Artificial Intelligence

Contributed By:
Dr. Sonal Sharma

Disclaimer

Artificial Intelligence

Topic:

Artificial Intelligence

Contributed By:

Dr. Sonal Sharma

—: "श्री गणेशाय नमः" :—

—: "सरस्वती नमः" :—

Date
29/6/18

Unit #1

Artificial Intelligence

Authors → Saroj Kau
John. Knig

Q. → What is AI ?

Ans. → Artificial Intelligence (AI) is the study of how to make computers do things which at the moment, people do better.

'OR'.

Artificial Intelligence is the ability of machi like computers to perform functions that nor mally require human intelligence.

Q. Which type of functions ?

Ans. These functions include the ability to learn, reason, analyse, take decisions, recognise speech & visual perception among others.

In simple terms, AI is the ability of S/w to develop & apply intelligence like huma (शक्ति)

☆ These two definitions are briefing the AI, coz of its reference to the current state of comp tet science.

But it also fails at some areas of potentially very large impact, namely problems that cannot now be solved well by either computers or people

Types of AI ⟶ On the basis of the definitio AI can be categorised under different heads

1. Algorithm ⟶ The computer executes a funct without deviating from the program or algm given to it. With an algm, we tell the computer exactly what we want it to do. Algorithms are used for data processing, calculations, & automated reasoning. There are also algm that let computers learn on their own.

2. Machine Learning ⟶ M.L. allows computers to learn without being completely programme Instead of using an algm to extract data for human use, the computer learns to analyse & interpret the data & make interferences.

3. Narrow AI ⟶ It is also called as Weak A I. This system is designed for one particular task & follows a set of rules without deviating (change) from it. Eg. → Apple's Siri, Microsoft's Cortana & Amazon's Alexa are all a form of weak A I. They are designed to answer questions they understand.

4] General AI ⟶ Also known as Artificial General Intelligence or strong AI, This system is programmed to perform human cognitive abilities which help it appl intelligence to find a solution to an unfamili task. By this, a computer repeatedly impro itself and can become an ultra-intelligent, superhuman machine that can surpass hum intelligence.

5] Bot (self running program) ⟶ This is a S/w designed to auto mate tasks that we would usually d on our own – like adding an appointment i the calender or making a reservation for din A common form of bot is chatbots that simulate conversations.

AI in everyday life

- Smartphones
- Google search
- Siri, cortana, Google Assistant → virtual persona assistant, they algm. & speech recogni to understand & respo
- Video Games
- Youtube, Ecommerce sites.

Other Examples, that may not come across in ou everyday lives

- Smart cars → Tesla, self drive car

The AI Problems

① The main problem of AI is "commonsense reasoning". It includes reasoning about physical objects & their relationship to each other, as well as reasoning about action & their consequences (eg. – if you let go of something it will fall to the floor & maybe break).

Three scientists – Newell, Shaw, Simon — built → General Problem Solver (GPS)

which they applied to several commonsense tasks, as well as to the problem of performing symbolic manipulations of logical expressions.

But again, no attempt was made to create a program with a large amount of knowledge about a particular problem domain.

② The second problem is the ability to use language to communicate a wide variety of ideas is perhaps the most important thing that separate humans from the others. & this problem is hard to solve.

Programs that can solve problems in these domains also fall under the age of AI.

Fig. 1.1 lists some of the tasks that are the targets of work in AI.

सांसारिक Mundane Tasks

- Perception
  - Vision
  - Speech
- Natural Language
  - Understanding
  - Generation
  - Translation
- Common sense Reasoning
- Robot control.

Formal Tasks

- Games
  - Chess
  - Checkers
- Mathematics
  - Geometry
  - Logic
  - Integral Calculus
  - Proving properties of programs.

Expert Tasks.

- Engineering
  - Design
  - fault finding
  - Manufacturing planning
- Scientific analysis
- Medical diagnosis
- Financial analysis.

fig. Some of the Tasks Domains of A.I.

Artificial Intelligence

Topic:

Intelligent Systems

Contributed By:

Dr. Sonal Sharma

Intelligent Systems ⟶ AI is a combination of computer science, physiology, and philosophy

John McCarthy was one of the founders of AI field who stated that AI is the science and engineering of making intelligent machines, especially intelligent computer programs. Different people think of AI differently and there is no unique definition.

The very first so-called intelligent system named "ELIZA" passed the Turing Test which was written by "Joseph Weizenbaum" during the period from 1964 to 1966.

ELIZA ⟶ It was a program that conversed with user in English. The program was able to converse about any subject, because it stored subject information in data banks.

The basic philosophy & characteristics in all the programs are the same. The main characteristics of Eliza are briefly mentioned here:-

i) Simulation of Intelligence ⟶ These programs are not intelligent at all in real sense. They do not understand the meaning of words or sentences or utterance.

ii) Quality of Response ——→ It is limited by the sophistication of the ways in which they can process the input at a syntac level.

iii) Coherence ——→ The earlier version of the system imposed no structure on the conversation. Each statement was based entirely on the current i/p & no context info was used.

iv) Semantics ——→ Such systems have no semantic representation of the content of either the user's i/p or the reply.

Categorization of Intelligent system ——→ In order to design intelligent systems, it is important to categorize these systems. There are four possible categories of such systems -

a.) System that thinks like humans.
b.) —"— acts —"—.
c.) —"— Thinks rationally. → based on logic.
d.) —"— acts rationally. → doing the right things. Even if the method is illogical, the observed behaviour must be rational.

Artificial Intelligence

Topic:

Components Of AI Programs

Contributed By:

Dr. Sonal Sharma

Components of AI programs ⟶ AI techniques must be independant of the problem domain as far as possible. Any AI problem or program should have knowledge base, & navigational capability which contains control strategy & interference mechanisms.

(learning nature)
(which rule to be applied)
(search through knowledge base)

Foundations of AI ⟶ Commonly used AI techniques & theories are rule based, fuzzy logic, neural networks, decision theory, statistics, probability theory, genetic algorithms etc. Since AI is interdisciplinary in nature, foundations of AI are in various fields such as -

* Mathematics
* Neuroscience
* Control theory
* Linguistics - speech demonstrates

Applications ⟶ AI finds applications in almost all areas of real-life applications. Broadly speaking, business, engineering, medicine education & manufacturing are the main areas. Instead of it fraud detection, object identification space shuttle scheduling, information

Expert System Architecture → (269) (5)

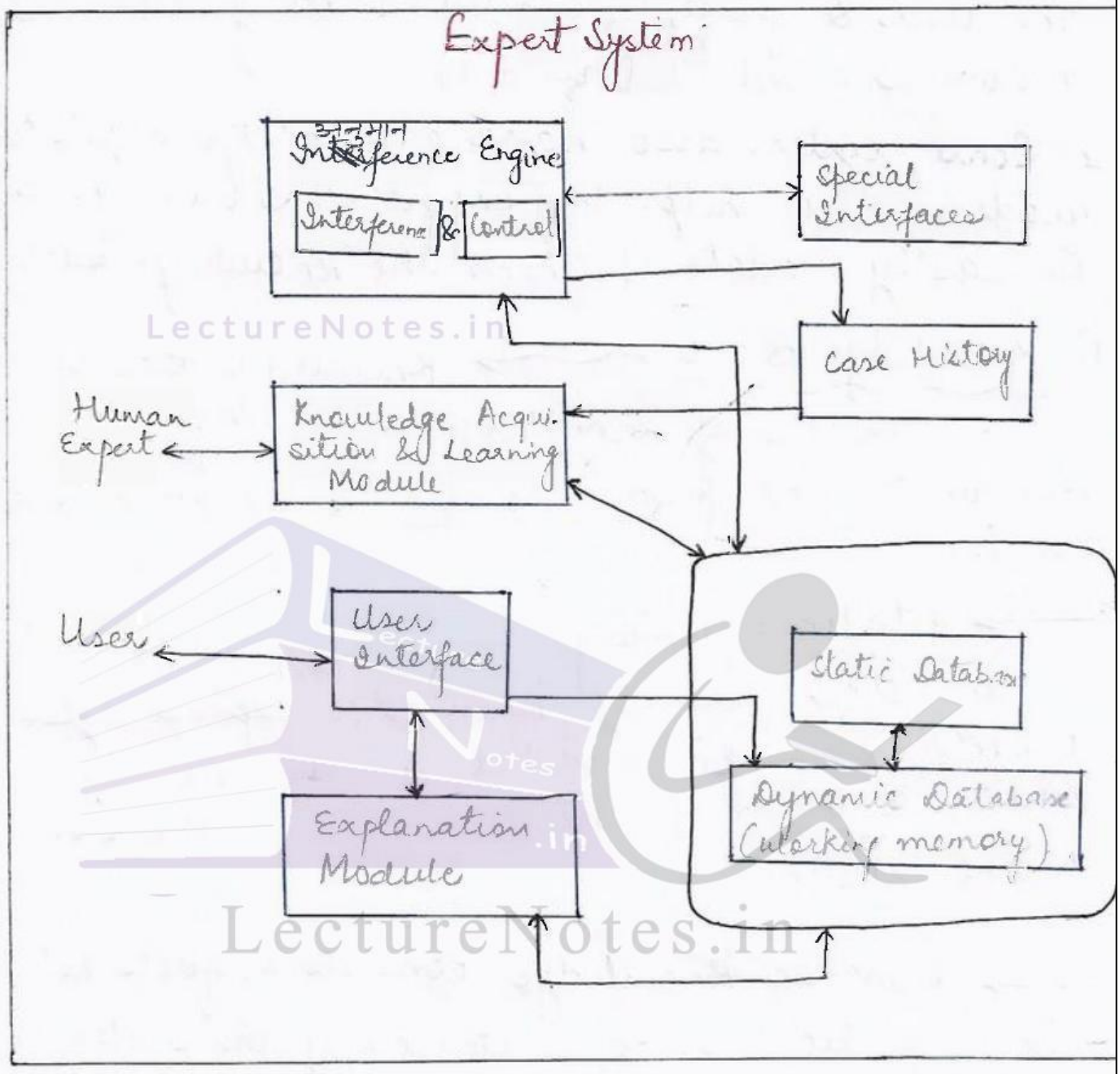


Fig. → Architecture of an Expert System.

* The user interacts with the system through a user interface which may use menus, natural language or any other other style of interactions.
* Then, an Inference engine is used to reason with the expert knowledge as well as the data specific to the problem being solved.

* Case specific data includes both data provided by the user & partial conclusions along with certainty measures based on this data.

* Some system also have a knowledge acquisition module that helps the expert or knowledge engineer to easily update & check the knowledge base.

1) Knowledge Base ⟶ Knowledge Base of an ES consists of knowledge regarding problem domain in the form of static & dynamic databases.

⟶ Static knowledge consists of rules and facts or any other form of knowledge representation which may be compiled as a part of the system & does not change during the execution of the system.

⟶ Dynamic knowledge consists of facts related to a particular consultation of the system collected by asking various questions to the user who is consulting the ES.

At the beginning of the consultation, the dynamic knowledge base (often called working memory) is empty.

As the consultation progresses, dynamic knowledge base (in the form of facts only) grows & is used in decision making along with static knowledge.

2.) Inference Engine ⟶ An inference engine developed for an ES consists of inference mechanisms & control strategy. The term inference refers to the process of sea ching through knowledge base & deriving n knowledge.

Inference Rule → if clause
→ then clause

enables expert systems to find solutions to diagnostic & perscriptive problems

Eg. If symptons are fever, cough & running nose, then patient has measles.

Eg. If symptom are headache, sneezing, runni nose, then patient ha cold.

Each rule is independent of others & may be deleted or added without affecting other rul Control strategy determines the order in whi rules are applied.

Inference Rules

Backward Chaining

⟶ It starts with goals of working & to see the data to allow the work to conclude any of these goals.

* → It searches the inference rules until it find "then". "If" part of that inference rule is not known to be true, then it ... added to the list of "goals".

Forward Chaining.

⟶ It starts with th available data & uses inference rules to con de more data until a desired goal is achie This method is also k as "Data driven metho

Eg. — [with pencil].

3) Knowledge Acquisition ⟶ Knowledge present in an ES may be obtained from many sources such as textbooks, reports, case studies, empirical data, and domain expert which are a prominent source of knowledge. A knowledge acquisition module allows the system to acquire more knowledge regarding the problem domain from experts.

Interaction between the knowledge engineer & the domain expert involves the strategies, methods, procedures, rules for solving the problem at hand. Later the knowledge can be updated by using knowledge acquisition module of the system.

4) Case History ⟶ It stores the files created by inference engine using the dynamic database & is used by the learning module to enrich its knowledge base.

5) User Interfaces ⟶ User interface of an ES allows user to communicate with the system in an interactive manner & helps the system in creating working knowledge for the problem that has to be solved.

function of User Interface → i) Present questions & Inf^m.
ii) Supply the responses of the user to the inference engine.

The user interface checks all responses to ensure that they are of the correct data type. Answers by the user must be checked with legal ans, if the answers are illegal, then a notification with the message can be sent to the user.

Eg.
⇒

System	Do you have fever?
User	Yes
System	Do you have some throat?
User	No
System	Do you have cough?
User	Yes
System	Are you suffering from running nose
User	Yes
System	Are you suffering from headache?
User	No

Table 1 → Dialogue module of a user Interface

{Answers in binary (yes/no) for simplification}.

* Dialogues are not preplanned in ES & there is no fixed control structure.

They are generated from the current info present in the system & the knowledge contained in the knowledge base.

6) Explanation Module ——→ Most of the ES ha

Explanation facilities that enable users to ques the system about why it asked some questions & how it reached some conclusions; these modules are call "How" & "Why".

Sub-module "How" tells the users regarding th process through which the system has reached a particular solution,

Sub module "Why" explains to the user, the reasoning behind arriving at a solution to any particular problem. These questions are answered by referring to the system goals, the rules being used, and any existing problem data.

Invoke	Why / How Modules
System	Do you have running nose?
User	Yes
System	I am trying to prove that you have cold. Do you have headache?
User	Yes
System	Do you have sore throat?
User	No
System	Then you do not have cold. Do you have fever?
User	Yes
System	Now I am looking at the possibility of measles. Do you have cough?
User	Yes
System	I can infer measles using the rule "If symptoms are fever, cough, the patient have measles. Concluded
User	How?
System	Since you have fever, cough & there is a rule so measles is concluded for you.

Table 2 → Explanation Module

7) Special Interfaces ⟶ It may be used in ES for performing specialized activities, such as handling uncertainty in knowledge.

Expert System versus Traditional Systems ⟶

(1) The basic difference b/w an ES and a TS is that an ES manipulates knowledge, whereas a TS manipulates data.

(2) The distinction b/w these systems lies in the manner in which the problem related expertise is coded into them.

a) In TS, problem expertise is encoded in program as well as in the form of data structures

b) In ES, all problems related expertise is encoded in data structures only & not in the programs.

(3) In ES, small fragments of human experience are collected into a knowledge base which are used to reason through a problem.

Another advantage of ES over traditional system is that they allow the use of confidence & certainty factors. The use of words such as if, then, might, probably, etc., indicate that there is some uncertainty involved in the system. This type of reasoning can be imitated by using numeric values called confidences in ES.

Artificial Intelligence

Topic:

Characterstics Of Expert Systems

Contributed By:

Dr. Sonal Sharma

Characteristics of Expert Systems ⟶ Some key characteristics that every ES must posses are described as follows –

i) Expertise ⟶ have high level of skills, & possess adequate robustness.

ii) Symbolic Reasoning ⟶ Knowledge in an ES is represented symbolically which can be easily reformulated & reasoned.

iii) Self Knowledge ⟶ A system should be able to explain & examine its own reasoning

iv) Learning Capability ⟶ A system should learn from its mistakes and mature as it grows. Flexibility provided by an ES helps it grow incrementally.

v) Ability to provide Training ⟶ Every ES should be capable of providing training by explaining the reasoning process behind solving a particular problem using relevant knowledge.

vi) Predictive modelling power ⟶ This is one of important feature of ES. The system can act as an information processing model of problem solving It can explain how new situation led to the change which helps users to evaluate the effect of new facts & understand their relationship to the solution

Evaluation of Expert System ⟶ Evaluation

of an ES consists of performance & utility evaluation.

Advantages and Disadvantages of ES ⟶

i) Guidelines of designing of an ES ⟶

(a) Specialized knowledge problems ⟶

(b) High payoff

(c) Existence of cooperative experts.

(d) Justification of cost involved in developing ES.

(e) The type of problem.

Advantages ⟶

1. Helps in preservating scarce expertise.
2. Provides consistent answers for repetitive decisions, processes & tasks.
3. fastens the pace of human professional or semi-professional works.
4. Holds & maintains significant levels of inf^{m}
5. Provides improved quality of decision making.
6. Domain experts are not always able to explain their logic & reasoning like ES.
7. Causes introduction of new products.
8. Never forgets to ask a question, unlike a human
9. Leads to major internal cost saving within

Disadvantages ⟶

i) Unable to make creative responses as human experts would in unusual circumstances.
ii) Lacks common sense needed in some decision making.
iii) May cause errors in the knowledge base, & lead to wrong decisions.
iv) Cannot adapt to changing environments, unless knowledge base is changed.

Rules Based Expert Systems ⟶ Rule based systems can be either goal driven (using backward chaining to test whether a given hypothesis is true) or data driven (using forward chaining to draw new conclusions from existing data). Expert system may use either one or both strategies, but the most common is probably the goal-driven / backward-chaining strategy. One reason for this is that normally an ES will have to collect info^m about the problem from the user by asking questions. However, in case of a goal-driven strategy, we can just ask questions that are relevant to a hypothesized solution.

1] Expert System shell in Prolog ⟶

[Prolog = Programming in Logic]

Prolog provides backward chaining

To define a special syntax for the rules using operator declaration (:-op).

Using op, any standard system operator declaration can be changed or new operator can be defined by the user.

:- op (Prec, Type, Atom)

Prec: unsigned integer [1, 1200]

Type: Type ∈ {fx, fy, xfx, xfy, yfx, yfy} is a associativity.

Atom: symbol or a name declared as an oper

Artificial Intelligence

Topic:

Knowledge Representation

Contributed By:

Dr. Sonal Sharma

-: "श्री गणेशाय नमः" :-
-: "सरस्वती नमः" :-

Unit #2

Part-I

Knowledge Representation

Introduction ⟶ It is an important issue in both cognitive science & AI. In cognitive science, it is concerned with the way in which infⁿ is stored & processed by humans.

