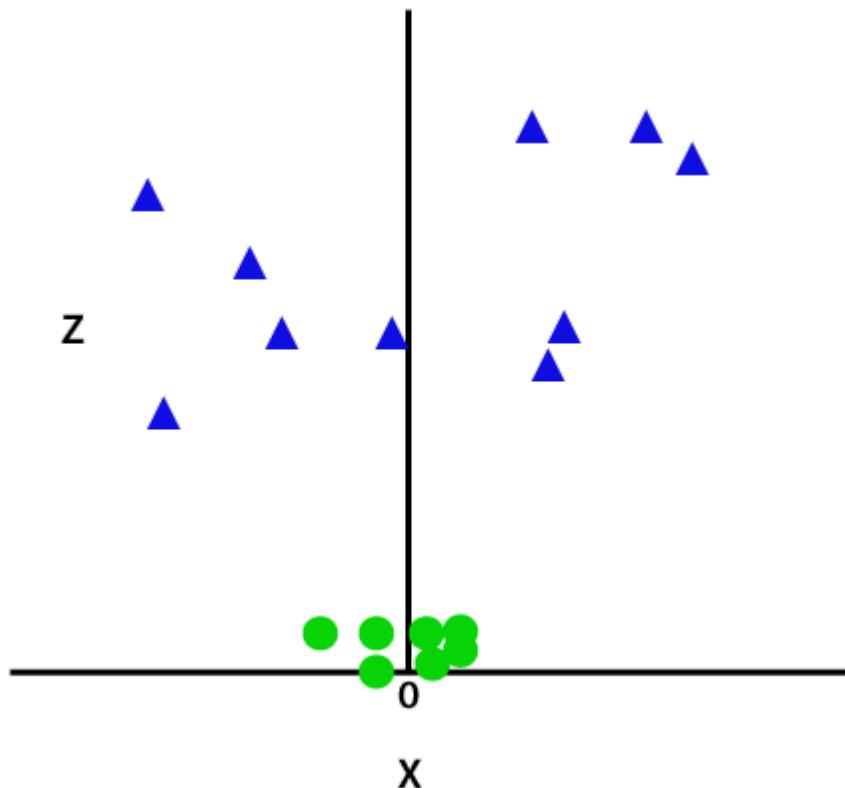
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



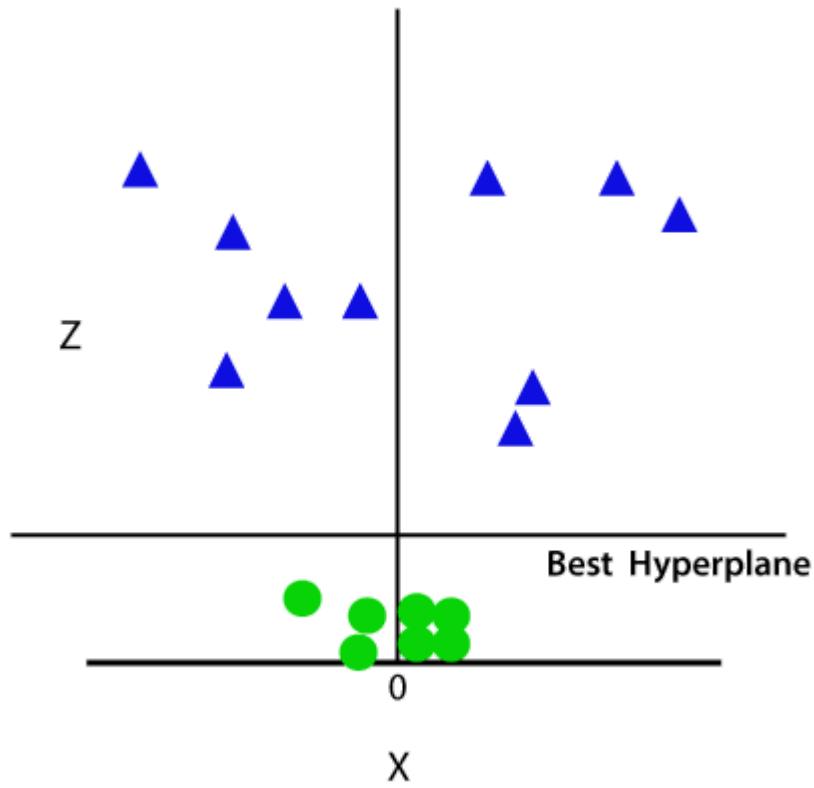
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z = x^2 + y^2$$

