MCA 203 Artificial Intelligence and Machine Learning

Madhya Pradesh Bhoj (Open) University

(Established Under an Act of State Assembly in 1991)

मध्य प्रदेश भोज (मुक्त) विश्वविद्यालय

Contents

List of Figures

List of Tables

1. Artificial Intelligence

1.1. What is Artificial Intelligence (AI)?

Artificial Intelligence is an interdisciplinary field aiming at developing techniques and tools for solving problems that people at good at.Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think.AI is accomplished by studying how the human brain thinks and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

According to the father of Artificial Intelligence, John McCarthy, it is “The science and engineering of making intelligent machines, especially intelligent computer programs”. Some other classical

- AI is a branch of Computer Science that pursues creating computers or machines as intelligent.
- It is the science and engineering of making intelligent machines, especially intelligent computer programs.
- Artificial Intelligence is the study of how to make computers do things which, at the moment, people do better.

It has gained prominence recently due, in part, to big data, or the increase in speed, size and variety of data businesses are now collecting. AI can perform tasks such as identifying patterns in the data more efficiently than humans, enabling businesses to gain more insight out of their data. From a business perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.

From a programming perspective, AI includes the study of symbolic programming, problem solving, and search.

1.2. Types of Artificial Intelligence

- Artificial Narrow Intelligence (“ANI”): ability to carry out a specific task (play chess; get information based on voice directions (SIRI or Alexa); spot spam email; driverless cars).
- Artificial General Intelligence (“AGI”): ability to carry out different tasks that a human could do.
- Artificial Super Intelligence(“ASI”): ability to learn from its experiences and from new data to perform a wide range of actions and to generate new computer code on its own to help achieve its objectives.

1.2.1. AI Vocabulary

Intelligence relates to tasks involving higher mental processes, e.g. creativity, solving problems, pattern recognition, classification, learning, induction, deduction, building analogies, optimization, language processing, knowledge and many more. Intelligence is the computational part of the ability to achieve goals.

Intelligent behaviour is depicted by perceiving one’s environment, acting in complex environments, learning and understanding from experience, reasoning to solve problems and discover hidden knowledge, applying knowledge successfully in new situations, thinking abstractly, using analogies, communicating with others and more.

Science based goals of AI pertain to developing concepts, mechanisms and understanding biological intelligent behaviour. The emphasis is on understanding intelligent behaviour.

Engineering based goals of AI relate to developing concepts, theory and practice of building intelligent machines. The emphasis is on system building.

AI Techniques depict how we represent, manipulate and reason with knowledge in order to solve problems. Knowledge is a collection of ‘facts’. To manipulate these facts by a program, a suitable representation is required. A good representation facilitates problem solving.

Learning means that programs learn from what facts or behaviour can represent. Learning denotes changes in the systems that are adaptive in other words, it enables the system to do the same task(s) more efficiently next time

Applications of AI refers to problem solving, search and control strategies, speech recognition, natural language understanding, computer vision, expert systems, etc.

1.2.2.Problems of AI:

Intelligence does not imply perfect understanding; every intelligent being has limited perception, memory and computation. Many points on the spectrum of intelligence versus cost are viable, from insects to humans. AI seeks to understand the computations required from intelligent behaviour and to produce computer systems that exhibit intelligence. Aspects of intelligence studied by AI include perception, communicational using human languages, reasoning, planning, learning and memory.

1.2.3.Applications of AI :

AI has applications in all fields of human study, such as finance and economics, environmental engineering, chemistry, computer science, and so on. Some of the applications of AI are listed below:

- Preception
    - Machine Vision
    - Speech understanding
    - Touch sensation
- Robotics
- Natural Language Processing
    - Natural Language understanding
    - Speech Understanding
    - Language Generation
    - Machine Translation
- Planning
- Expert Systems
- Machine Learning
- Theorem Proving
- Symbolic Mathematics
- Game Playing

1.2.4.AI Technique:

Artificial Intelligence research during the last three decades has concluded that Intelligence requires knowledge. To compensate overwhelming quality, knowledge possesses less desirable properties.

A. It is huge.

B. It is difficult to characterize correctly.

C. It is constantly varying.

D. It differs from data by being organized in a way that corresponds to its application.

E. It is complicated.

An AI technique is a method that exploits knowledge that is represented so that:

- The knowledge captures generalizations that share properties, are grouped together, rather than being allowed separate representation.
- It can be understood by people who must provide it—even though for many programs bulk of the data comes automatically from readings.
- In many AI domains, how the people understand the same people must supply the knowledge to a program.
- It can be easily modified to correct errors and reflect changes in real conditions.
- It can be widely used even if it is incomplete or inaccurate.
- It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be usually considered.

1.3. Introduction to Programming LISP

LISP stands for **LIS**t **P**rogramming. John McCarthy invented LISP in 1958, shortly after the development of FORTRAN. It was first implemented by Steve Russell on an IBM 704 computer. It is particularly suitable for Artificial Intelligence programs, as it processes symbolic information efficiently.

Common LISP originated during the decade of 1980 to 1990, in an attempt to unify the work of several implementation groups, as a successor of Maclisp like ZetaLisp and New Implementation of LISP (NIL) etc.

It serves as a common language, which can be easily extended for specific implementation. Programs written in Common LISP do not depend on machine-specific characteristics, such as word length etc.