In AI, the main focus is on storing knowledg or infⁿ in such a manner that programs ca process it and achieve human intelligence.

In AI, knowledge representation is an imp. area because intelligent problem solving can be achieved and simplified by choosing an appro priate knowledge representation technique.

A no. of representation have been devised to structure infⁿ.

Any knowledge representation system should possess properties such as learning, efficiency in acquisition, representation adequacy, & inferential adequacy. These properties may be defined as follows –

i) Learning refers to a capability that acquires new knowledge, behaviours, understanding etc. It does not simply involve adding new facts to a knowledge base but new inf^{m} may have to be classified to avoid redundancy & replication in the existing knowledge prior to storage to enable easy retrieval.

ii) Efficiency in acquisition refers to the ability to acquire new knowledge using automatic methods wherever possible rather than relying on human intervention.

iii) Representational adequacy refers to the ability to represent the required knowledge

iv) Inferential adequacy refers to the ability of manipulating knowledge to produce new knowledge from the existing one.

Efficiency of a method depends greatly on the representation scheme of the knowledge. Many AI methods have tried to model human intelligence KR is a core component of a no. of app^{n} such as ES, Machine translation systems, computer aided maintenance system, Inf^{m} retrieval system, & database front-ends.

Approached to Knowledge Representation

AI programs use structures known as knowledg structures to represent objects, facts, relationshi & procedures. The main function of these knowledge structures is to provide expertise & ingm so that a program can operate in an intel igent way.

KS composed of → Traditional Structure (+) such as {Semantic nw, frames, scripts, Conceptual dependency structu
→ Complex Structure.

A KBase is a special type of DB that holds the knowledge of the domain.

1.] Relational Knowledge → RK comprise objects consisting of attributes & associate values. This one is the simplest way to stored facts. In this method, each fact is stored in a row of relational table as done in relational database. A table is defined as a set of data values that is organized using a model of horizontal rows & vertical columns
corresponding values
attribute na

Name	Age (in yrs)	Sex	Qualification	Salary (in Rs)
John	38	Male	Graduate	20000
Mike	25	Male	Undergraduate	15000
Mary	30	female	Ph.D.	25000
James	29	Male	Graduate	18000

Table. → Relational Table.

With the help of table, we can analyze the related data of a particular attribute.

And also we can obtain the answers of the follow queries like —

a) What is the age of John ?

b) How much does Mary earn ?

c) What is the qualification of Mike ?

But, we cannot obtain the answers of the unrelat questions of this table.

Eg. Does a person having a Ph-D. qualification ear more ?

So, inferencing new knowledge is not possible fro such structures.

2] Knowledge Representation as Logic ——→ Infer

tial capability can be achieved if knowledge is represented in the form of formal logic.

For eg. - knowledge regarding mortality such as "All humans are mortal" cannot be represented using relational approach; instead, it can be easily represented in predicate logic as follows -

$(\forall X)$ human $(X) \leftarrow$ mortal (X).

Eg. If John is a human, then we can easily infe that John is mortal.

Advantage of this approach — we can represent a set of rules, derive more facts, truths, & verify the correctness of new statements.

3. Procedural knowledge ⟶ Procedural

Knowledge is encoded in the form of procedures which carry out specific tasks based on relevant knowledge.

for eg. — an interpreter of an programming language interprets a program on the bases of the available knowledge regarding the syntax and semantics of the language.

Advantage of this approach — domain specific knowledge can be easily represented & side effects of actions may also be modelled

Problem of - Completeness (all cases may not be represented) & Consistency (all deductions may not be correct)

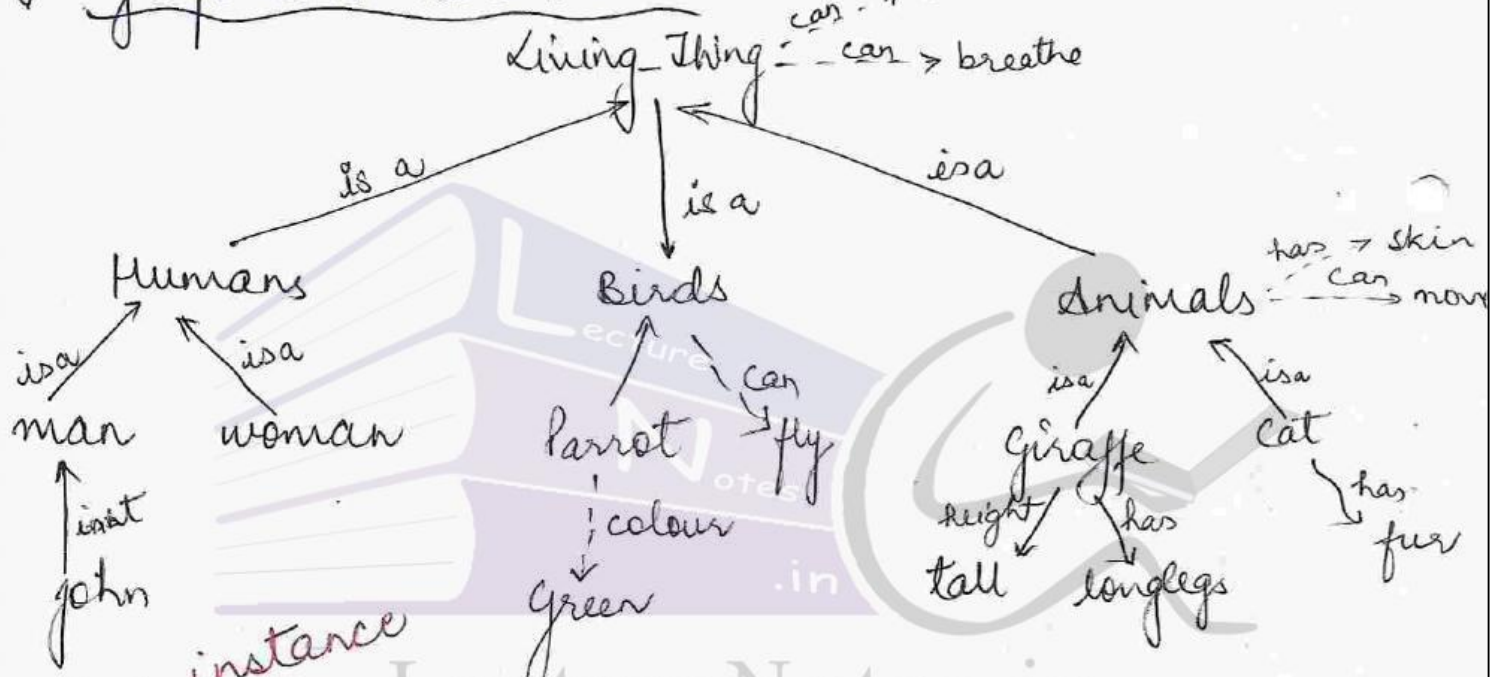
Knowledge Representation using semantic network

The basic idea applied behind using a semantic nw is that the meaning of a concept is derived from its relationship with other concepts, & that the info is stored by interconnecting nodes with labelled arcs.

for eg. → Every human, animal, & birds are living things that can breathe & eat.
All birds — fly ; Man & Woman — 2 legs ;
All animals — Skin ; Parrot — Bird is green

We can represent such knowledge using a structure called a semantic n/w (or semantic net)

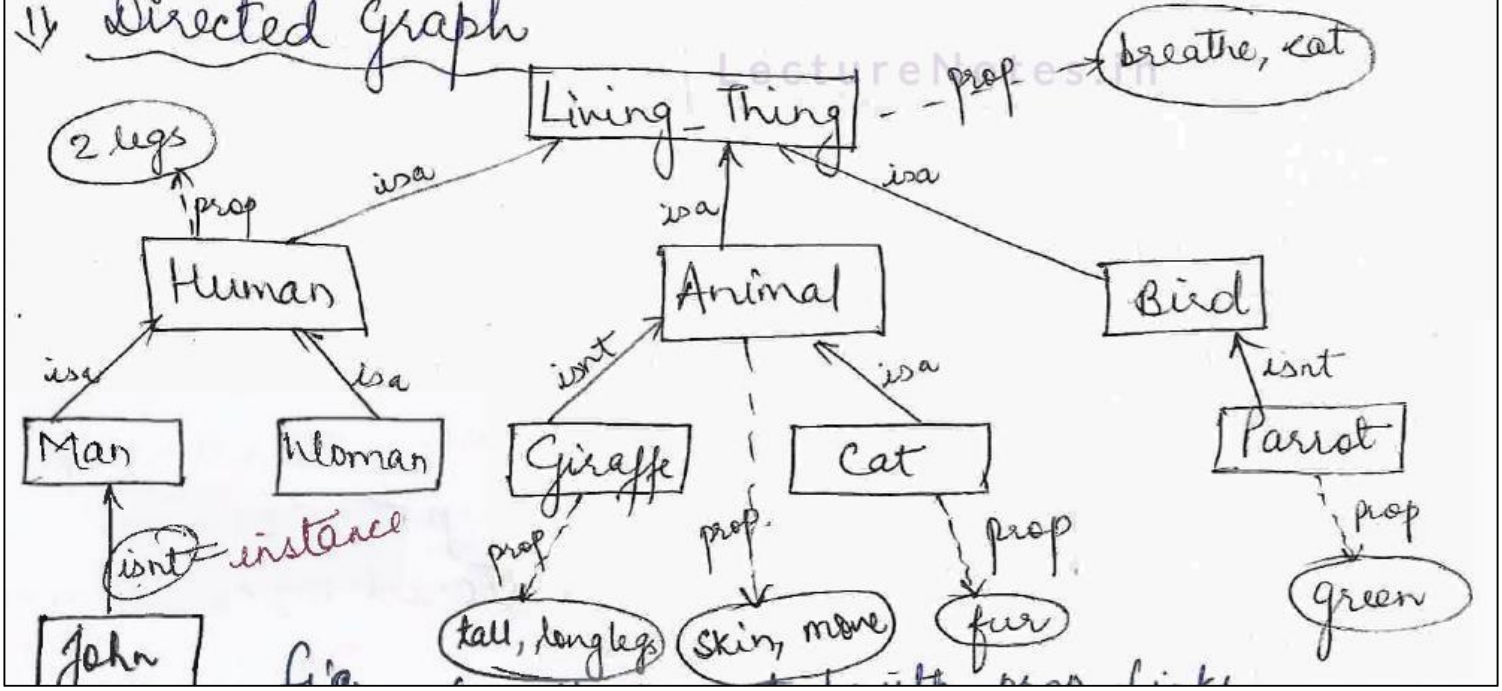
i) Graphical Notation



inst = instance

fig. → KR using Semantic Net.

ii) Directed Graph



is a ⟶ It connects 2 classes, where one concept is a kind or subclass of the other class.

inst ⟶ It relates specific members of a class.

Other relations such as {can, has, colour, height} are known as property relations

(i) Inheritance in semantic net ⟶ Hierarchical structure of KR allows knowledge to be stored at the highest possible level of abstraction which reduces the size of the knowledge base. It also helps us to maintain the consistency of the knowledge base by adding new concepts & members to existing ones.

Algorithm ⟶ Property Inheritance Algorithm

Input ⟶ Object & property to be found from semantic Net;

Output ⟶ returns yes, if the object has the desired property else returns false;

Procedure ⟶

- find an object in the semantic net;
- found = false;
- While [(object ≠ root) OR found] Do
  {
  - If there is an attribute attached with an object then found = true;
    else {object = isa (object, class) or
    object = inst (object, class)};
  };
- If found = true then print 'Yes' else print 'No';

Semantic net can be implemented in any programming language along with an inheritance procedure implemented explicitly in that language. The implementation of semantic net is shown in previous fig.

Prolog facts ⟶ The facts in prolog would be written as shown in Table-

Isa facts	Instance facts	Property facts
isa (living_thing, nil)	inst (john, man)	prop (breathe, living_thing)
isa (human, living_thing)	inst (giraffe, animal)	prop (eat, living_thing)
isa (animal, living_thing)	inst (parrot, bird)	prop (two_legs, human)
isa (birds, living_thing)		prop (move, animal)
isa (man, human)		prop (skin, animal)
isa (woman, human)		prop (fur, bird)
isa (cat, animal)		prop (tall, giraffe)
		prop (long_legs, giraffe)
		prop (tall, animal)
		prop (green, parrot)

Table → Prolog facts

Inheritance Rules in Prolog. ⟶ In class hierarchy structure, a member subclass of a class is also a member of all super classes connected through isa link

for eg. → if man is a member of subclass human, then man is also member of living_class.

Various queries can be answered by the following inheritance program.

Instance rules

```
instance (X,Y) :- inst(X,Y).
instance (X,Y) :- inst (X,Z), subclass(Z,Y)
```

Subclass rules

```
subclass (X,Y) :- isa (X,Y)
subclass (X,Y) :- isa (X,Z), subclass (Z,Y).
```

Property rules

```
property (X,Y) :- prop (X,Y)
property (X,Y) :- instance (Y,Z) property (X,Z)
property (X,Y) :- subclass (Y,Z), property (X,Z).
```

English Query	Prolog Goal	Output
Is john human?	?- instance (john, humans)	Yes
Is parrot a living thing?	?- instance (parrot, livingthings)	Yes
Is giraffe an animal?	?- instance (giraffe, animal)	Yes
Is woman a subclass of L.T?	?- subclass (woman, living thing)	Yes
Does parrot fly?	?- property (fly, parrot)	Yes
Does parrot have fur?	?- Does parrot have fur?	No
Does john breathe?	?- property (john, breathe)	Yes
Does cat fly?	?- property (fly, cat)	No

Table. — Various queries for Inheritance Program

Extended Semantic N/w for KR ——→ Logic

and semantic n/w are two different formalisms that can be used for knowledge representations. Simple semantic n/w is represented as a directed graph whose nodes represent concepts or objects and arks represent relationships b/w concepts or objects.

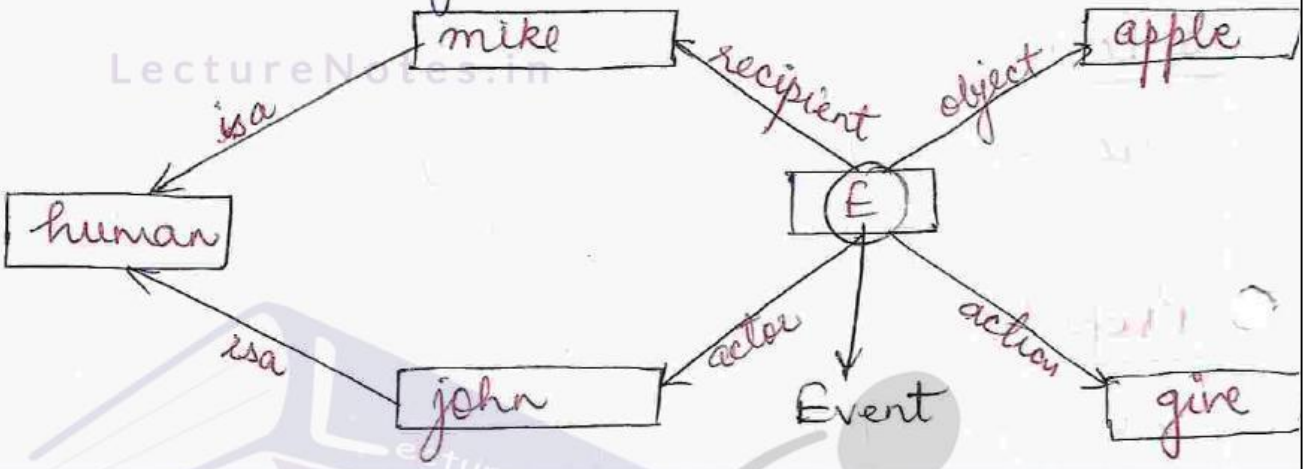


fig → Semantic net.

'E' represents an event which is an act of giving, whose actor is john, the object is an apple, & recipient is mike.

In this eg., 'john gives an apple to mike' is easily represented in predicate logic by

give (john, mike, apple).

here

john, mike, apple — arguements
given → predicate relation.

for eg. → the sentence 'john gives an apple to everyone he likes' is expressed in predicate logic

give (john, X, apple) ← likes (john, X)

Here the symbol 'X' is a variable representing any individual. The arrow represents the logical connective implied by.

left side of ← contains conclusion
right side of ← contains conditions.

In conventional semantics n/w, we cannot express clausal form of logic. To overcome this shortcoming 'R Kowalski' & his colleagues (1979) proposed an "Extended Semantic Network" [ESNet]; that combines the advantages of both logic & semantic n/w.

Conclusions & conditions of clausal form are represented in ESNet by different kinds of arcs

- - - -> conditions ⇝ denial links

——→ conclusions ⇝ assertion links

Eg

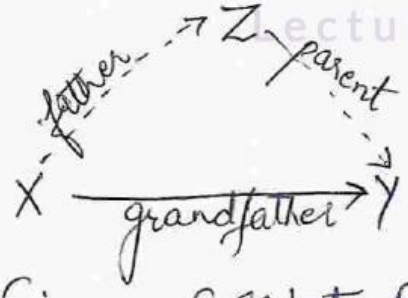


fig. → ESNet Representation

grandfather(X,Y) ← father(X,Z), parent(Z,Y)

→ clausal Representation

Eg. male(x), female(x) ← human(x),
can be represented using binary representation as
isa (x, male), isa (x, female) ← isa (x, human)

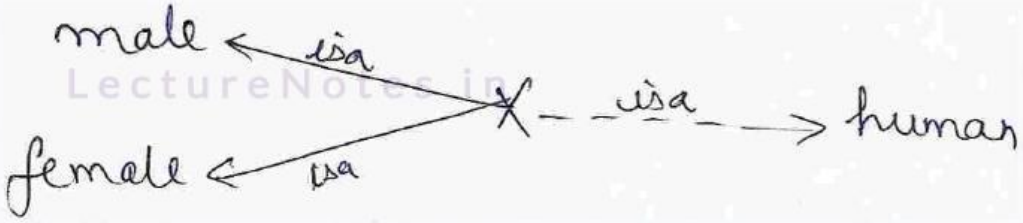


fig → ES Net Representation.