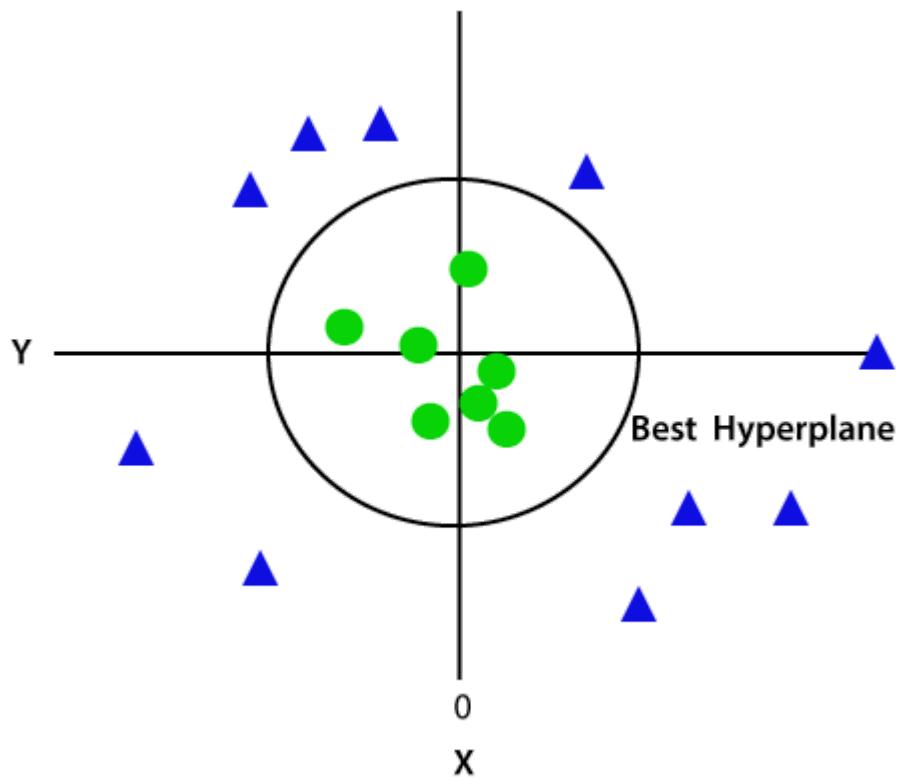
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with  $z=1$ , then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

## Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset **user\_data**, which we have used in Logistic regression and KNN classification.

- **Data Pre-processing step**

Till the Data pre-processing step, the code will remain the same. Below is the code:

```
1. #Data Pre-processing Step
2. # importing libraries
3. import numpy as nm
4. import matplotlib.pyplot as mtp
5. import pandas as pd
6.
7. #importing datasets
8. data_set= pd.read_csv('user_data.csv')
9.
10.#Extracting Independent and dependent Variable
11.x= data_set.iloc[:, [2,3]].values
12.y= data_set.iloc[:, 4].values
13.
14.# Splitting the dataset into training and test set.
15.from sklearn.model_selection import train_test_split
16.

 x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0
)
17.#feature Scaling
18.from sklearn.preprocessing import StandardScaler
19.st_x= StandardScaler()
20.x_train= st_x.fit_transform(x_train)
21.x_test= st_x.transform(x_test)
```

After executing the above code, we will pre-process the data. The code will give the dataset as:

data\_set - DataFrame

| Index | User ID  | Gender | Age | EstimatedSalary | Purchased |
|-------|----------|--------|-----|-----------------|-----------|
| 0     | 15624510 | Male   | 19  | 19000           | 0         |
| 1     | 15810944 | Male   | 35  | 20000           | 0         |
| 2     | 15668575 | Female | 26  | 43000           | 0         |
| 3     | 15603246 | Female | 27  | 57000           | 0         |
| 4     | 15804002 | Male   | 19  | 76000           | 0         |
| 5     | 15728773 | Male   | 27  | 58000           | 0         |
| 6     | 15598044 | Female | 27  | 84000           | 0         |
| 7     | 15694829 | Female | 32  | 150000          | 1         |
| 8     | 15600575 | Male   | 25  | 33000           | 0         |
| 9     | 15727311 | Female | 35  | 65000           | 0         |
| 10    | 15570769 | Female | 26  | 80000           | 0         |
| 11    | 15606274 | Female | 26  | 52000           | 0         |
| 12    | 15746139 | Male   | 20  | 86000           | 0         |
| 13    | 15704987 | Male   | 32  | 18000           | 0         |
| 14    | 15628972 | Male   | 18  | 82000           | 0         |

Format    Resize     Background color     Column min/max    Save and Close    Close

The scaled output for the test set will be:

The image shows two data frames side-by-side in a Jupyter Notebook interface. The left data frame is titled "x\_test - NumPy array" and contains 13 rows and 2 columns of numerical data. The right data frame is titled "y\_test - NumPy array" and contains 13 rows and 1 column of categorical data.

|    | 0          | 1          |
|----|------------|------------|
| 0  | -0.804802  | 0.504964   |
| 1  | -0.0125441 | -0.567782  |
| 2  | -0.309641  | 0.157046   |
| 3  | -0.804802  | 0.273019   |
| 4  | -0.309641  | -0.567782  |
| 5  | -1.1019    | -1.43758   |
| 6  | -0.70577   | -1.58254   |
| 7  | -0.210609  | 2.15757    |
| 8  | -1.99319   | -0.0459058 |
| 9  | 0.878746   | -0.770734  |
| 10 | -0.804802  | -0.596776  |
| 11 | -1.00287   | -0.422817  |
| 12 | -0.111576  | -0.422817  |

|    | 0 |
|----|---|
| 0  | 0 |
| 1  | 0 |
| 2  | 0 |
| 3  | 0 |
| 4  | 0 |
| 5  | 0 |
| 6  | 0 |
| 7  | 1 |
| 8  | 0 |
| 9  | 0 |
| 10 | 0 |
| 11 | 0 |
| 12 | 0 |

### Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import **SVC** class from **Sklearn.svm** library. Below is the code for it:

1. `from sklearn.svm import SVC # "Support vector classifier"`
2. `classifier = SVC(kernel='linear', random_state=0)`
3. `classifier.fit(x_train, y_train)`

In the above code, we have used **kernel='linear'**, as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(`x_train, y_train`)

### Output:

```
Out[8] :
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
 decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
 kernel='linear', max_iter=-1, probability=False, random_state=0,
 shrinking=True, tol=0.001, verbose=False)
```

The model performance can be altered by changing the value of **C(Regularization factor)**, **gamma**, and **kernel**.