(i) Inference Rules ⟶ Inference rules are embedded in the representation itself.

a. The representation of the inference for every action of 'giving', there is an action of 'taking' in clausal logic is action (f(x), take) ← action (x, Give)

The representation in ES Net is –

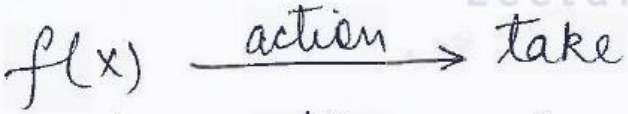


fig. – ES Net Representation.

b. The inference rule that an action who performs a 'taking' action is also the 'recipient' of this action & can easily be represented in clausal logic.

recipient(E, X) ← action(E, take), actor(E, X)

ESNet representation is as follows –

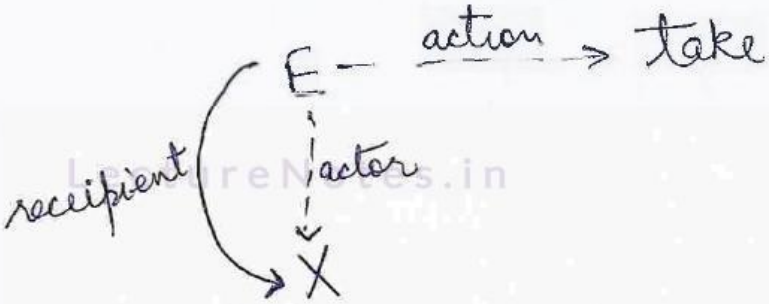


fig – ESNet Representation.

Eg. → Represent the following clauses in ESNet

recipient (E, X) ← action (E, take), actor (E, X)
object (e, apple)
action (e, take)
actor (e, john)

Solution → E = variable for some event
e = actual event

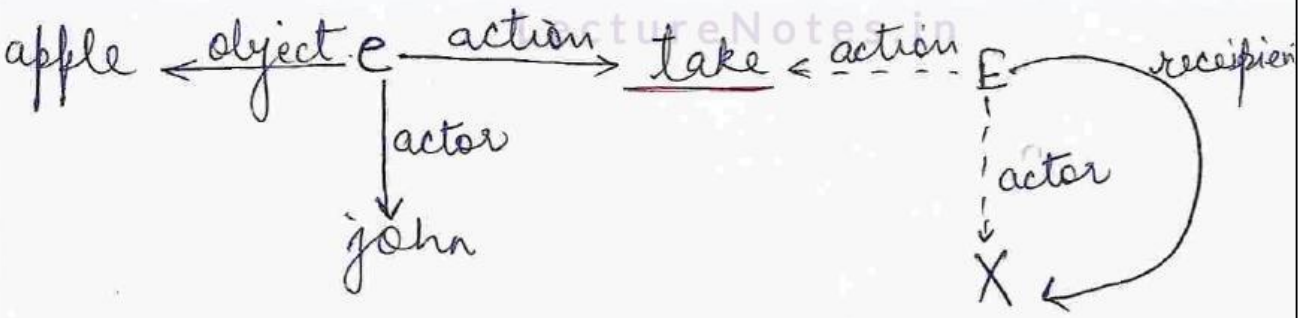
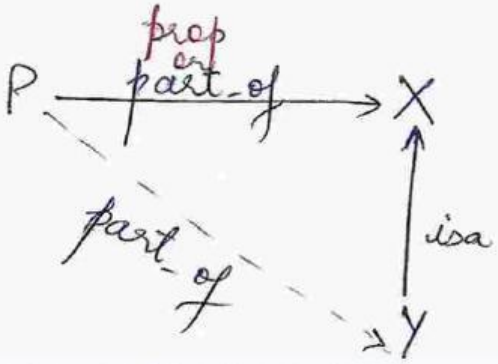


Fig. → ESNet Representation

c] The contradiction of ESNet can be represented in fig.

Eg.



Here P part_of X is conclusion and P part_of Y is condition, where Y is linked with X via 'isa' link. Such kind of representation is contradictory & hence there is a contradiction in E:SNet.

(ii) Deduction in Extended Semantic Networks ⟶

In logic, there are two types of inference mechanisms, —

a) forward reasoning inference mechanism
b) backward reasoning inference mechanism

a) Forward Reasoning Inference ⟶ Also called "bottom-up approach" in clausal logic derives new assertion from old ones.

Given an ESNet, apply the following reduction (resolution) using modus ponen rule of logic {i.e., given (A ← B) & B, then conclude A}.

for eg. — isa (X, human) ← isa (X, man)
isa ('john, man)

Modus Ponen Rule ⟶ It can be summarized as "P implies Q and P are both asserted to be true, so therefore Q must be true".

Eg. If today is Tuesday, then John will go to work
- Today is Tuesday
- Therefore, John will go to work.

Using modus ponen rule of logic, we can easily derive that isa (john, human) holds true.

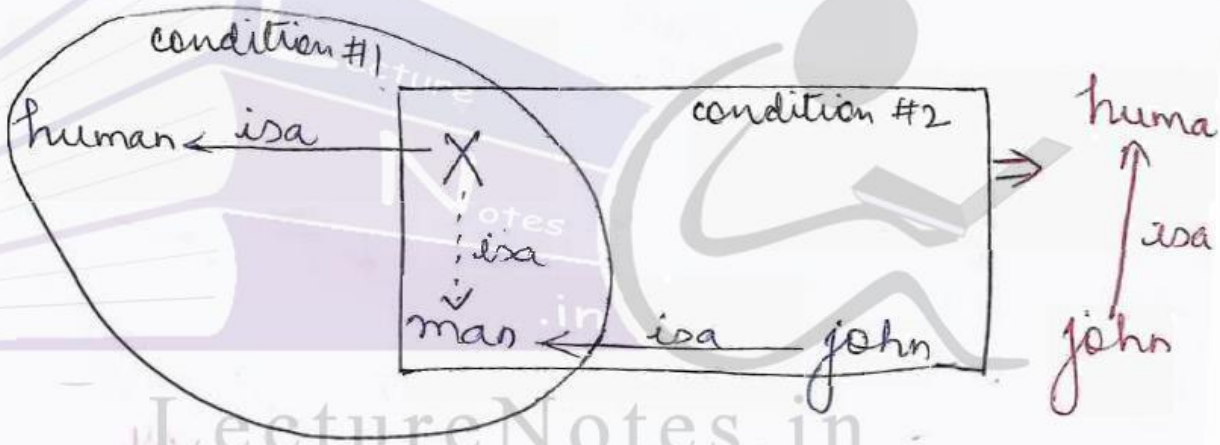


Fig → forward reasoning inference mechanism

Given set of clauses	Inferencing
isa (X, human) ← isa (X, man)	isa (john, human)
isa (john, man)	

Here the contradiction is enclosed in a rectangular box.

b) Backward Reasoning Inference ⟶ Also called "top-down approach". Here, we can prove a conclusion or goal from a given ESNet by adding the denial of the conclusions to the n/w & show that the resulting set of clauses in the n/w gives contradiction. This is done by performing successive steps of resolution until an explicit contradiction is generated.

Eg. ⟶

Given set of clauses	Prove conclusions
isa(X, human) ← isa(X, man) isa(john, man)	Query: isa(john, human)

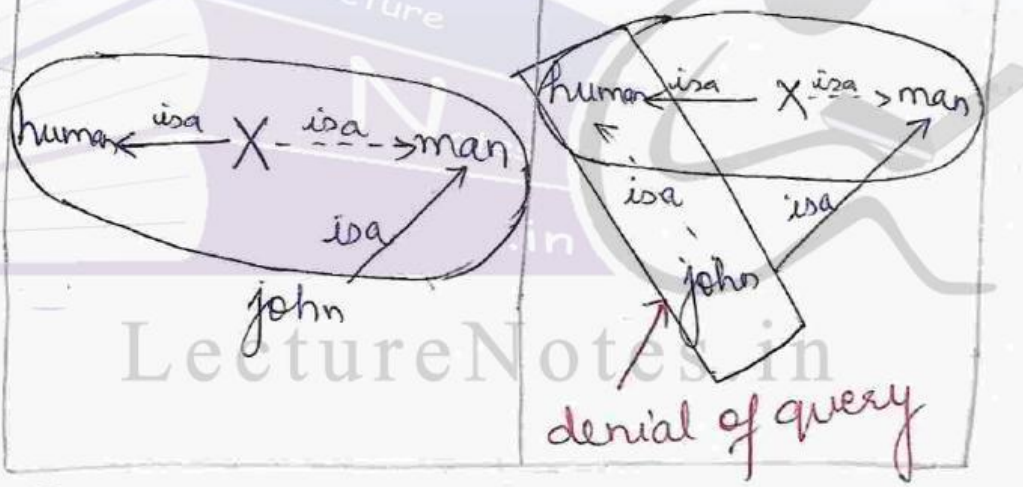
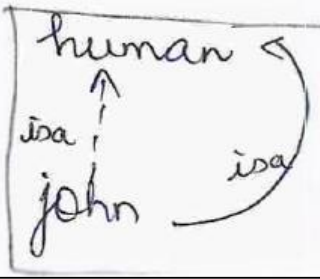


Fig. – Backward Reasoning Inferencing

After adding denial link in ESNet, we get the reduction in ESNet by elimination of assertion & their denial link with the help of appropriate substitution.



X=john

Contradiction or empty n/w is generated. Hence, isa(john, human) is proved.

Fig. ⟶ Reduction of ESNet

Example → Consider the following example, represented in clausal form

isa (X, living thing) ← isa (X, animate)
isa (X, animate) ← isa (X, human)
isa (X, human) ← isa (X, man)
isa (john, man)

Solution → Corresponding ESNet representation

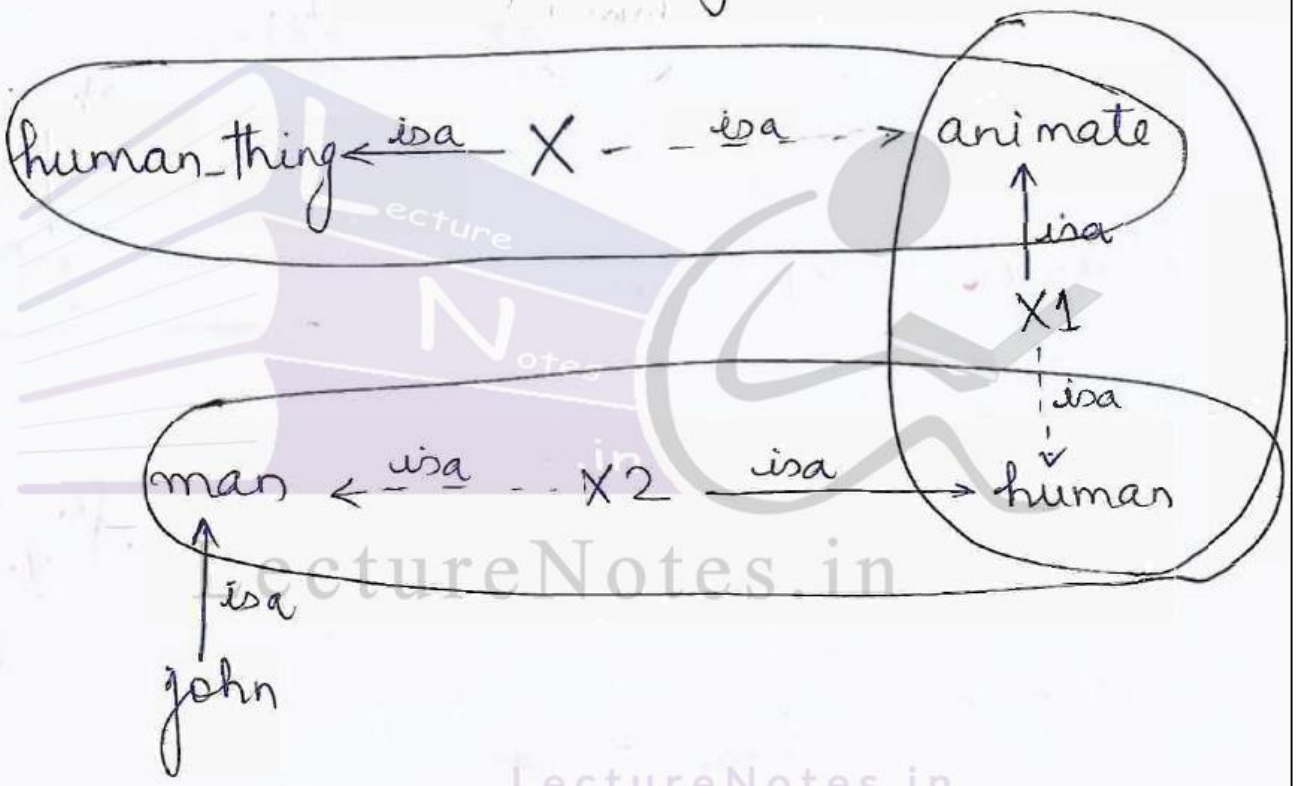
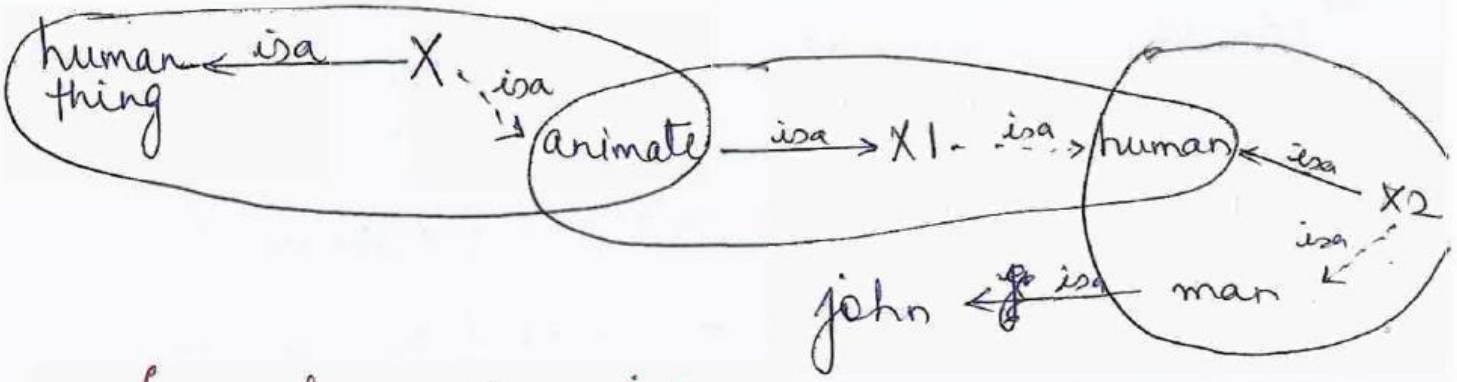
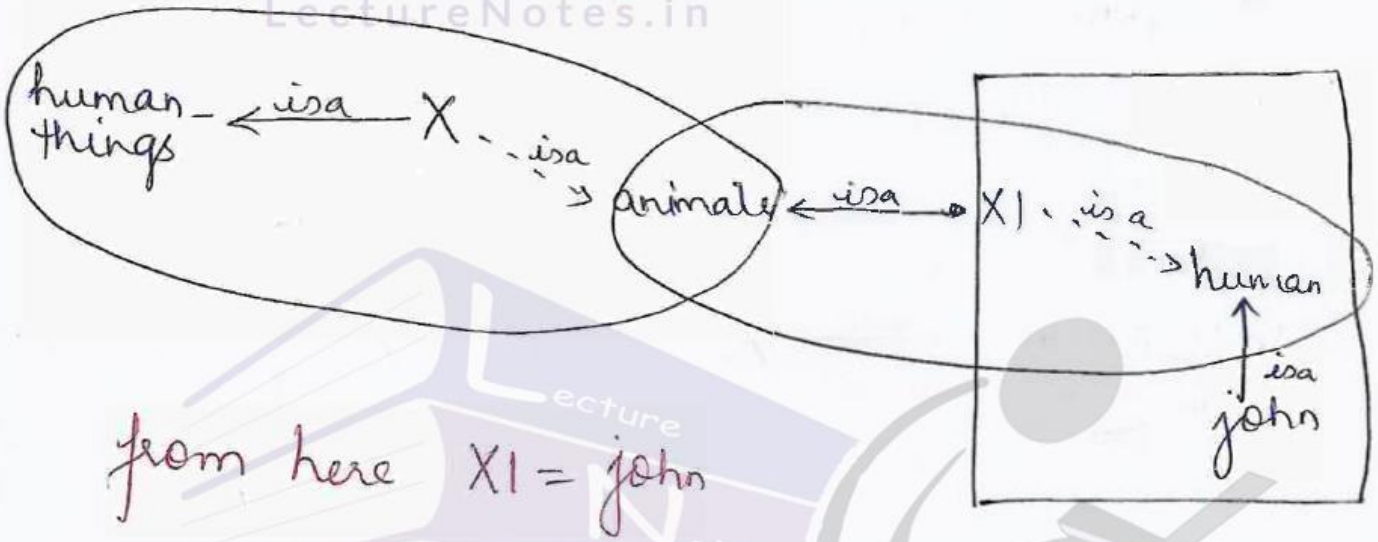


fig → ESNet Representation

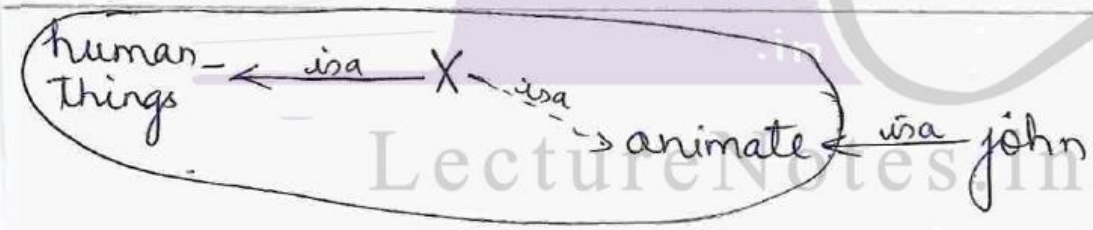
a) Forward Reasoning Inference → With the help of this, we can prove that john is a living_thing.



from here X2 = john

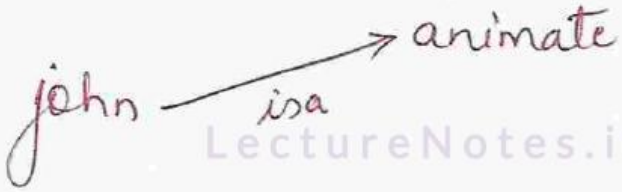


from here X1 = john



from here X = john

from here



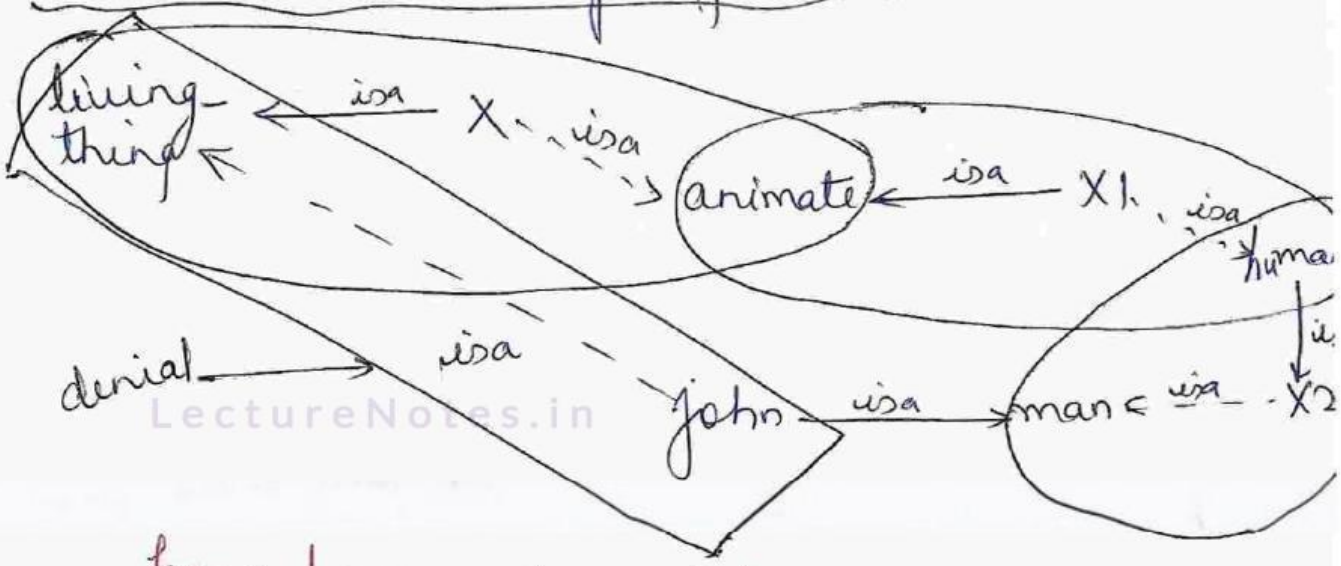
∴ john is animate : isa (john, animate) is inferred.

Human_things ← isa — john.

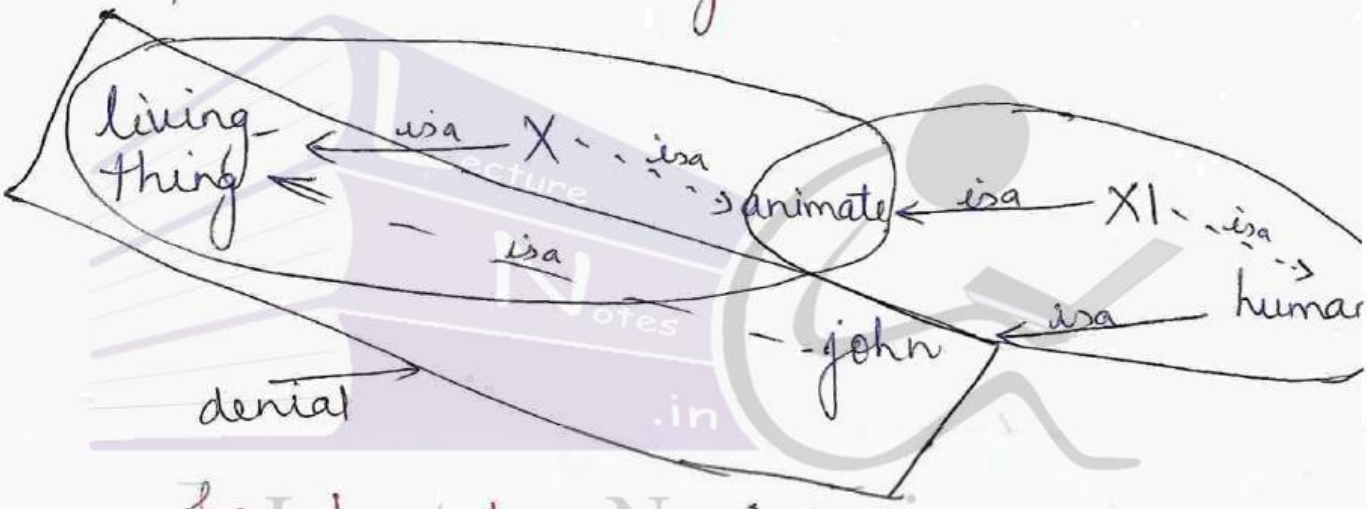
fig. → Derivation of john is animate.

further, john is living_thing can be derived by binding X with john.

b) Backward Reasoning Inference ⟶



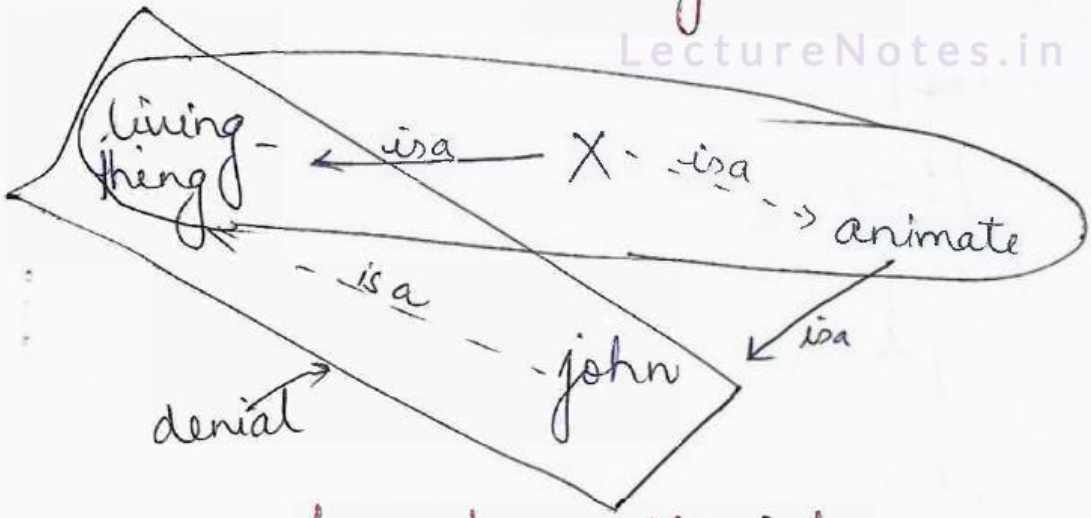
from here X2 = john



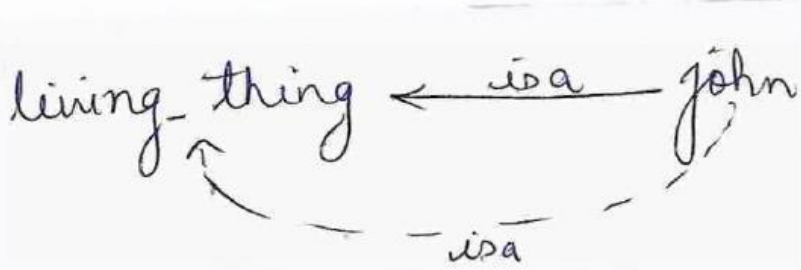
from here human = john

"OR"

X1 = john



from here X = john



leads to empty clause or contradiction.

fig → Solving query isa(john, living_thing)

Example 2. → The sentence, "Anyone who gives something he likes to a person likes that person" al John gives an apple to Mike. John likes an apple" - can be expressed in both library clausal form & ES Net representation as given below.

Solution → Clausal Representation

likes (X, Z) ← action (E, give), object (E, Y), actor (E, X), recipient (E, Z), likes (X, Y)

action (e, give)
object (e, apple)
actor (e, john)
recipient (e, Mike)
likes (john, apple).

E → variable of an actual event
e → process of resolution.

ES Net Representation

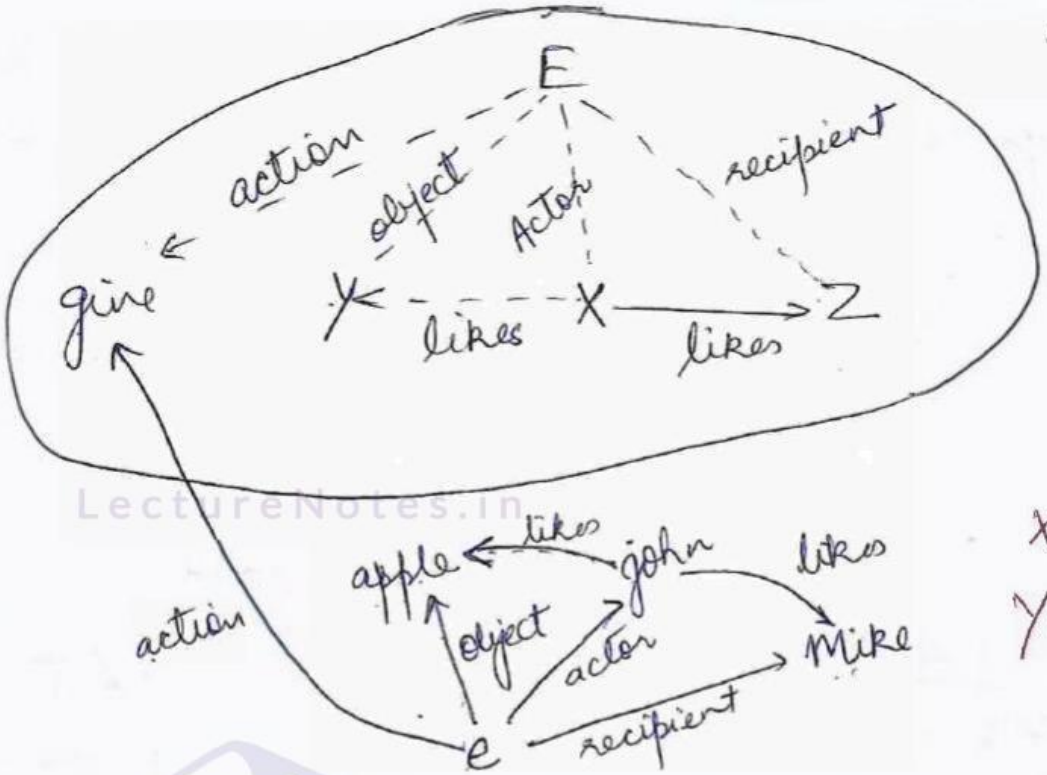


fig → ESNet Representation

a) Forward Reasoning Inference → Here we have to reduce the n/w by resolving clauses till we obtain the desired assertion.

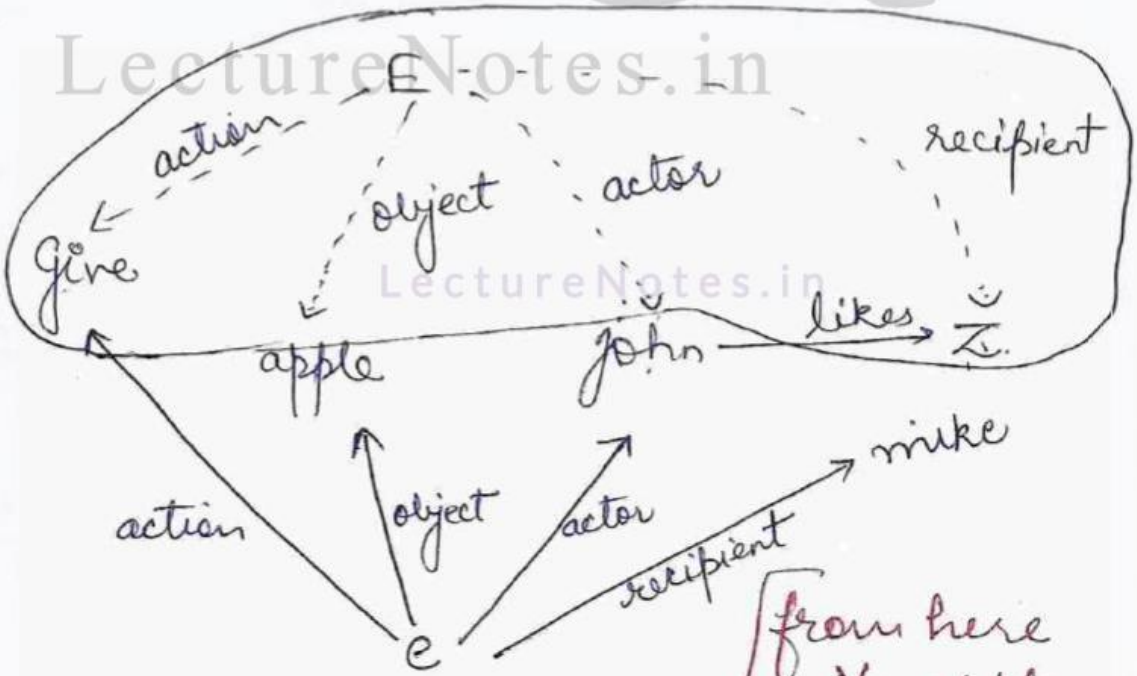


fig → Reduced ESNet

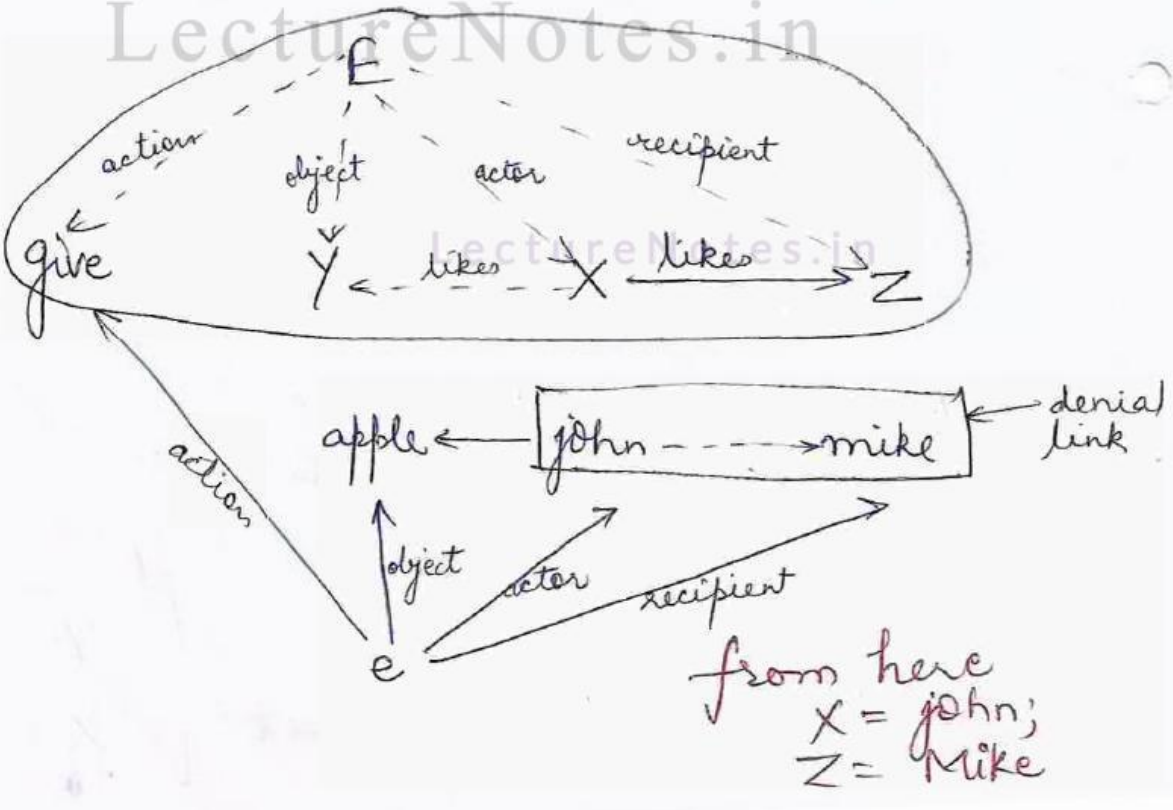
The n/w shown in fig is further reduced to the following conclusions by unifying E = e and Z = mike.