- **Predicting the test set result:**

Now, we will predict the output for test set. For this, we will create a new vector **y\_pred**. Below is the code for it:

1. #Predicting the test set result
2. **y\_pred= classifier.predict(x\_test)**

After getting the **y\_pred** vector, we can compare the result of **y\_pred** and **y\_test** to check the difference between the actual value and predicted value.

**Output:** Below is the output for the prediction of the test set:

| y_pred - NumPy array |   |
|----------------------|---|
| 0                    | 0 |
| 1                    | 0 |
| 2                    | 0 |
| 3                    | 0 |
| 4                    | 0 |
| 5                    | 0 |
| 6                    | 0 |
| 7                    | 1 |
| 8                    | 0 |
| 9                    | 0 |
| 10                   | 0 |
| 11                   | 0 |
| 12                   | 0 |

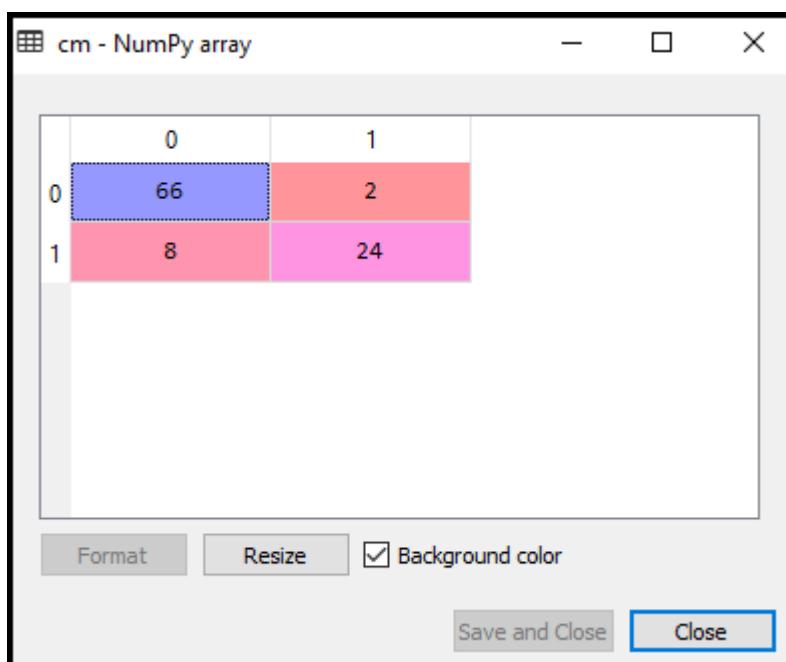
- **Creating the confusion matrix:**

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression

classifier. To create the confusion matrix, we need to import the **confusion\_matrix** function of the **sklearn** library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y\_true**( the actual values) and **y\_pred** (the targeted value return by the classifier). Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix(y\_test, y\_pred)

#### Output:



As we can see in the above output image, there are  $66+24= 90$  correct predictions and  $8+2= 10$  correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

- o **Visualizing the training set result:**

Now we will visualize the training set result, below is the code for it:

1. from matplotlib.colors **import** ListedColormap
2. x\_set, y\_set = x\_train, y\_train
3. x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),
4. nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

5.

```
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
6. alpha = 0.75, cmap = ListedColormap(['red', 'green')))
7. mtp.xlim(x1.min(), x1.max())
8. mtp.ylim(x2.min(), x2.max())
9. for i, j in enumerate(nm.unique(y_set)):
10. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
11. c = ListedColormap(['red', 'green'])(i), label = j)
12. mtp.title('SVM classifier (Training set)')
13. mtp.xlabel('Age')
14. mtp.ylabel('Estimated Salary')
15. mtp.legend()
16. mtp.show()
```

#### Output:

By executing the above code, we will get the output as:



As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have **used a linear kernel in the classifier**. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

- **Visualizing the test set result:**

1. #Visulaizing the test set result

```

2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_test, y_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6.