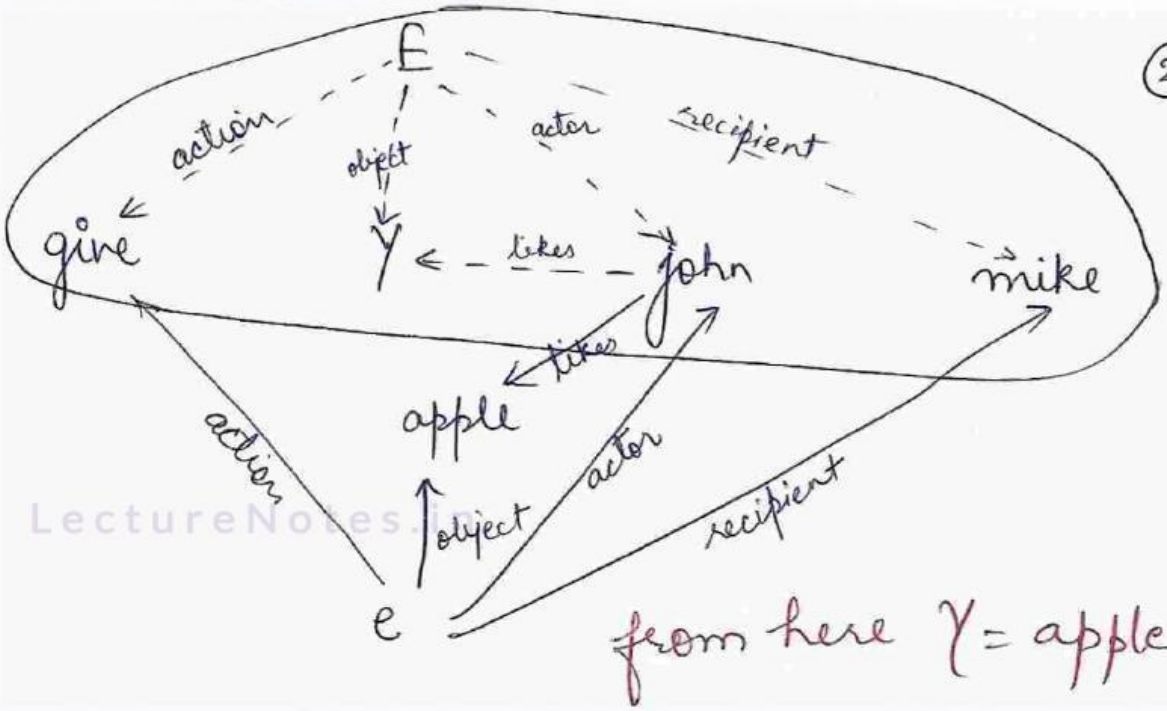
E = e ; Z = mike

john —likes→ Mike

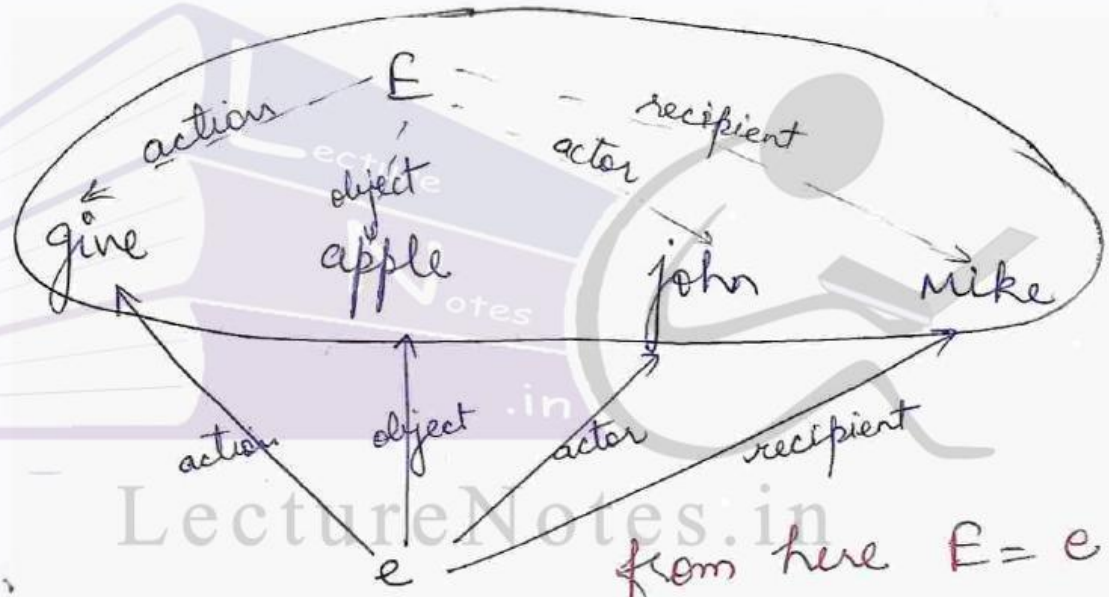
b) Backward Reasoning Inference → Let us now prove likes (john, mike) using backward reasoning method. In order to prove likes (john, mike) add the denial link to the n/w & try to reduce the n/w to empty.

The denial link is john —likes→ mike





from here Y = apple



from here E = e

we get empty n/w

Fig. → Reduction to Empty ESNet.

When 'E' is unified 'e', then entire ESNet is reduced to contradiction & hence, the query likes (john, mike) is proved using backward reasoning inference process.

(iv) **Inheritance** ⟶ In a conventional semantic n/w, lower level nodes in isa heirarchy inherit properties from higher level nodes unless the properties are redefined in the node itself. Consider the following logic program

Eg
isa (X, living_thing) ← isa (X, animate)
isa (X, animate) ← isa (X, human)
isa (X, human) ← isa (X, man)
isa (john, man)
→ part_of (human, two_legs)

Soln

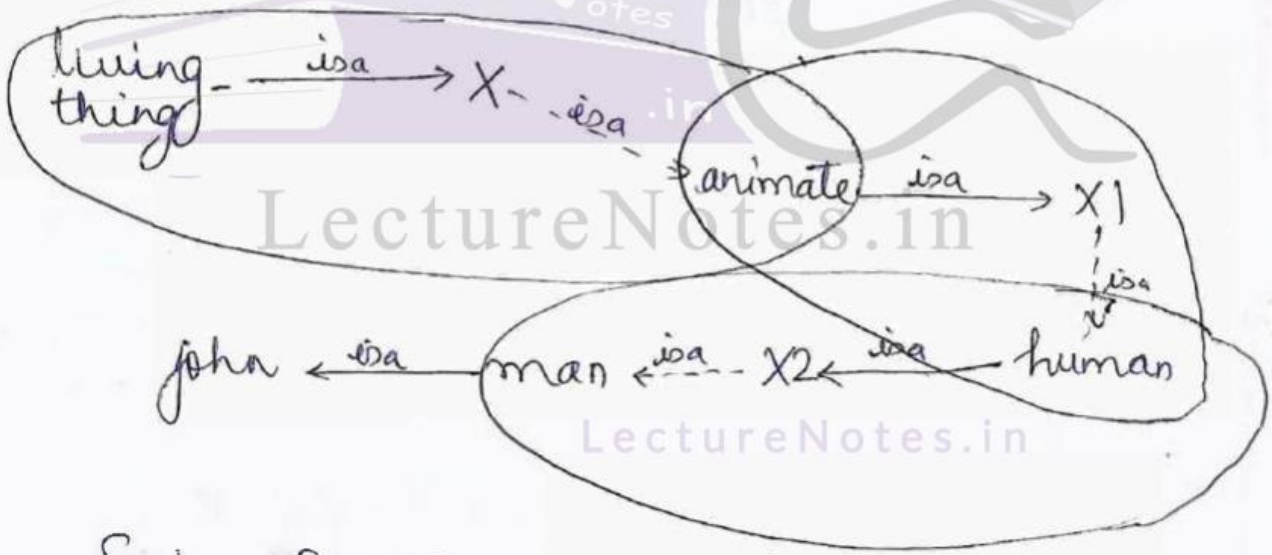


Fig. → ES Net for inheritance.

In order to show that "john has 2 legs", we add a denial link of "john has 2 legs" to the n/w & use general backward reasoning inference mechanisms & try to get a contradiction.

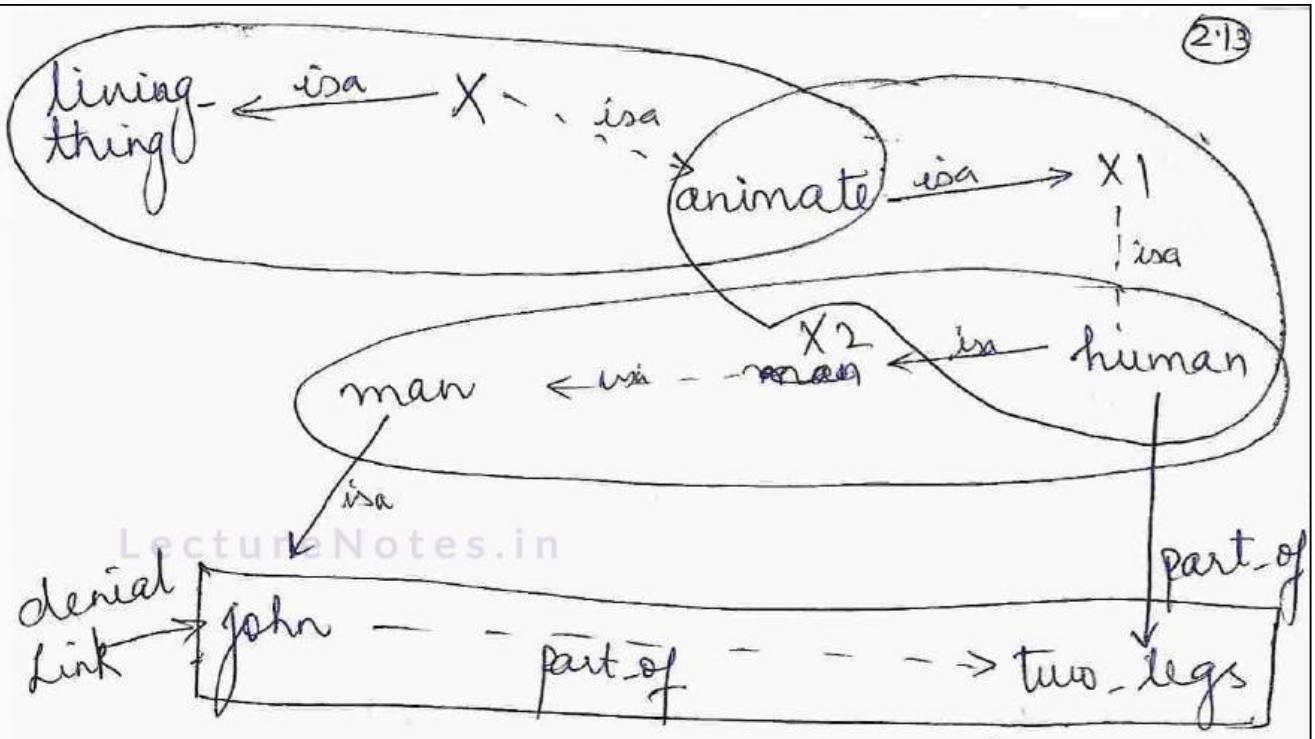


fig → Addition of a Denial link to the n/w

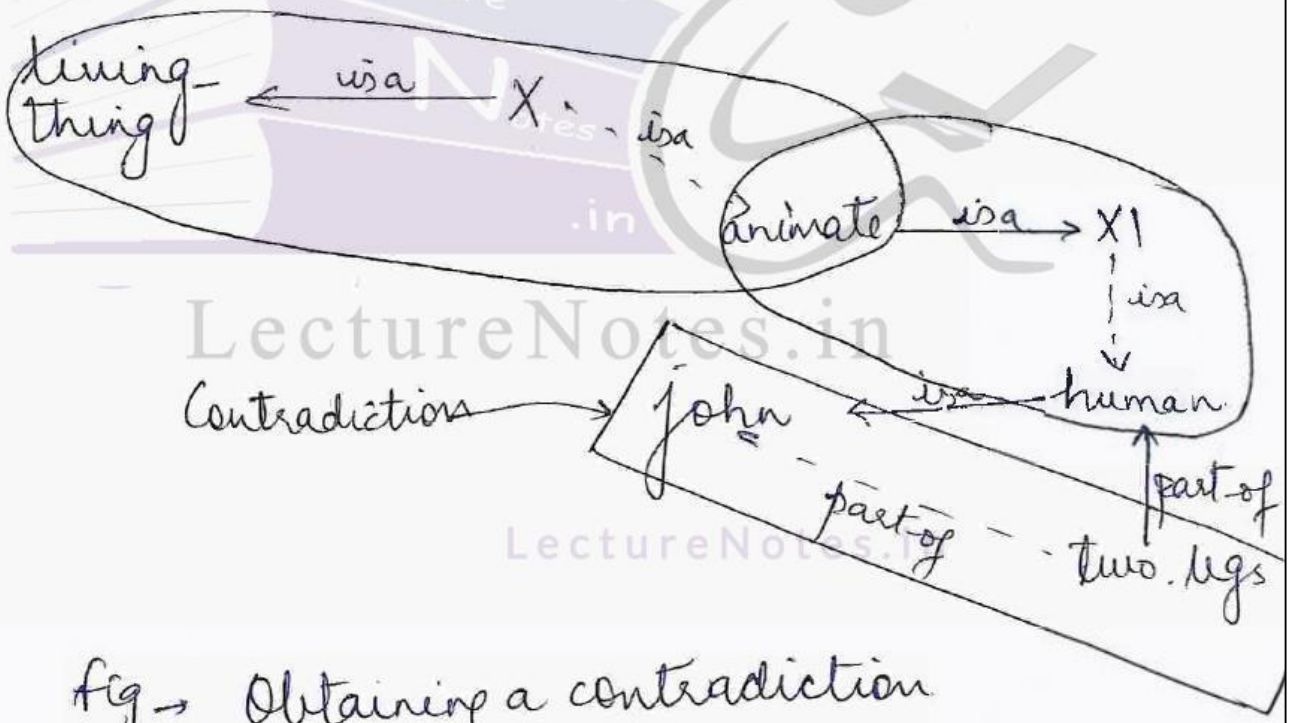


fig → Obtaining a contradiction

Implementation ⟶ Implementation of ESNet can be done in any programming language or using a tool which facilitates implementation of semantic n/w.

Explicitly ⇝ adding them to the n/w.

Implicitly ⇝ structure sharing method.

A major difference b/w conventional semantic n/w & extended semantic n/w is as follows :-

i) In Conventional semantic n/w, procedures are generally written in the host programming language.

ii) In ESNet, procedures are integrated with the rest of the database & are executed by the same general purpose mechanism which performs inference in the n/w.

Artificial Intelligence

Topic:

Knowledge Representation Using Frames

Contributed By:

Dr. Sonal Sharma

Knowledge Representation Using Frames

Many of the ideas about frames & frame system & how they can be used for the process of knowledge representation were first introduced by "Marvin Minsky (1975)".

Frames are regarded as an extention to semantic nets; each node of a semantic net is represented by a frame. A frame is may be defined as a data structure that is used for representing a stereotyped situation.

Frames are slightly similar to the concept of class of object-oriented paradigm; class also contains attributes & methods. In frames, it consists of "attributes or slots"; slots are described with attribute-value pairs < slot_name, value >.

frame name
slot - filler
default values
constraints on values within the slots of frame
pointers (links) to other frames
ako (a-kind-of or subclass)
inst (instance)
instantiation procedure
inheritance procedure
default inference procedure
triggers

Table- Structure of a frame

The graphical representation of a frame n/w for the class 'hospital' described here

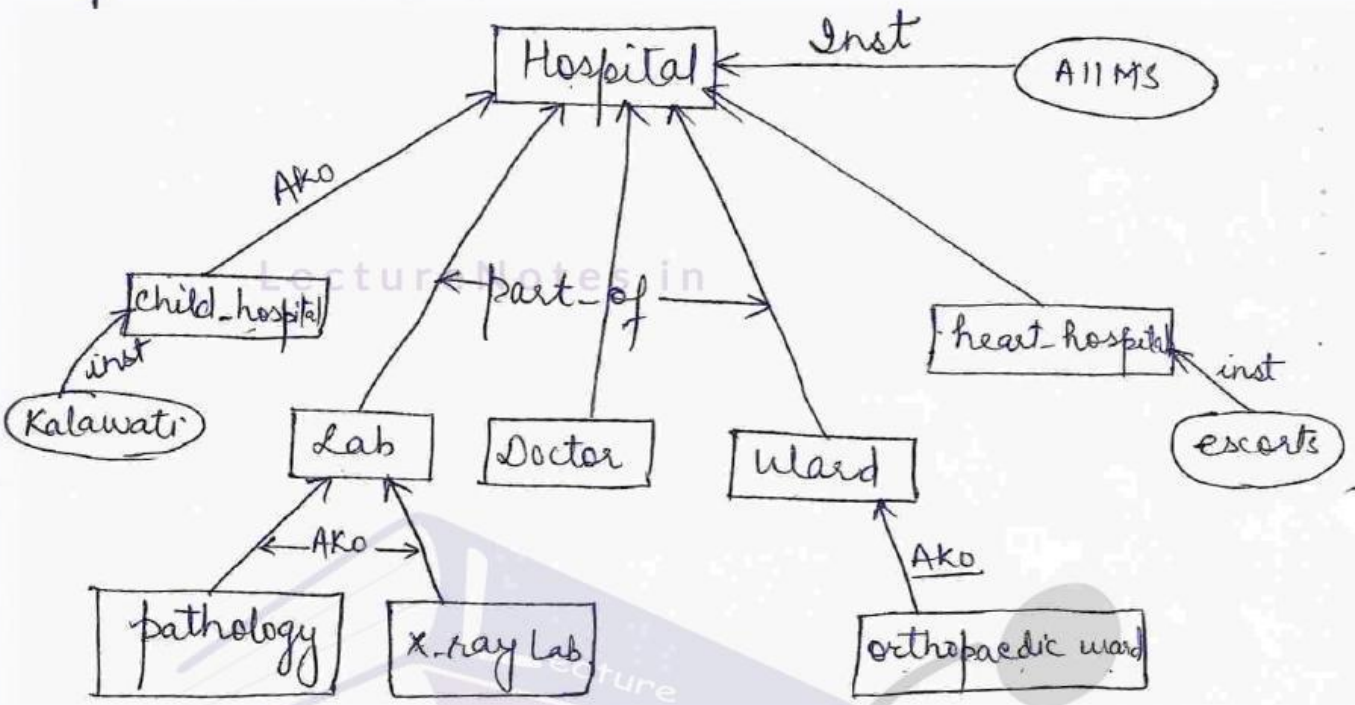


Fig. → A simple frame system.

6

1/3

Artificial Intelligence

Topic:

Production System

Contributed By:

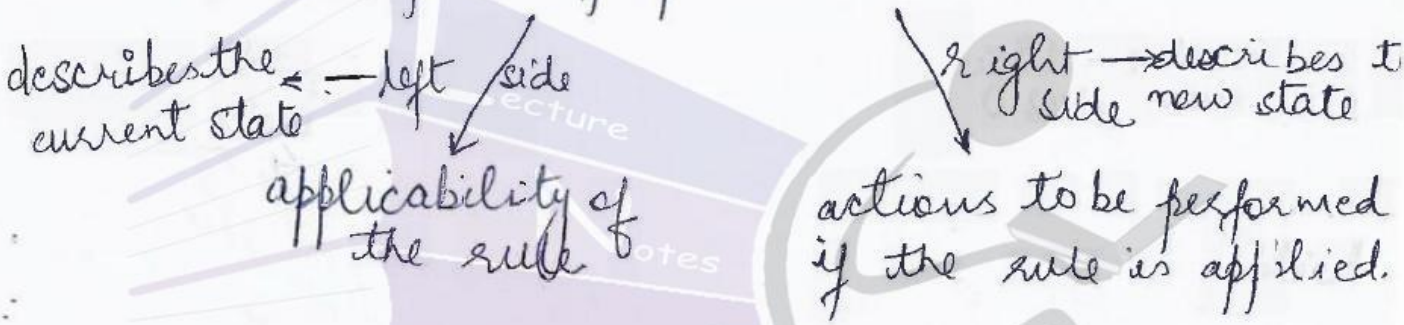
Dr. Sonal Sharma

Part II

General Problem Solving :→

1.] Production System ~→ PS is one of the formalisms that helps AI programs to search process more conveniently in state-space problems. This system comprises of start (initial state & goal (final) state of the problem along with one or more databases consisting of suitable & necessary info for the particular tasks.