```

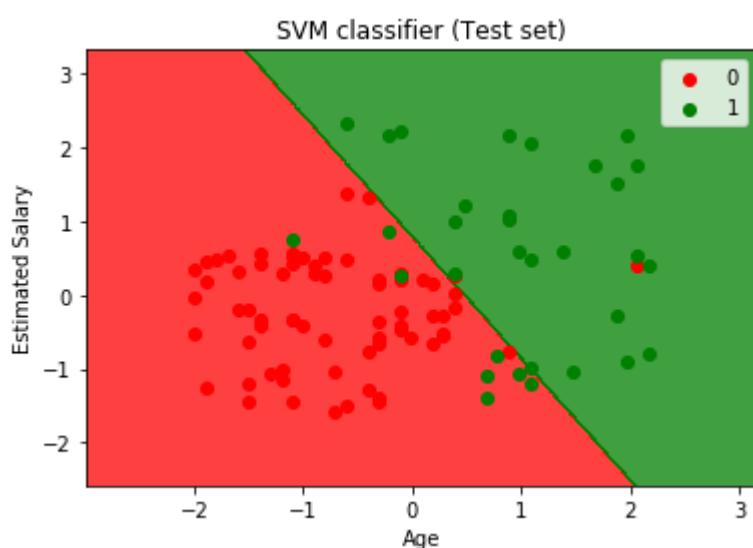
```

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.
shape),
7. alpha = 0.75, cmap = ListedColormap(['red', 'green']))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12. c = ListedColormap(['red', 'green'])(i), label = j)
13. mtp.title('SVM classifier (Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

### Output:

By executing the above code, we will get the output as:



As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased). Users who purchased the SUV are

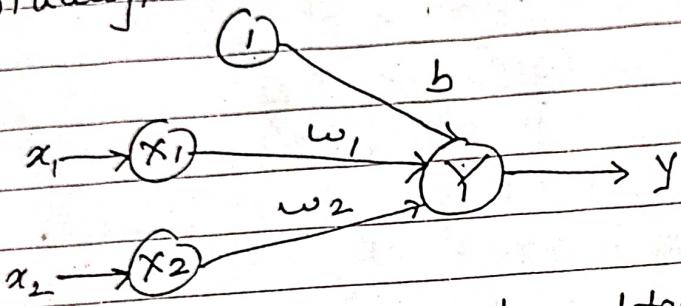
in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

\* Implement AND function using perceptron  
H/w for bipolar inputs & targets.

Truth table for bipolar inputs & targets:

| $x_1$ | $x_2$ | t  |
|-------|-------|----|
| 1     | 1     | 1  |
| 1     | -1    | -1 |
| -1    | 1     | -1 |
| -1    | -1    | -1 |

Initially, assume  $w_1 = w_2 = b = 0, d = 1$



For the first input pattern,  $x_1=1, x_2=-1, t=1$

$$\begin{aligned} \text{net input, } y_{in} &= b + x_1 w_1 + x_2 w_2 \\ &= 0 + 1(0) + (-1)(0) \\ &= 0 \end{aligned}$$

Since,  $y_{in} = 0 \Rightarrow y = 0$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

Check whether  $t=y$ ,  $t=1, y=0$

Hence,  $t \neq y$

Hence, weight updation:

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$

$$w_1(\text{new}) = 0 + 1(1)(1) = 1$$

$$w_2(\text{new}) = 0 + 1(-1)(-1) = 1$$

$$b(\text{new}) = 0 + 1 \times 1 = 1$$

Hence,  $w_1=1, w_2=1, b=1$

Next input pattern,  $x_1=1, x_2=-1, t=-1$

net input,  $y_{in} = b + x_1 w_1 + x_2 w_2$

$$= 1 + 1(1) + (-1)(-1)$$

$$= 1 + 1 - 1$$

$$= 1$$

Since  $y_{in} = 1, y=1$ , whether  $t=y$ ?

No, Hence, weight updation.

$$w_1(\text{new}) = w_1(\text{old}) + \alpha + x_1,$$

$$= 1 + 1(-1)(1)$$

$$= 1 - 1 = 0$$

$$w_2(\text{new}) = w_2(\text{old}) + \alpha + x_2,$$

$$= 1 + 1(-1)(-1)$$

$$= 1 + 1 = 2$$

$$b(\text{new}) = b(\text{old}) + \alpha t$$

$$= 1 + 1(-1)$$

$$= 1 - 1 = 0$$

Hence,  $w_1=0, w_2=2, b=0$

Next input pattern,  $x_1=-1, x_2=1, t=-1$

net input,  $y_{in} = b + x_1 w_1 + x_2 w_2$

$$= 0 + (-1)(0) + (1)(2)$$

$$= -1 + 0 + 2 = 1$$

Since,  $y_{in} = 1, y=1$ , whether  $t=y$ ?

No, Hence, weight updation.

$$w_1(\text{new}) = w_1(\text{old}) + \alpha + x_1,$$

$$= 0 + 1(-1)(-1)$$

$$= 1$$

$$w_2(\text{new}) = w_2(\text{old}) + \alpha + x_2,$$

$$= 2 + 1(-1)(1)$$

$$= 2 - 1 = 1$$

$$b(\text{new}) = b(\text{old}) + \alpha t$$

$$= 0 + 1(-1)$$

$$= -1$$

Hence, no weight updation.

Next input pattern is,  $x_1=0, x_2=0, t=-1$

$$\begin{aligned}y_{in} &= b + w_1 x_1 + w_2 x_2 \\&= 1 + 1(0) + 1(0) \\&= 1\end{aligned}$$

Since,  $y_{in}=1, y=1 \quad t \neq y$

Hence, weight updations.

$$\begin{aligned}w_1 &= w_1(\text{old}) + \alpha t x_1 \\&= 1 + 1(-1) 0 \\&= 1\end{aligned}$$

$$\begin{aligned}w_2 &= w_2(\text{old}) + \alpha t x_2 \\&= 1 + 1(-1) 0 \\&= 1\end{aligned}$$

$$\begin{aligned}b &= b(\text{old}) + \alpha t \\&= 1 + 1(-1) \\&= 0\end{aligned}$$

Hence,  $w_1 = -1, w_2 = 1, b = 0$

Epoch-2 :  $x_1=1, x_2=1, t=1$

$$\begin{aligned}y_{in} &= b + w_1 x_1 + w_2 x_2 \\&= 0 + 1(1) + 1(1) \\&= 2\end{aligned}$$

Since,  $y_{in}=2, y=1 \quad t \neq y$

Hence, no weight updation.

$x_1=1, x_2=0, t=1$

$$\begin{aligned}y_{in} &= b + w_1 x_1 + w_2 x_2 \\&= 0 + 1(1) + 1(0) \\&= 0 + 1 + 0 = 1\end{aligned}$$

Since,  $y_{in}=1, y=1 \quad t=y$

Hence, no updation.

(to be continued)

\*\* Find the weights required to perform the following classification using perceptron. The vectors  $(1, 1, 1, 1)$  and  $(-1, 1, -1, -1)$  are belonging to class  $C$  (target value 1) & vectors  $(1, 1, 1, -1)$  &  $(1, -1, -1, 1)$  don't belong to class  $C$  (target value 0). Assume learning rate  $\alpha$  & initial weight as 0. Also assume activation function as

$$y = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \leq y_{in} \leq 0.2 \\ -1 & \text{if } y_{in} < -0.2 \end{cases}$$

Assume,  $w_1 = w_2 = w_3 = w_4 = b = 0$

$$\begin{aligned} y_{in} &= b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 \\ &= 0 + 1(0) + 1(0) + 1(0) + 1(0) \\ &= 0 \end{aligned}$$

Since,  $y_{in} = 0$ ,  $y = 0 \therefore \neq t$

Hence, weight updation.

$$\begin{aligned} w_1(\text{new}) &= w_1(\text{old}) + \alpha t x_1 \\ &= 0 + 1(1)(1) \\ &= 1 \end{aligned}$$

$$\begin{aligned} w_2(\text{new}) &= w_2(\text{old}) + \alpha t x_2 \\ &= 0 + 1(-1)(-1) \\ &= 1 \end{aligned}$$

$$\begin{aligned} w_3(\text{new}) &= w_3(\text{old}) + \alpha t x_3 \\ &= 0 + 1(1)(1) \\ &= 1 \end{aligned}$$

$$\begin{aligned} w_4(\text{new}) &= w_4(\text{old}) + \alpha t x_4 \\ &= 0 + 1(1)(-1) \\ &= 1 \end{aligned}$$

Next input pattern:  $x_1 = -1, x_2 = 1, x_3 = -1, x_4 = -1$

$$\begin{aligned}y_{in} &= b + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4, \\&= 1 + 1(-1) + 1(1) + 1(-1) + 1(-1) \\&= 1 - 1 + 1 - 1 - 1 = -1\end{aligned}$$

Since  $y_{in} = -1, y = -1$   
 $\therefore t \neq y$

Hence, weight updatations.

$$\begin{aligned}w_1(\text{new}) &= w_1(\text{old}) + \alpha t x_1, \\&= 1 + 1(1)(-1) \\&= 1 - 1 = 0\end{aligned}$$

$$\begin{aligned}w_2(\text{new}) &= w_2(\text{old}) + \alpha t x_2, \\&= 1 + 1(1)(1) \\&= 1 + 1 = 2\end{aligned}$$

$$\begin{aligned}w_3(\text{new}) &= w_3(\text{old}) + \alpha t x_3, \\&= 1 + 1(1)(-1) \\&= 1 - 1 = 0\end{aligned}$$

$$\begin{aligned}w_4(\text{new}) &= w_4(\text{old}) + \alpha t x_4, \\&= 1 + 1(1)(-1) \\&= 1 - 1 = 0\end{aligned}$$

$$\begin{aligned}b(\text{new}) &= b(\text{old}) + \alpha t \\&= 1 + 1(1) = 2\end{aligned}$$

$$w_1 = 0, w_2 = 2, w_3 = 0, w_4 = 0, b = 2$$

Next input pattern:  $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = -1$

$$\begin{aligned}y_{in} &= b + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4, \\&= 2 + 0(1) + 2(1) + 0(1) + 0(-1) \\&= 2 + 2 = 4\end{aligned}$$

Since  $y_{in} = 4, y = 1 \therefore t \neq y$

Hence, weight updatation.