PS consists of no. of production rules



PS also consists of "control strategies" that specify the sequence in which the rules are applied when several rule match at once.

Advantages of P.S ~→

(i.) It is a good way to model the strong state driven nature of intelligent action.

(ii.) New rules can be easily added to account for new situations without disturbing the rest of the system.

(iii.) It is quite important in real time AI environment & applications where new i/p to the database changes the behaviour of the system.

Water jug Problem ⟶ Given = 2 jugs

5 gallon, 3 gallon

Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 4 gallons of water into the 5-gallons jugs?

Soln ⟶ The set of ordered pairs of integers (x, y)

$x = 0, 1, 2, 3, 4 = 5$ ⟶ no. of gallons of water in 5 gallon jug.

$y = 0, 1, 2 = 3$ ⟶ quantity of water in the 3 gallon jug.

Start state = (0,0) Goal state = (4, n) {for any value of n, coz n = y gallon}

The possible op^n that can be used in this problem -

- ⟶ fill 5-g jug from the tap & empty the 5-g jug by throwing water down the drain
- ⟶ fill 3-g jug from the tap & empty the 3-g jug by throwing water down the drain.
- ⟶ Pour some or 3-g water from 5-g jug into the 3-g jug to make it full.
- ⟶ Pour some or full 3-g jug water into the 5-g jug.

These operations can formally be defined as produc rule

(2.16)

Rule No	Left of rule	Right of rule	Description
1	$(X,Y \mid X<5)$	$(5,Y)$	fill 5-g jug
2	$(X,Y \mid X>0)$	$(0,Y)$	Empty 5-g jug
3	$(X,Y \mid Y<3)$	$(X,3)$	fill 3-g jug
4	$(X,Y \mid Y>0)$	$(X,0)$	Empty 3-g jug
5	$(X,Y \mid X+Y\leq 5 \wedge Y>0)$	$(X+Y,0)$	Empty 3-g jug into 5-g jug
6	$(X,Y \mid X+Y\leq 3 \wedge X>0)$	$(0,X+Y)$	Empty 5-g jug into 3-g jug
7	$(X,Y \mid X+Y\geq 5 \wedge Y>0)$	$(5, Y-(5-X))$ Until 5-g jug is full	Pour water from 3-g jug into 5-g jug
8	$(X,Y \mid X+Y\geq 3 \wedge X>0)$	$(X-(3-Y),3)$	Pour water from 5-g jug into 3-g jug until 3-g jug is full.

Table — Production rules for Water Jug problem.

- It should be noted that there may be more tha one solution for a given problem.

Rule applied	5-g Jug	3-g Jug	Step No
Start state	0	0	0
1	5	0	1
8	2	3	2
4	2	0	3
6	0	2	4
1	5	2	5
8	4	3	6
Goal state	4	—	

Table — Solution Path - 1

Rule Applied	5-g jug	3-g jug	Step No
Start State	0	0	
3	0	3	1
5	3	0	2
3	3	3	3
7	5	1	4
2	0	1	5
5	1	0	6
3	1	3	7
5	4	0	8
Goal state	4	–	

Table – Solution path 2

We have shown 2 possible solution paths as given in Tables.

We notice that solution-1 requires 6 steps as compared to solution-2 that requires 8 steps.

In order to apply rules, we have to choose appropriate "control strategy."

Some More Problems ⟶

1. The Missionaries & Cannibals Problem ⟶
2. The Monkey & the Banana Problem
3. The Cryparithmic
4. The Eight-Puzzle Problem

↳ **Problem Statement** ⟶ The eight puzzle problem has a 3 x 3 grid with 8 randomly numbered (1 to 8) tiles arranged on it with one empty cell.

Start State

3	7	6
5	1	2
4	□	8

Goal State

5	3	6
7	□	2
4	1	8

Steps →

i. Start State : [[3,7,6], [5,1,2], [4,0,8]]
ii. Goal State could be represented as [[5,3,6][702][
iii. Operators can be thought of moving {Up, Down, Left, Right}, the direction in which blank space effectively moves.

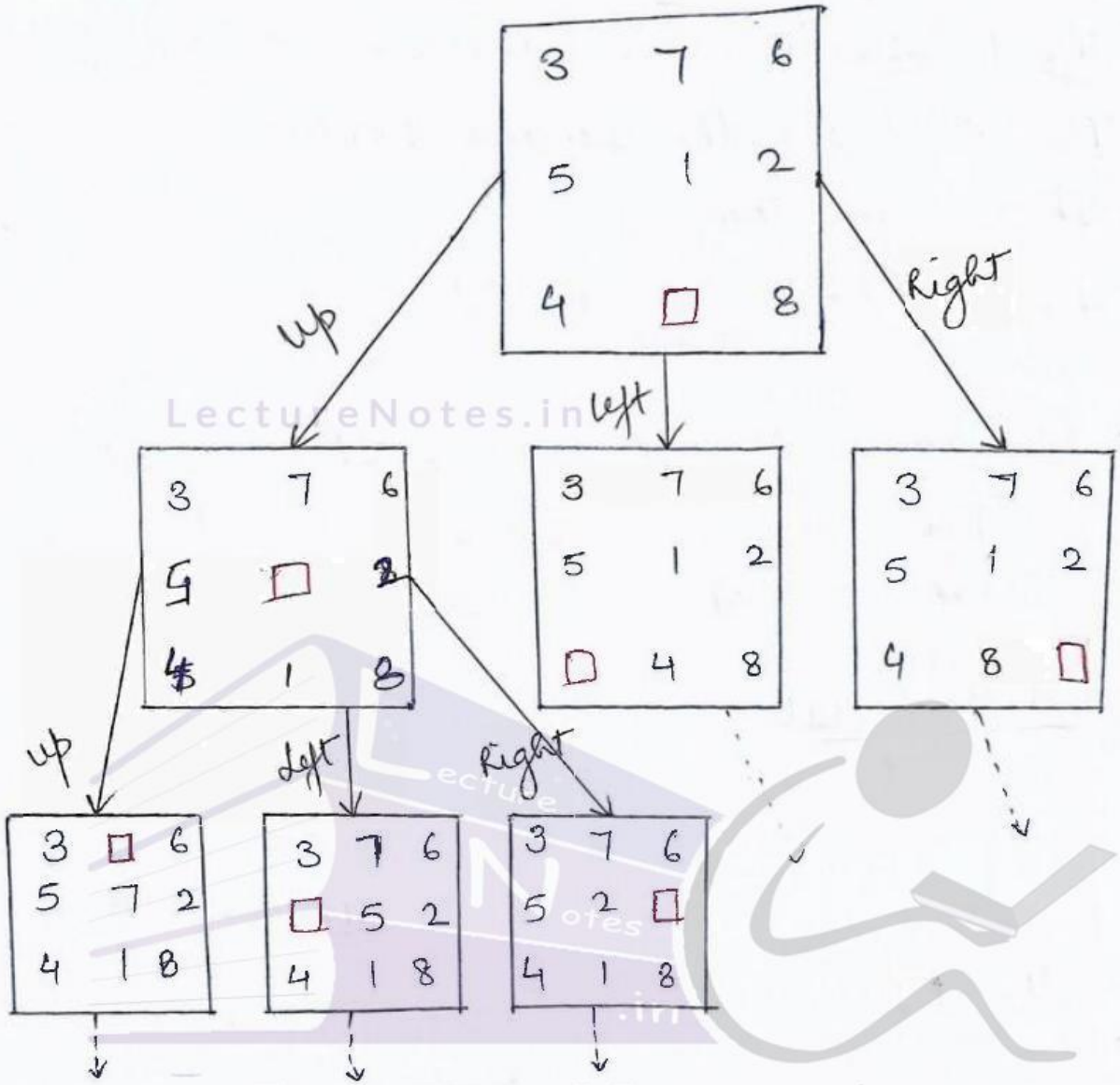


fig. → Partial search Tree for Eight Puzzle Problem

Continue searching like this till we reach the goal state.

Control strategies ⟶ This strategy focus on to decide which rule to apply next during the process of searching for a sol^n to a problem. Control strategy is one of the most important components of problem solving that describes the order of application of the rules to the current state.

* It causes motion towards a solution.

Eg. → Water jug problem (sol^n - 1 + 2)

The second requirement of control strategy is that it should explore the sol^n space in a systematic manner.

Eg. → Sometime repeated the same steps again & again for getting appropriate solution. This is because control strategy is not systematic.

Depth-first & Breadth-first are systematic control strategies but these are blind searches.

The problem can be solved by searching for a sol^n. The main work in the area of search strategies is to find the correct search strategy for a given problem. There are 2 directions in which such a search could proceed.

i) Data Driven Search, called forward chaining from the start state.

ii) Goal Driven Search, called backward chaining from the goal state.

i) Forward Chaining. ——→ The process of forward chaining begins with known facts & works towards a conclusion.

ii) Backward Chaining. ——→ It is goal-directed strategy that begins with the goal state & continues working backward; generating more sub goals that must also be satisfied to satisfy main goal until we reach to start state.

Eg. → Prolog (Programming in Logic) language uses this strategy.

We can use both data-driven and goal-directed strategies for problem solving, depending on the nature of the problem.

Characteristics of Problem. ——→ There are some key characteristics of searching & finding the solution for the problem.

1) Type of Problem ——→ There are three type of problems in real life i.e.

a) Ignorable ——→ There are the problems where we can ignore the sol^n steps.

Eg. → In proving a theorem, if some lemma is proved to prove a theorem & later we can realize that it is not useful, then we can ignore that sol^n step & prove another. Simple control strategy can be used

b) Recoverable ——→ These are the problems where sol^n steps can be undone. for eg. - Wa jug problem (empty & fill state)

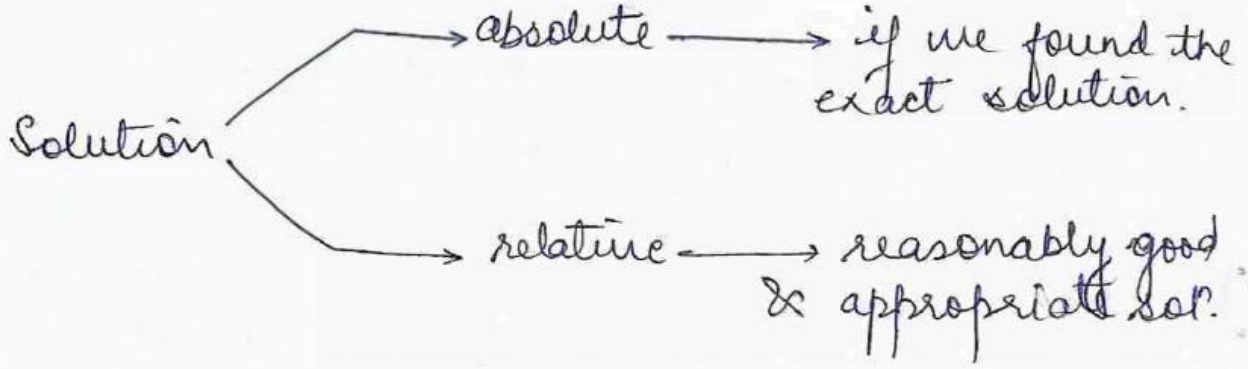
c) Irrecoverable ——→ The problems where sol^n steps can not be undone. for eg. - an two - player game such chess, playing card snake & ladder etc.

2) Decomposability of a Problem. ——→ Divid the problem into a set of independent smaller sub-problems, solve them and combine the solutions to get the final solution. 'Divide-and-conquer' technique is th commonly used method for solving such problem

3) Role of Knowledge ——→ Knowledge could be in the form of rules & facts which hel generating search space for finding the solution.

4) Consistency of Knowledge Base used in solving problem ——→ Make sure that knowledge ba used to solve problem is consistent. Inconsistent knowledge base will lead to wrong solutions.

5) Requirement of Solutions ——→ He should analyze the problem whether solution reqd. is absolute or relative.



Exhaustive Searches ⟶

1] Breadth First Search ⟶ The BFS expands all the states one step away from the start state, & then expands all states two steps from start state, then three steps etc., until a goal state is reached.

This search is implemented using two lists called OPEN & CLOSED. The OPEN list contains those states that are to be expanded & CLOSE lists keeps track of states already expanded.

OPEN ⟶ Queue
CLOSED ⟶ STACK

Algorithm (BFS) ⟶

Input :— START & GOAL STATES
Local Variables :— OPEN, CLOSED, STATE-X, SUCCS, FOU
Output :— Yes or No
Method :—

- Initialize OPEN list with START & CLOSED = ϕ;
- FOUND = false;
- while (OPEN $\neq \phi$ & FOUND = false) do
  {
  - remove the first state from OPEN & call it STATE-X;
  - put STATE-X in the front of CLOSED list {maintained as STACK};
  - if STATE-X = GOAL, then FOUND = TRUE else
    {
    - perform EXPAND operation on STATE-X producing a list of SUCCS;

• remove from successors those states, if any, that are in the CLOSED list;
• append succs at the end of the OPEN list; /* give
}
} /* end while */
• if found = true then return Yes else return No
Stop.

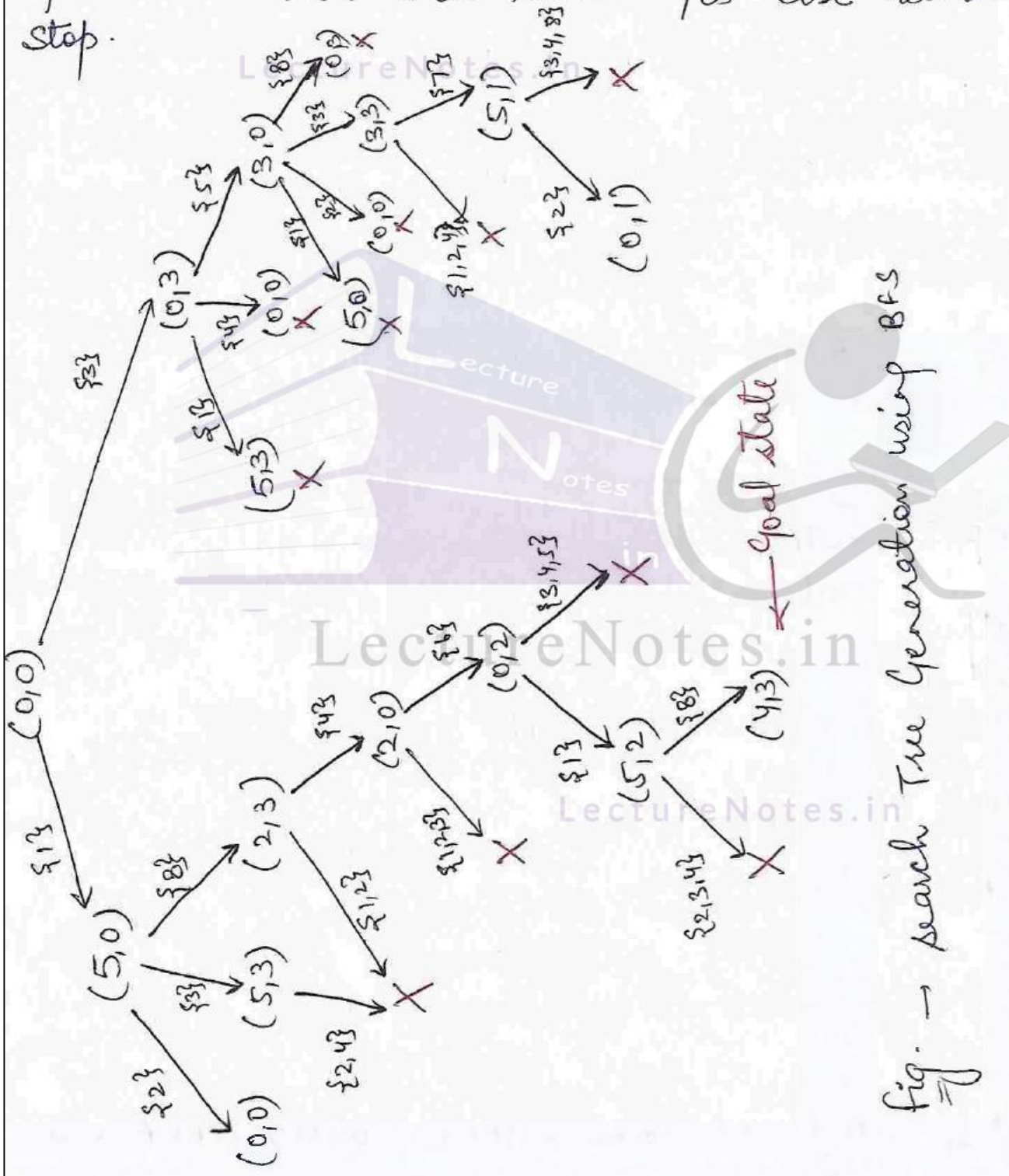


fig. → search Tree Generation using BFS

Solution path :-

(0,0) → (5,0) → (2,3) → (2,0) → (0,2) → (5,2) →

Search tree is developed level wise. This is not memory efficient as partially developed tree is to be kept in the memory but it finds optimal sol or path.

2.] Depth first Search ⇝ In DFS, we go as far down as possible into the sear tree / graph before backing up & trying alternati DFS is memory efficient, as it only stores a sing path from the root to leaf node along with th remaining unexpanded siblings for each node on the path.

We can implements DFS by using two lists called OPEN & CLOSED.

OPEN → It contains those states that are to be expanded, maintained as stack, first element is Goal State, then search terminates successfully.

CLOSED → It keeps track of states already ex- panded, maintained as stack, the imp can be obtained by modifying CLOSED list, by putting pointer back to its parent in the search tree.

Algorithm ——

Input :- START & GOAL states of the problem

Local Variables :- OPEN, CLOSED, RECORD_X, SUCCESSORS, FOUND

Output :- A path sequence from START to GOAL state. if one exists otherwise return NO.