$$\begin{aligned}w_1(\text{new}) &= w_1(\text{old}) + \alpha t x_1, \\&= 0 + 1(-1)(1) \\&= -1\end{aligned}$$

$$\begin{aligned}w_2(\text{new}) &= w_2(\text{old}) + \alpha t x_2 \\&= 1 + 1(-1)(1) \\&= 1 - 1 = 0\end{aligned}$$

$$\begin{aligned}w_3(\text{new}) &= w_3(\text{old}) + \alpha t x_3 \\&= 0 + 1(-1) \\&= 0 - 1 = -1\end{aligned}$$

$$\begin{aligned}w_4(\text{new}) &= w_4(\text{old}) + \alpha t x_4 \\&= 0 + 1(-1)(-1) \\&= 1\end{aligned}$$

$$\begin{aligned}b(\text{new}) &= b(\text{old}) + \alpha t \\&= 2 + 1(-1) \\&= 1\end{aligned}$$

Hence,  $w_1 = -1, w_2 = 1, w_3 = -1, w_4 = 1, b = 1$

Next input pattern:  $x_1 = 1, x_2 = -1, x_3 = -1, x_4 = 1$   
 $t = -1$

$$\begin{aligned}y_{\text{in}} &= b + \alpha w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 \\&= 1 + (-1)(1) + 1(-1) + (-1)(-1) + 1(1) \\&= 1 - 1 - 1 + 1 + 1 \\&= 1\end{aligned}$$

Since,  $y_{\text{in}} = 1, y = 1 \therefore t \neq y$

Hence, weight updation.

$$\begin{aligned}w_1(\text{new}) &= w_1(\text{old}) + \alpha t x_1 \\&= (-1) + 1(-1)(1) \\&= -2\end{aligned}$$

$$\begin{aligned}w_2(\text{new}) &= w_2(\text{old}) + \alpha t x_2 \\&= 1 + 1(-1)(-1) \\&= 2\end{aligned}$$

$$\begin{aligned}w_3(\text{new}) &= w_3(\text{old}) + \alpha t x_3 \\&= -1 + 1(-1)(-1) \\&= 0\end{aligned}$$

Hence,  $w_1 = -2, w_2 = 2, w_3 = 0, w_4 = 0$

Epoch-2

:

:

:

Epoch-3:

$$w_1 = -2, w_2 = 2, w_3 = 0, w_4 = 2, b = 0.$$

First input pattern:  $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$   
 $t = 1$ .

$$\begin{aligned}y_{in} &= b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 \\&= 0 + 1(-2) + 1(2) + 1(0) + 1(2) \\&= 0 - 2 + 2 + 2 \\&= 2\end{aligned}$$

Since,  $y_{in} = 2, y = 1 \therefore t = y$

No weight updation.

Second i/p pattern:  $x_1 = -1, x_2 = 1, x_3 = -1, x_4 = 1$   
 $t = 1$

$$\begin{aligned}y_{in} &= b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 \\&= 0 + (-1)(-2) + 1(2) + (-1)(0) + (-1)(2) \\&= 2 + 2 + 0 - 2 \\&= 2\end{aligned}$$

Since,  $y_{in} = 2, y = 1, \therefore t = y$

- \* Classify two-dimensional pattern shown in figure below using perceptron w/w.



target value: 1      target value: -1  
 Assume  $\theta = 0$ ,  $\alpha = 1$ , activation function

$$y = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } -0 \leq y_{in} \leq 0 \\ -1 & \text{if } y_{in} < -0 \end{cases}$$

(\* represents 1 & • represents 0)

→ The training pattern is

|   | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | $t$ |
|---|-------|-------|-------|-------|-------|-------|-------|-------|-------|-----|
| C | 1     | 1     | 1     | 1     | 0     | 0     | 1     | 1     | 1     | 1   |
| A | 0     | 1     | 0     | 1     | 1     | 1     | 1     | 0     | 1     | 0   |

$$\begin{aligned}
 y_{in} &= b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + \\
 &\quad x_5 w_5 + x_6 w_6 + x_7 w_7 + x_8 w_8 + \\
 &\quad x_9 w_9 \\
 &= 0 + 1(0) + 1(0) + 1(0) + 1(0) + 0(0) + \\
 &\quad 0(0) + 1(0) + 1(0) + 1(0) \\
 &= 0
 \end{aligned}$$

Since,  $y_{in} = 0$ ,  $y = 0 \therefore t \neq y$

Hence, weight updation.

$$\begin{aligned}
 w_1(\text{new}) &= w_1(\text{old}) + \alpha x_1 \\
 &= 0 + 1(1)(1) = 1
 \end{aligned}$$

$$\begin{aligned}
 w_2(\text{new}) &= w_2(\text{old}) + \alpha x_2 \\
 &= 0 + 1(1)(1) = 1
 \end{aligned}$$

$$\begin{aligned}
 w_3(\text{new}) &= w_3(\text{old}) + \alpha x_3 \\
 &= 0 + 1(1)(1) = 1
 \end{aligned}$$

$$w_5(\text{new}) = w_5(\text{old}) + \alpha + x_5 \\ = 0 + 1(1)(0) = 0$$

$$w_6(\text{new}) = w_6(\text{old}) + \alpha + x_6 \\ = 0 + 1(1)(0) = 0$$

$$w_7(\text{new}) = w_7(\text{old}) + \alpha + x_7 \\ = 0 + 1(1)1 = 1$$

$$w_8(\text{new}) = w_8(\text{old}) + \alpha + x_8 \\ = 0 + 1(1)1 = 1$$

$$w_9(\text{new}) = w_9(\text{old}) + \alpha + x_9 \\ = 0 + 1(1)1 = 1$$

$$b(\text{new}) = b(\text{old}) + \alpha t \\ = 0 + 1(1) = 1$$

Second Pattern is:

$$y_{1n} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + x_5 w_5 \\ + x_6 w_6 + x_7 w_7 + x_8 w_8 + x_9 w_9 \\ = 1 + 0(1) + 1(1) + 0(1) + 1(1) + 1(0) + \\ 1(0) + 1(1) + 0(1) + 1(1) \\ = 1 + 1 + 1 + 1 + 1 = 5$$

Since,  $y_{1n} = 5$ ,  $y = 1 \neq y$

Hence, weight updations.

$$w_1(\text{new}) = 0 + 1(0)0 = 0$$

$$w_2(\text{new}) = w_2(\text{old}) + \alpha + x_2 = 1 + 1(0)1 = 1$$

$$w_3(\text{new}) = w_3(\text{old}) + \alpha + x_3 = 1 + 1(0)0 = 1$$

$$w_4(\text{new}) = w_4(\text{old}) + \alpha + x_4 = 1 + 1(0)1 = 1$$

$$w_5(\text{new}) = w_5(\text{old}) + \alpha + x_5 = 0 + 1(0)1 = 0$$

$$w_6(\text{new}) = w_6(\text{old}) + \alpha + x_6 = 0 + 1(0)1 = 0$$

$$w_7(\text{new}) = w_7(\text{old}) + \alpha + x_7 = 1 + 1(0)1 = 1$$

$$w_8(\text{new}) = w_8(\text{old}) + \alpha + x_8 = 1 + 1(0)0 = 1$$

$$w_9(\text{new}) = w_9(\text{old}) + \alpha + x_9 = 1 + 1(0)1 = 1$$

$$b(\text{new}) = b(\text{old}) + \alpha t = 1 + 1(0)1 = 1$$

Epoch - 2

$$\begin{aligned}
 y_{in} &= b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + x_5 w_5 \\
 &\quad + x_6 w_6 + x_7 w_7 + x_8 w_8 + x_9 w_9 \\
 &= 1 + 1(0) + 1(1) + 1(1) + 1(1) + 0(0) + \\
 &\quad 0(0) + 1(1) + 1(1) + 1(1) \\
 &= 1 + 1 + 1 + 1 + 1 + 1 = 7
 \end{aligned}$$

Since,  $y_{in} = 7$ ,  $y = 1$   $t = 1$

Hence, no weight updation.

$$y_{in} = b +$$

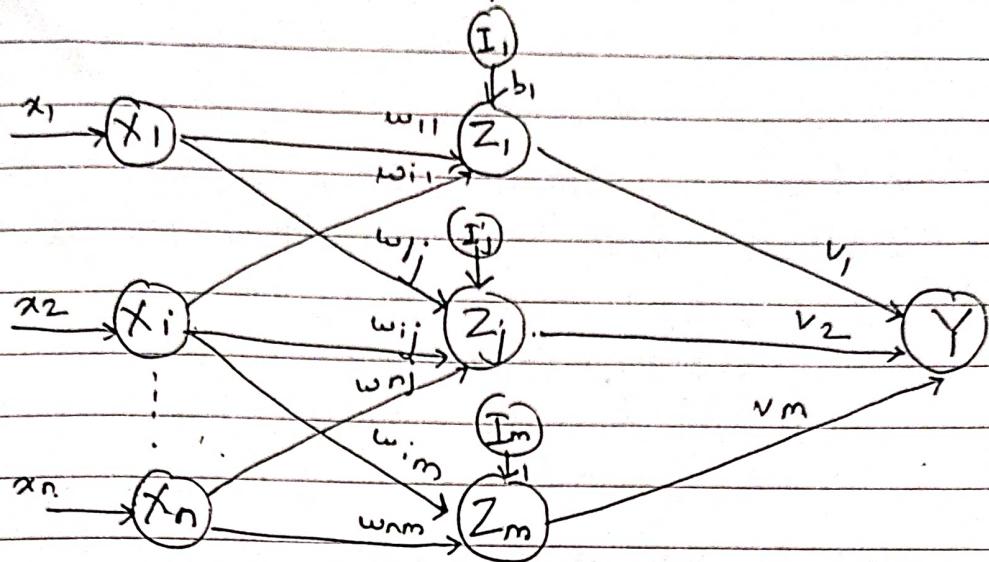
Second i/p pattern

$$\begin{aligned}
 y_{in} &= b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + x_5 w_5 \\
 &\quad + x_6 w_6 + x_7 w_7 + x_8 w_8 + x_9 w_9 \\
 &= 1 + 0(0) + 1(1) + 0(1) + 1(1) + 1(0) + \\
 &\quad 1(0) + 1(1) + 0(1) + 1(1) \\
 &= 1 + 1 + 1 + 1 + 1 = 5
 \end{aligned}$$

Adaline .

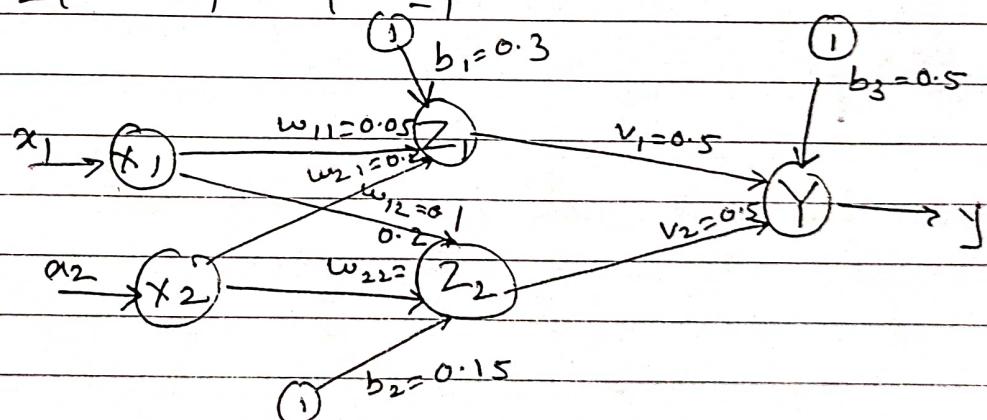
# Multiple Adaptive Linear Neurons

Date \_\_\_\_\_  
Page \_\_\_\_\_



$$x_1 \quad x_2 \quad 1 \quad t$$

~~$$\begin{matrix} & 1 & 1 & 1 & -1 \\ 1 & & & & \\ -1 & & & & \\ -1 & & & & \end{matrix}$$~~



$$[v_1 \ v_2 \ b_3] = [0.5 \ 0.5 \ 0.5]$$

The initial weights & bias are

$$[w_{11} \ w_{21} \ b_1] = [0.05 \ 0.2 \ 0.3]$$

$$[w_{12} \ w_{22} \ b_2] = [0.1 \ 0.2 \ 0.15]$$

For first i/p sample,  $x_1=1$ ,  $x_2=1$ ,  $t=-1$