Method :-

- initialize OPEN list with (START, nil) & set CLOSED = ϕ;
- FOUND = false;
- while (OPEN $\neq \phi$ and FOUND = false) do
  {
  - remove the first record (initially (START, nil)) from OPEN list & call it RECORD-X;
  - put RECORD-X in the front of CLOSED list (maintained as STACK);
  - if (STATE_X of RECORD_X = GOAL) then FOUND = true
    else
    {
    - perform EXPAND operation on STATE_X producing a list of records called SUCCESSORS; create each record by associating parent link with its state;
    - remove from successors any record that is already in the CLOSED list;
    - insert successors in the front of the OPEN list /* STACK */

    }

  } /* end of while */

- if FOUND = true then return the path by tracing through the pointers to the parents on the CLOSED list
- else return NO.

Water Jug Problem

Search tree generation using DFS	OPEN List	CLOSED List
Start state (0,0)	[((0,0), nil)]	
(5,0)	[((5,0), (0,0))]	[((0,0), nil)]
(5,3)	[((5,3), (5,0))]	[((5,0),(0,0)), ((0,0), nil)]
(0,3)	[((0,3), (5,3))]	[((5,3),(5,0)), ((5,0),(0,0)), ((0,0), nil)] .
(3,0)	[((3,0), (0,3))]	[((0,3), (5,3)), ((5,3),(5,0)), ((5,0),(0,0)), ((0,0), nil)].
(3,3)	[((3,3), (3,0))]	[((3,3), (3,0)), ((0,3), (5,3)), ((5,3),(5,0)), ((5,0),(0,0)), ((0,0), nil)].
(5,1)	⋮	⋮
(0,1)		
(1,0)		
(1,3)		[((4,0),(1,3)), ((1,3),(1,0)), ((1,0),(0,1)), ((0,1),(5,1)), ((5,1),(3,3)), ((3,3),(3,0)), ((3,0),(0,3)), ((0,3),(5,3)), ((5,3),(5,0)), ((5,0),(0,0)), ((0,0), nil)].
(4,0)		
Goal state	[((4,0),(1,3))]	

Fig. → Search Tree Generation using DFS

The path is obtained from the list stored in CLOSED. The sol" path is –

(0,0) → (5,0) → (5,3) → (0,3) → (3,0) → (3,3)
→ (5,1) → (0,1) → (1,0) → (1,3) → (4,0)

Comparisons ⟶ Since these are unguided, blind and exhaustive searches, we cannot say much about them but can make some observations.

- BFS is effective when the search tree has a low branching factor.
- BFS can work even in trees that are infinitely deep.
- BFS requires a lot of memory as no. of nodes in level of the tree increases exponentially.
- BFS is superior when the GOAL exists in the upper right portion of a search tree.
- BFS gives an optimal solution.
- DFS is effective when there are few subtrees in the search tree that have only one connection point to the rest of the states.
- DFS is best when the GOAL exists in the lower left portion of the search tree.
- DFS can be dangerous when the path closer to the START & farther from the GOAL has been chosen.
- DFS is memory efficient as the path from START to current node is stored. Each node should contain states & its parent.
- DFS may not give optimal solution.

Generate and Test ⟶ The generate & test strategy is the simplest of all the approaches we discuss.

Algorithm ⟶

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

The generate and test algm is a depth-first search procedure since complete solⁿ must be generated before they can be tested.

Generate and test, of course, also operate by generating solutions randomly, but then there is no guarantee that solⁿ will ever be found.

The most straight forward way to use generate and test is as a depth first search tree with backtracking. If some intermediate states are likely to appear often in the tree, it may be better to modify that procedures, as described above (algm) to traverse a graph rather than tree.

Hill Climbing ⟶ Quality Measurement turn Depth-first search into Hill Climbing (variation of generate and test strategy). It is an optimization technique that belongs to the family of local searches.

Hill Climbing can be used to solve problems that have many solutions but where some solutions are better than others. If there is some way of ordering the choices so that the most promising node is explored first, then search efficiency may be improved.

Algorithm ⟶ (Simple Hill Climbing)

Input :- START & GOAL states

Local Variables :- OPEN, NODE, SUCCs, FOUND;

Output :- Yes or No

Method :-

- store initially the start node in a OPEN list (maintained as stack);
- FOUND = false;
- while (OPEN ≠ empty and found = false) do
  {
  - remove the top element from OPEN list & call it NODE;
  - If NODE is the goal node, then FOUND = true else
    - find SUCCs of NODE, if any;
    - sort succs by estimated cost from NODE to goal state and add them to the front of OPEN list;

  } /* end while */
- if FOUND = true then return Yes otherwise return No.

Problems with Hill Climbing. ⟶ There are few problems with Hill Climbing. The search process may reach to a position that is not a solution but from there no move improves the situations.

Local maximum ⟶ It is a state that is better than all its neighbours but not better than some other states which are far away. From this state all moves looks to be worse.

Plateau ⟶ It is a flat area of the search space where all neighbouring states has the same value. It is not possible to determine the best direction.

Ridge ⟶ It is an area of search space that is higher than surrounding areas but that cannot be traversed by single moves in any one direction. It is a special kind of local maxima.

Best-First Search ⟶ Best-first search is based on expanding the best partial path from current node to goal node. Here forward motion is from the best open node so far in the partially developed tree.

If the state has been generated earlier & new path is better than the previous one, then change the parent & update the cost.

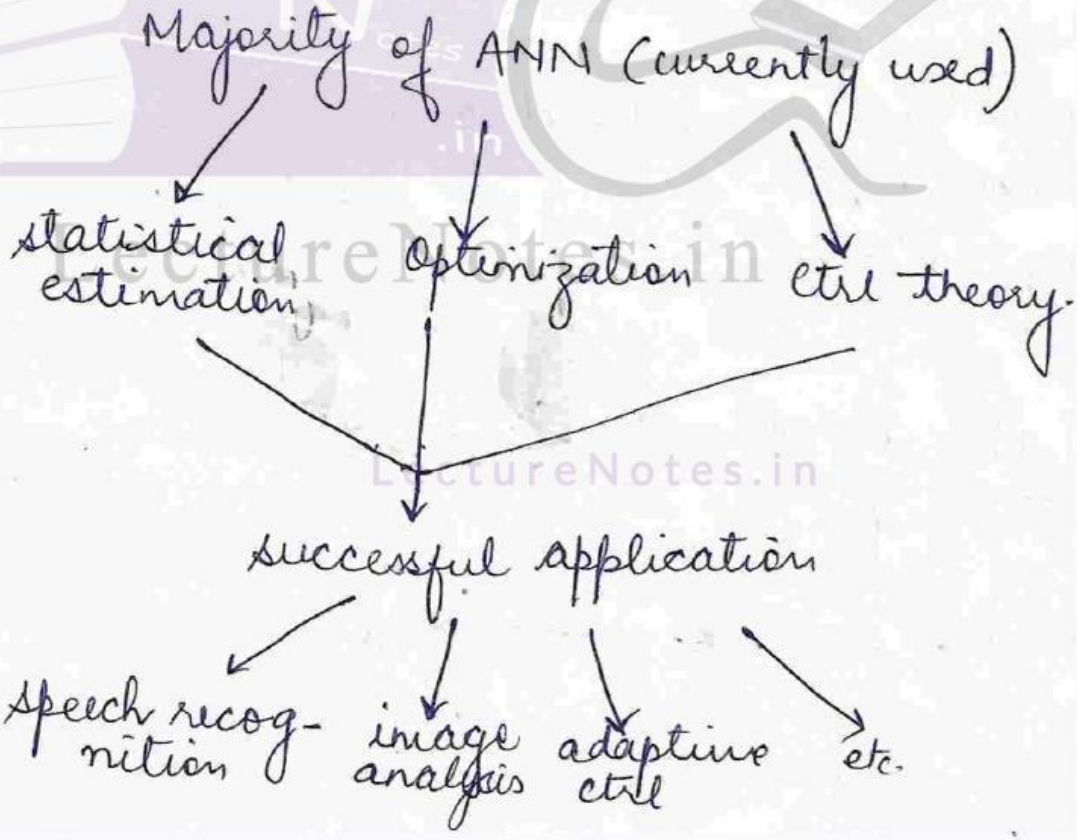
In Hill climbing, sorting is done on the successors nodes, whereas in the best-first search, sorting is done on the entire list. It is not guaranteed to find an optimal solution, but generally it finds some solution faster than solution obtained from any other method.

Unit #3.

Artificial Neural Networks

The term neural n/w is generally used to descri a n/w of biological neurons that are functional connected to the central nervous system of mos living organisms. An artificial neural n/w (AN is composed of artificial nodes or neurons that are connected together to form a n/w.

ANN can thus be considered to be non-line statistical data-modeling or decision-maki tools.



Artificial Intelligence

Topic:

Artificial Neural Network

Contributed By:

Dr. Sonal Sharma

These may also be used to :-

i) gain an understanding of biological neural n/ws.

ii) solve AI problems without having to create models of real biological system.

iii) model complex relationships b/w i/p & o/p

iv) find patterns in given data

Artificial Neural Networks (ANN) ⟶

An ANN is defined as a n/w consisting of a large no. of neurons connected to each other with some weights & is also referred to as a connectists model.

The neurons are trained by using knowledge of the domain which is given in the form of training examples. Any ANN may be specified by the following components :-

i) Neuron Model ⟶ The inf^n processing unit of ANN.

ii) Architecture ⟶ A set of neurons & links connecting neurons, where each link possesses a weight.

iii) A learning algorithm ⟶ for training ANN by modifying the weights in order to model a particular learning task correctly on the training ex.

The Neuron Model ⟶ According to this model, a neuron consists of the following —

- Inputs
- Weights
- Adder
- Activation function

It consists of set of links describing

X_i = Neuron Inputs

w_i = weights $(w_1, w_2 \ldots w_m)$

Σ = Adder function

U = weighted sum

$$U = \sum_{i=1}^{m} X_i * w_i$$ where m = the no. of i/p.

It requires an activation function 'ψ' for limiting the amplitude of the neuron output (Y).

$$Y = \psi(U + b) = \phi(V)$$ where $\{V = U + b\}$

b = bias

V = induced field of neurons.

$$V = \sum_{i=0}^{m} w_i * X_i$$ where $w_0 = b$ & $X_0 = 1$

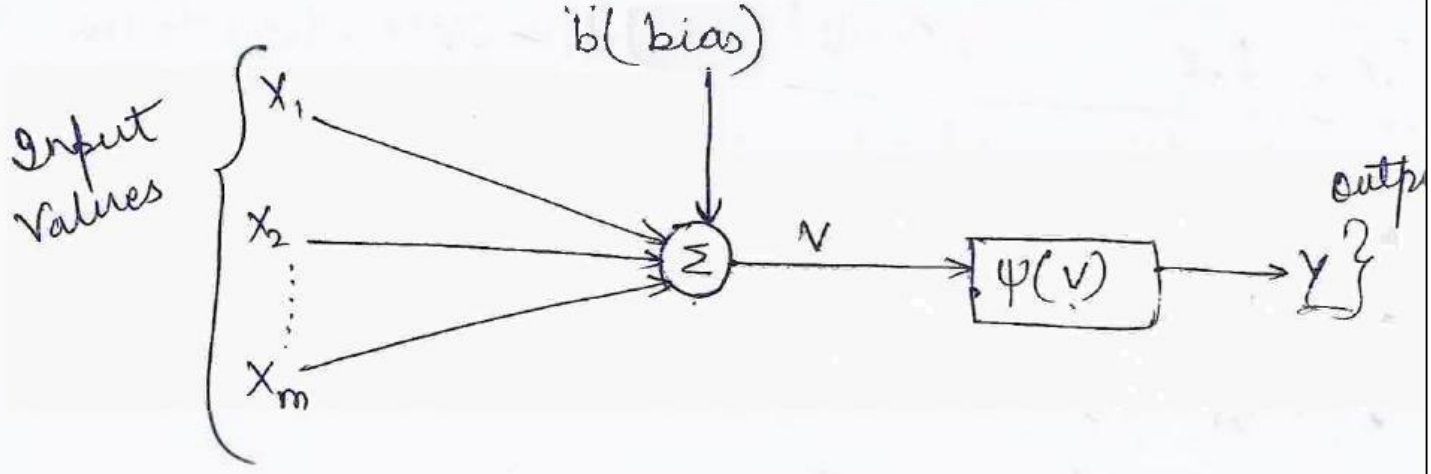


fig → Neuron Diagram

Activation function ——→ Some of the commonly used activation function are as follow

i) Step function → {also known as sign function} {used for binary classification}
ii) Ramp function
iii) Sigmoid function ——→ when i/p data is continuous then these three functions may be applied as activation function for classification & regression problems.
iv) Gaussian function

i) **Step function** ~~~→

$$\varphi(v) = \begin{cases} a, & \text{if } v < c \\ b, & \text{if } v \geq c \end{cases}$$

centre

ii) Ramp function ⟶

$$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > d \\ a + ((v-c) * (b-a)/(d-c)) & \text{otherwise} \end{cases}$$

iii) Sigmoid function with Z, X, Y parameters

$$\varphi(v) = Z + \frac{1}{\exp(-x * v + y)}$$

iv) Gaussian function

$$\varphi(v) = \exp\left\{-\frac{(v-c)^2}{\sigma^2}\right\}$$

C = centre

σ = spread or radius

Neural Network Architectures ⟶ There are three different classes of n/w architecture which are listed as follows:—

a) Single-Layer feed-forward n/w.
b) Multi-Layer feed-forward n/w
c) Recurrent Work.

a) Single-Layer feed-forward Network →

They were the first & the simplest of ANNs to be developed. It is defined as a n/w where connections b/w various nodes do not form a directed cycle such that the inf^n in this n/w moves only in the forward direction, that is, from i/p nodes to o/p nodes, through a series of weights.

There are no hidden layers, contains only i/p and o/p layers. Each i/p node is connected to each o/p node with a link having weight.

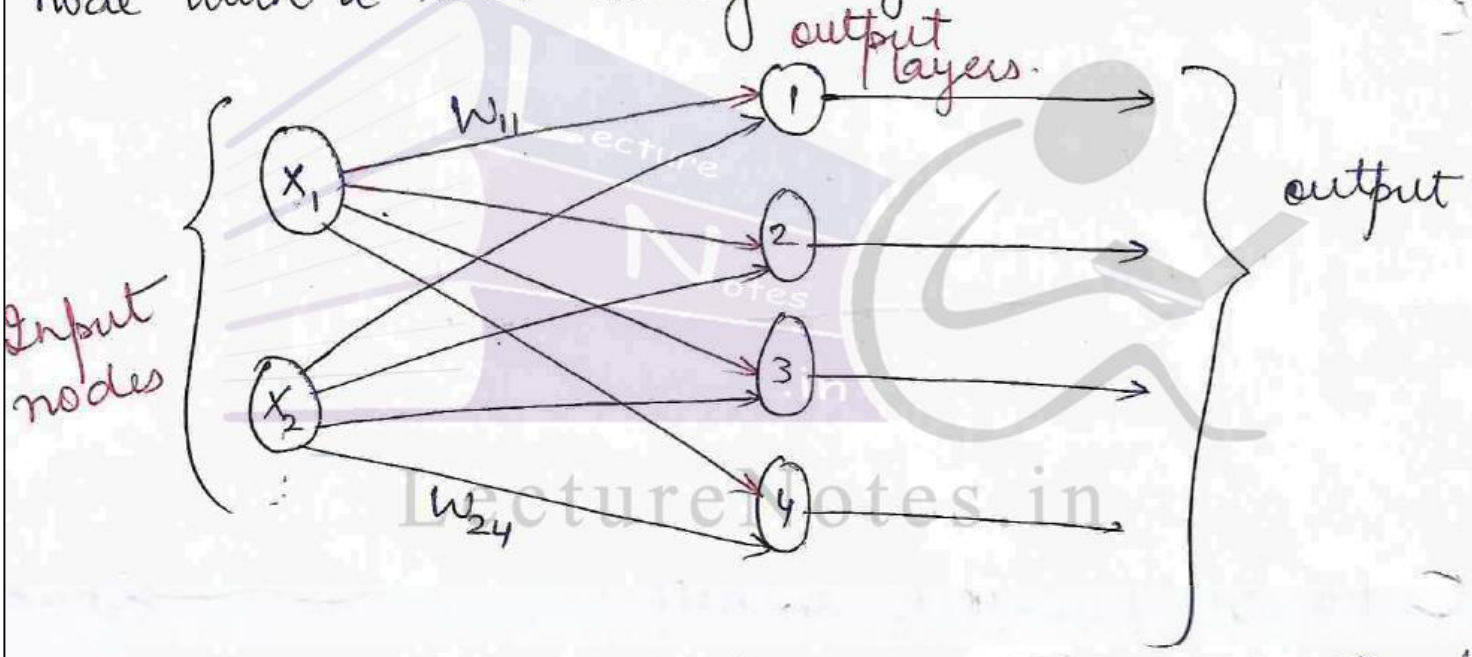


Fig. → Single layer feed forward Neural Network

I/p Nodes = $\{X_1, X_2\}$

O/p Nodes = $\{W_{ij}$; where $i = 1, 2$
$j = 1, 2, 3, 4\}$

In the following subsection, we proceed to discus a special case of this type of n/w.

i) Perceptron: Neuron Model ⟶ It is a special form of SLFFN & was first pro ed by 'Rosenblatt' in 1958.

It is a simple neuron model primarily for "bina classification", i.e. it classifies the i/p into o of two given categories. It has only one output mode.

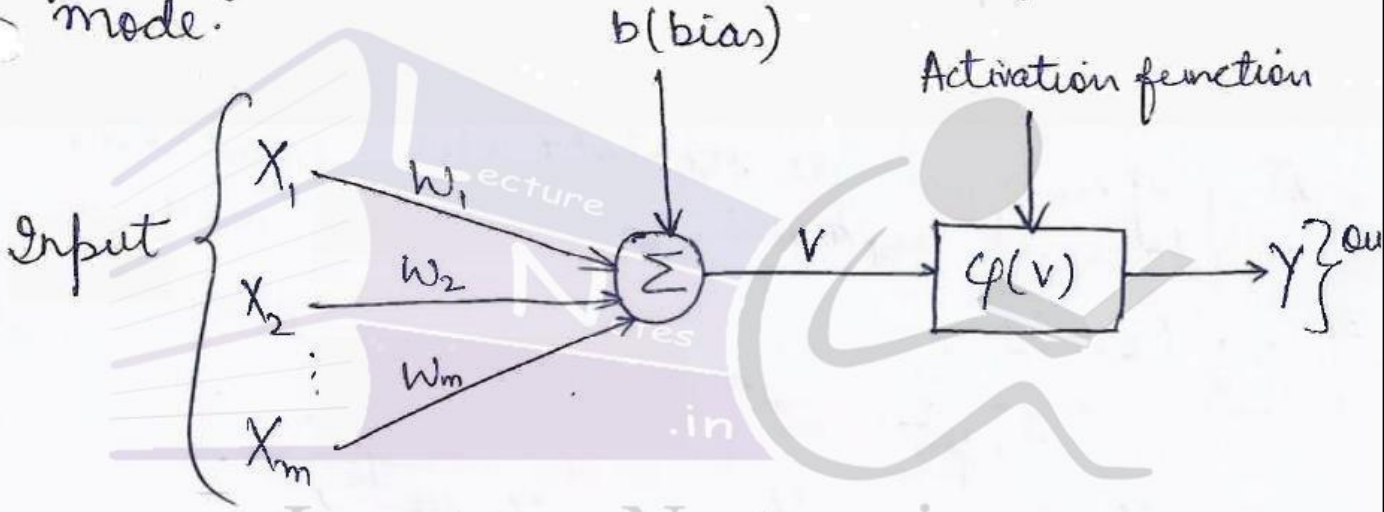


fig. → Perceptron for Binary Classification

$\varphi(V)$ ⟶ used as a step function instead o activation function

$$\varphi(V) = \begin{cases} -1 & \text{if } V < \theta \\ +1 & \text{if } V \geq \theta \end{cases}$$

step function

θ = defined threshold

V = sum of i/ values

Before a perceptron can be used for binary classification, it needs to be trained for this purpose.

Perceptron can be is a binary classifier, it can only model linearly separable classes. In cases, where the classes are not linearly separable, it is desired to find a linear separator that minimizes the mean squared error.

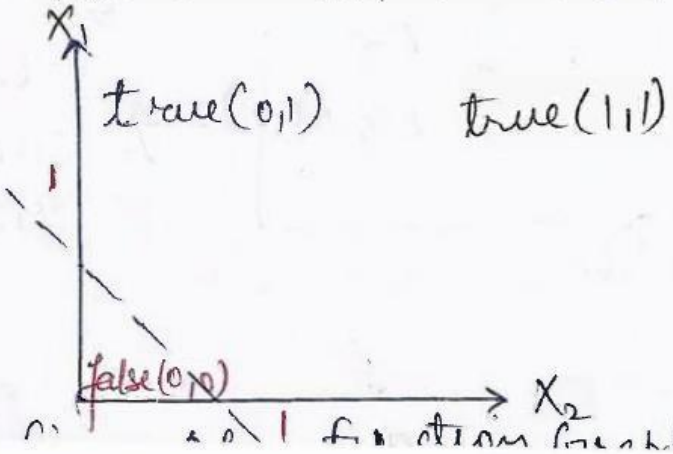
The Boolean functions AND, OR, Complement are linearly separable, it function, whereas XOR is non-linearly separable function

When linearly separable functions are plotted on a 2-D graph, a single straight line can be drawn such that it separates the values in two parts

Eg. → Consider OR function

Input		Output
x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

Table → OR function



Learning Algorithm for Perceptron

- Initially random weights are assigned to i/p variat in the range −0.5 and +0.5.
- Training data is presented to perceptron & its o/p is observed.
- If o/p is incorrect, the weights are adjusted accordingly using the following formula:-

$$W_i = W_i + (A * X_i * E)$$

where E → error produced

A (0 < A < 1) → learning rate.

- The learning rate A may be defined as follow
  - * If o/p is correct, then A = 0
  - * If o/p is too low, then A = some +ive no
  - * If o/p is too high, then A = some −ive no.
- After modification of weights, the next trainir data is applied with the modified weights & the weights are further adjusted accordingly.
- Once all the training data have been applie the process starts again until all the weights a correct & all errors are zero.
- Each iteration of this process is known as an epoch.

Multi-Layer feed-forward Networks →

Multi-layer feed forward neural networks (FFN have to be used to handle data that is not line separable or not separable by hyper-plane. These contain hidden layers b/w I/P & O/P layers, which do not directly receive i/p & or send o/p to the external environments.

The architecture of FFNN is shown in fig. -

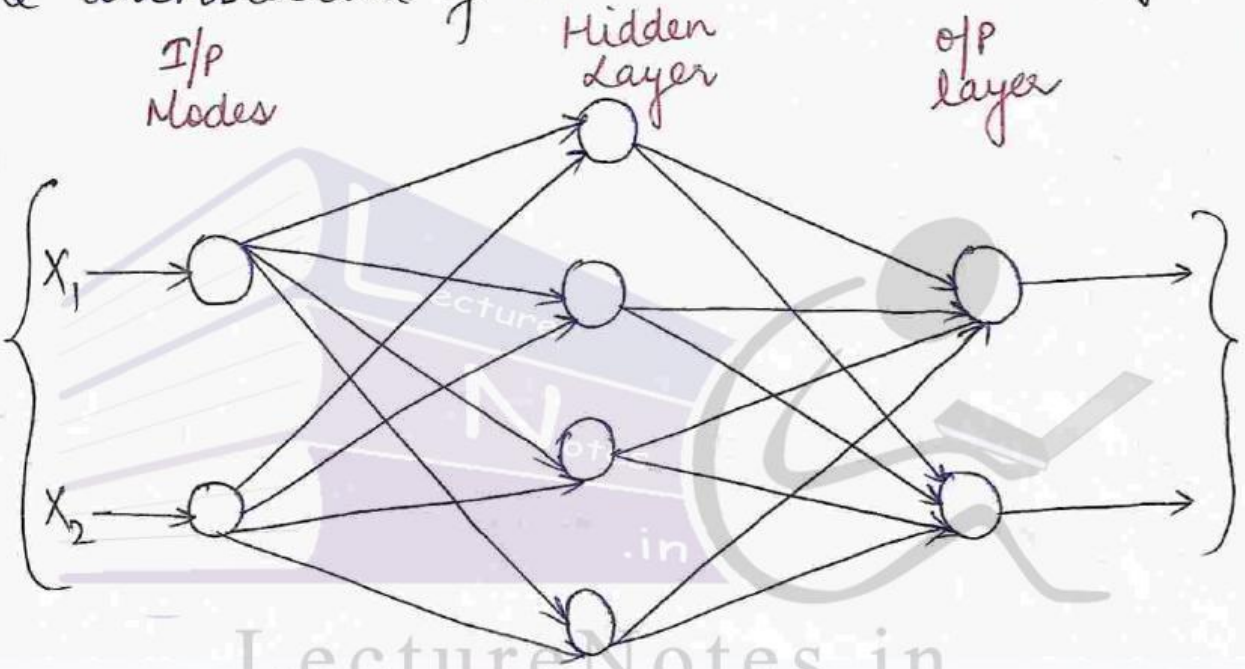


Fig. → 2-4-2 feed-forward Neural Network

Multi-layer feed forward n/w continuously evolve with the help of learning techniques. One of the mo crucial learning technique is known as back-propag tion method. This Technique works in the followi manner :-

* The O/p values obtained for a known set of i/p is compared with the correct answer in order to determine the value of some pre-defined error function.
* This error is then fed back into the n/w using back-propogation method (explained in detail later).
* The algorithm utilizes this inf^{n} & then adjusts the weights of all connections so that the corresponding error function is reduced to some extent.
* This procedure is repeated for large no. of training cycles, so that the n/w obtains a state where the error becomes negligible. We can say that the n/w has learned the target function.

A general method, called 'gradient descent', is applied in case of non-linear classification for proper adjustments of weights.

In this method, the derivative of the error function with respect to the n/w weights is determined, & the weights are then changed accordingly to decrease the margin of the error.

$$\frac{d}{dx} f(x)_{x \to \text{n/w weight}}$$

$f(x)$ = error function w.r.to n/w weight

A typical eg. of non-linearly separable function is the XOR function, which takes 2 i/p arguements with values in set $\{0,1\}$. & returns one o/p in set $\{0,1\}$

Input		Output
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Table. – XOR function.

XOR function computes logical exclusive OR

o/p will true if both I/p have different truth valu

0, 1 → truth values

false true

The graph of the XOR function on the basis of tabl values

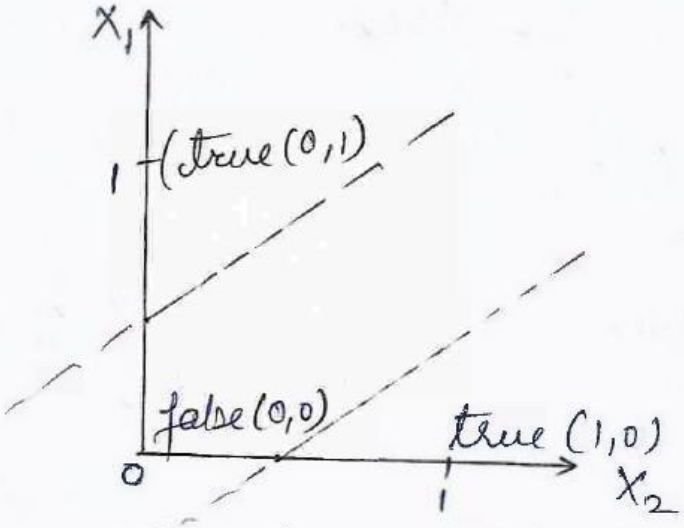


Fig. Graph of XOR function.

from the graph we can see that no single line can be drawen to separate false & true into two distinct classes. Therefore the function is said to be non-separable.

for such problems, FFNN is used with 2 hidden nodes, & it makes use of the sign activation funct (step function)

The activation function in this case is defined as follows :—

$$\varphi(V) = \begin{cases} 0, & \text{if } V \leq 0 \\ 1, & \text{if } V > 0 \end{cases}$$

Eg. I/P Nodes — Hidden layer — o/p layer — o/p.

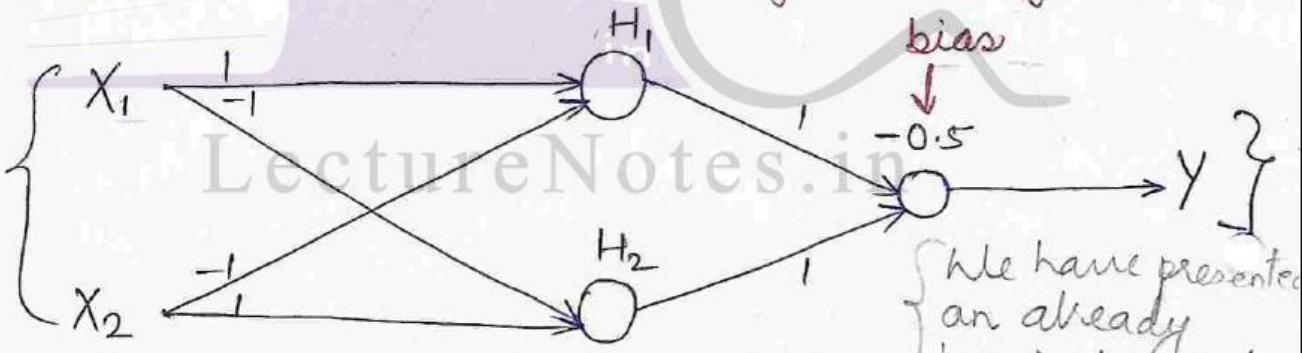


fig. → FFNN for XOR.

⁂ We are representing 2 states = 0 (false) & 1 (true)

So,

$-1 = 0$
$-0.5 = 0$
} Hidden Nodes

$1 = 1$
$+0.5 = 1$
} Hidden Nodes.

Inputs		Output of Hidden Nodes		O/P Node	XOR X_2
X_1	X_2	H_1	H_2	↑ answer	
0	0	0	0	$-0.5 \to 0$	0
0	1	$-1 \to 0$	1	$0.5 \to 1$	1
1	0	1	$-1 \to 0$	$0.5 \to 1$	1
1	1	0	0	$-0.5 \to 0$	0

Table → Working of an ffNN for XOR.

Back-Propagation Training Algorithm for ffN

Most commonly used training algorithm f ffNN is based on the gradient descent method. A typical activation function used in this case is the sigmoid function, which can be considered to a continuous approximation of the step function.

$$\varphi(V_i) = \frac{1}{1 + e^{(-A * V_i)}}$$

where $A > 0$, $V_i = \sum w_{ij} * y_j$

w_{ij} → weight of the link from node i to node j

y_j → output of node j.

Back Propogation Algorithm ⟶ Using a se of training examples (also called training set it determines those weights for which the total erro of the n/w is minimum. Back propogation consists of the repeated application of the following two passe

a.) Forward Pass ⟶ In this step, the n/w is act vated for one example & the error of each neuron at the o/p layer is compared.

b.) Backward Pass ⟶ In this step, the error is propogated backwards through the n/w layer by layer, by recursively computing the local gradient of each neuron.

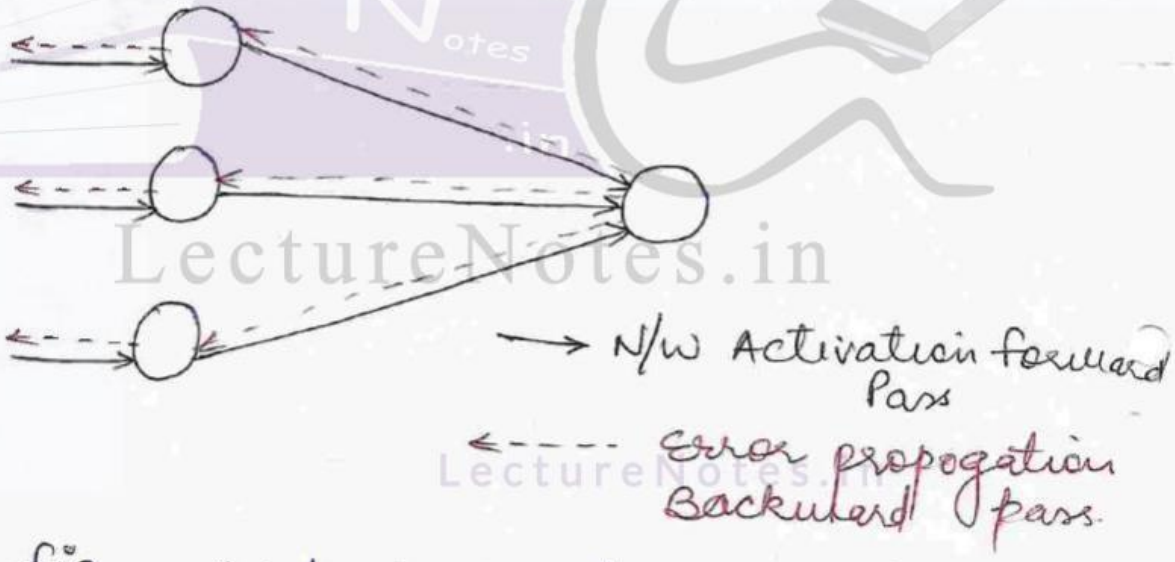


fig. → Back propogation algorithm.

Most training algorithms follow the following cycle to refine the weight values :-

i.) Run a set of i/p variables values through the n/w using a random set of weights.

ii.) Compute the difference b/w the computed & the actual target values for this case.

iii.) Average the error over the entire set of training cases.

iv.) Propogate the error backward through the and compute the gradient (vector of derivatives of the change in error with respect to chan in weight values.

v.) Make adjustments to the weights to reduce the er

vi.) Each cycle is called an epoch.

The error inf^n is propogated backward through the n/w, this type of training method is calle backward propogation.

Eg. → N/w with 3 layers.

i → nodes in the input layer

j → nodes in the hidden layer

k → nodes in the output layer.

W_{ij} → Weight of the connection b/w i & j

Y_j → Output value of node j.

$$Y_j = \frac{1}{1+e^{-X_i}}, \text{ where } X_i = \sum_{i=1}^{m} X_i * W_{ij} - \theta_j$$

θ_j → Threshold for node j

n → no. of i/p to node j.

Algorithmic Steps for Training ⟶

FFNN Training Algorithm

a) Initialize I = 1 ; W(I) randomly ;
b) while (stopping criterion is not satisfied or I < max_iterations)
{
c) for each example (X, D)
{
d) run the n/w with input X and compute the output Y;
e) update the weights in backward order starting from those of the o/p layer computed using the (generalized) delta rule explained below;
};
f) I = N+1;
g) end-while;

Weight Update Rule ⟶ The weight update in back-propogation method using gradient descent method. The values of weights are adjusted by an amount proportional to the first derivative (the gradient) of the error between the actual o/p & the computed o/p. The weight w_{ji} is updated by the following formula –

$$\boxed{W_{ji} = W_{ji} + \Delta W_{ji}}, \text{ where } \Delta W_{ji} = -\eta * [\delta E / \delta W_{ji}],$$

η = learning rate in the range [0.1,0.

Stopping Criterions ⟶ There are many stopping criteria used in training algorithm for FFNN. Some of the widely used criteria are given as follow

i) **Total mean squared error change** ⟶ According to this criterion, back propogation is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small, say in the range [0.1, 0.01].

ii) **Generalization based criterion** ⟶ According this criterion, after each epoch the FFNN is tested for generalization. If the generalized performance is adequate then the process is stopped.

Delta Rule (Least Mean Square) for Error Minimization ⟶ The delta rule is based on the concept of continuous adjustment of weights in such a manner that the difference of error (denoted by delta) between the actual & computed o/p is reduced.

The aim of this method is to minimize the mean squared error & thus it is also known as the "least mean square method".

The procedure is used in this method is as follows-

The training example —

X_i = a pair of i/p (as $(X(i), D(i))$

D = desired o/p (for $1 \le i \le$

* Compute the error of o/p neuron k after the activat of the n/w on the i-th training eg.

$$(X(i),\ D(i)) \text{ as } E_k(i) = D_k(i) - Y_k(i)$$

$Y_k(i)$ = o/p of the k^{th} neuron.

* The n/w error is defined as the sum of the square errors of the o/p neurons & is calculated as -

$$E(i) = \sum \left(E_k(i)\right)^2, \quad \text{for } k^{th} \text{ o/p. node.}$$

* The total mean squared error is defined as the average of the n/w errors of the training examples & is given by

$$E_{av} = \sum_{i=1}^{N} E(i)/N$$

* Minimize # E_{av} error.