- Calculate net i/p for hidden units

$$\begin{aligned}z_{in1} &= b_1 + x_1 w_{11} + x_2 w_{21} \\&= 0.3 + 1(0.05) + 1(0.2) \\&= 0.3 + 0.05 + 0.2 \\&= 0.55\end{aligned}$$

$$\begin{aligned}z_{in2} &= b_2 + x_1 w_{12} + x_2 w_{22} \\&= 0.15 + 1(0.1) + 1(0.2) \\&= 0.45\end{aligned}$$

- Calculate  $z_1, z_2$  by activation function:

$$f(z_{in}) = \begin{cases} 1 & \text{if } z_{in} > 0 \\ -1 & \text{if } z_{in} \leq 0 \end{cases}$$

$$\therefore z_1 = 1, z_2 = 1$$

- Calculate net i/p for o/p unit,

$$\begin{aligned}y_{in} &= b_3 + z_1 v_1 + z_2 v_2 \\&= 0.5 + 1(0.5) + 1(0.5) \\&= 1.5\end{aligned}$$

$$\therefore y = f(y_{in}) = f(1.5) = 1$$

Since,  $t \neq y$ , weight  $\omega$  updations

$$\begin{aligned}w_{ij}(\text{new}) &= w_{ij}(\text{old}) + \alpha(t - z_{inj}) \cdot x_i \\w_{11}(\text{new}) &= w_{11}(\text{old}) + \alpha(t - z_{in1}) x_1 \\&= 0.05 + 0.5(-1 - 0.55) \\&= 0.05 - 0.775 \\&= -0.725\end{aligned}$$

$$\begin{aligned}w_{12}(\text{new}) &= w_{12}(\text{old}) + \alpha(t - z_{in2}) x_2 \\&= 0.1 + 0.5(-1 - 0.45) \\&= 0.1 - 0.625\end{aligned}$$

$$\begin{aligned} b_1(\text{new}) &= b_1(\text{old}) + \alpha(t - z_{in_1}) \\ &= 0.3 + 0.5(-1 - 0.55) \\ &= -0.475 \end{aligned}$$

$$\begin{aligned} w_{21}(\text{new}) &= w_{21}(\text{old}) + \alpha(t - z_{in_1}) x_2 \\ &= 0.2 + 0.1(-1 - 0.55) \\ &= -0.575 \end{aligned}$$

$$\begin{aligned} w_{22}(\text{new}) &= w_{22}(\text{old}) + \alpha(t - z_{in_2}) x_2 \\ &= 0.2 + 0.5(-1 - 0.45) \\ &= -0.525 \end{aligned}$$

$$\begin{aligned} b_2(\text{new}) &= b_2(\text{old}) + \alpha(t - z_{in_2}) \\ &= 0.15 + 0.5(-1 - 0.45) \\ &= -0.575 \end{aligned}$$

$$\begin{bmatrix} x_1 & x_2 & t \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$$

net i/p for hidden units

$$\begin{aligned} z_{in_1} &= b_1 + x_1 w_{11} + x_2 w_{21} \\ &= -0.475 + 1(-0.725) - 1(-0.58) \\ &= -0.475 - 0.725 + 0.58 \\ &= -0.62 \end{aligned}$$

$$\begin{aligned} z_{in_2} &= b_2 + x_1 w_{12} + x_2 w_{22} \\ &= -0.575 + 1(-0.625) - 1(-0.525) \\ &= -0.675 \end{aligned}$$

$$z_{in_1} = -0.62 \quad z_1 = -1$$

$$z_{in_2} = -0.675 \therefore z_2 = -1$$

$$\begin{aligned} y_{in} &= b_3 + z_1 v_1 + z_2 v_2 \\ &= 0.5 + (-1)0.5 + (-1)(0.5) \\ &= -0.5 \end{aligned}$$

Since,  $t \neq y$

$$\begin{aligned} w_{11}(\text{new}) &= w_{11}(\text{old}) + \alpha(t - z_{in_1}) x_1 \\ &= -0.725 + 0.5(1 + 0.62) \\ &= 0.085 \end{aligned}$$

$$w_{12}(\text{new}) = w_{12}(\text{old}) + \alpha(t - z_{in_2})x_2$$
$$= -0.625 + 0.5(1 + 0.675)$$
$$= -1.4625$$

$$b_1(\text{new}) = b(\text{old}) + \alpha(t - z_{in_1})$$
$$= -0.475 + 0.5(1 + 0.625)$$
$$= 0.34$$

$$w_{21}(\text{new}) = w_{21}(\text{old}) + \alpha(t - z_{in_1})x_2$$
$$= -0.58 + 0.5(1 + 0.625) - 1$$
$$= -1.39$$

$$w_{22}(\text{new}) = w_{22}(\text{old}) + \alpha(t - z_{in_2})x_2$$
$$= -0.525 + 0.5(1 + 0.675) - 1$$
$$= -1.36$$

$$b_2(\text{new}) = b_2(\text{old}) + \alpha(t - z_{in_2})$$
$$= -0.575 + 0.5(1 + 0.675)$$
$$= 0.2625$$