1.3.1.Features of Common LISP

- It is machine-independent .
- It uses iterative design methodology
- It has easy extensibility
- It allows to update the programs dynamically
- It provides high level debugging.
- It provides advanced object-oriented programming.
- It provides convenient macro system.
- It provides wide-ranging data types like, objects, structures, lists, vectors, adjustable arrays, hash-tables, and symbols.
- It is expression-based.
- It provides an object-oriented condition system.
- It provides complete I/O library.
- It provides extensive control structures.

1.3.2.Applications Developed in LISP

The following applications are developed in LISP: Large successful applications built in LISP.

- Emacs: It is a cross platform editor with the features of extensibility, customizability, self-document ability, and real-time display.
- G2
- AutoCad
- Igor Engraver
- Yahoo Store

1.3.3.Basic Syntax:

LISP programs are made up of three basic elements:

- atom
- list
- string

An **atom** is a number or string of contiguous characters. It includes numbers and special characters. The following examples show some valid atoms:

```
hello-from-LISP
name
123008907
*hello*
Block#221
abc123
```

A **list** is a sequence of atoms and/or other lists enclosed in parentheses. The following examples show some valid lists:

```
( i am a list)
(a ( a b c) d e fgh)
(father ram( shyam))
(sunmontue wed thurfri sat)
( )
```

A **string** is a group of characters enclosed in double quotation marks. The following examples show some valid strings:

```
" I am a string"
"a ba c d efg #$%^&!"
"Please enter the following details:"
"Hello from 'LISP Classes'! "
```

Adding Comments

The semicolon symbol (;) is used for indicating a comment line.

```
(write-line "Hello World") ; greet the world
; tell them your whereabouts
(write-line "I am   Learning LISP")
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is:

```
Hello World
I am  Learning LISP
```

1.3.4.Notable Points

The following important points are notable:

- The basic numeric operations in LISP are +, -, *, and /
- LISP represents a function call f(x) as (f x), for example cos(45) is written as cos 45
- LISP expressions are not case-sensitive. Means, cos 45 or COS 45 are same.
- LISP tries to evaluate everything, including the arguments of a function. Only three types of elements are constants and always return their own value:
- Numbers
- The letter **t**, that stands for logical true
- The value **nil**, that stands for logical false, as well as an empty list.

1.3.5.Data Type

Scalar types - numbers, characters, symbols etc.

Data structures - lists, vectors, bit-vectors, and strings.

Any variable can take any LISP object as its value, unless you declare it explicitly. Although, it is not necessary to specify a data type for a LISP variable, however, it helps in certain loop expansions, in method declarations.

The data types are arranged into a hierarchy. A data type is a set of LISP objects and many objects may belong to one such set.

The **typep** predicate is used for finding whether an object belongs to a specific type.

The **type-of** function returns the data type of a given object.

Type Specifiers in LISP

Table 1.1 Type specifiers are system-defined symbols for data types.

Array	fixnum	package	simple-string
Atom	float	pathname	simple-vector
Bignum	function	random-state	single-float
Bit	hash-table	Ratio	standard-char
bit-vector	integer	Rational	stream
Character	keyword	readtable	string
[common]	list	sequence	[string-char]
compiled-function	long-float	short-float	symbol
Complex	nill	signed-byte	t
Cons	null	simple-array	unsigned-byte
double-float	number	simple-bit-vector	vector

1.3.6.Macro

A macro is a function that takes an s-expression as arguments and returns a LISP form, which is then evaluated. Macros allow you to extend the syntax of standard LISP.

Defining a Macro:-

In LISP, a named macro is defined using another macro named defmacro. Syntax for defining a macro is:

```
(defmacro macro-name(Optional documentation string.”body-form”)
Parameter-list)
```

The macro definition consists of the name of the macro, a parameter list, an optional documentation string, and a body of LISP expressions that defines the job to be performed by the macro.

Example:-

Let us write a simple macro named setTo10, which takes a number and sets its value to 10.

Create new source code file named main.lisp and type the following code in it:

```
defmacro setTo10(num)
(setq num 10)(print num))
(setq x 25)
(print x)
(setTo10 x)
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result is:

```
25
10
```

1.3.7.Variables

In LISP, each variable is represented by a symbol. The name of the variable is the name of the symbol and it is stored in the storage cell of the symbol.

Global Variables

Global variables are generally declared using the defvarconstruct. Global variables have permanent values throughout the LISP system and remain in effect until new values are specified.

```
 (defvar x 234)
(write x)
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result is:

```
234
```

As there is no type declaration for variables in LISP, you need to specify a value for a symbol directly with the **setq**construct.

```
->(setq x 10)
```

The above expression assigns the value 10 to the variable x. You can refer to the variable using the symbol itself as an expression.

The **symbol-value** function allows you to extract the value stored at the symbol storage place.

Create new source code file named main.lisp and type the following code in it:

```
 (setq x 10)
(setq y 20)
(format t "x = ~2d y = ~2d ~%" x y)
(setq x 100)
(setq y 200)
(format t "x = ~2d y = ~2d" x y)
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result is:

```
x = 10 y = 20
x = 100 y = 200
```

Local Variables

Local variables are defined within a given procedure. The parameters named as arguments within a function definition are also local variables. Local variables are accessible only within the respective function.

Like the global variables, local variables can also be created using the setqconstruct.

There are two other constructs - let and prog for creating local variables.

The let construct has the following syntax:

```
(let ((var1 val1) (var2 val2).. (varnvaln))<s-expressions>)
```

Where var1, var2,…,varn are variable names and val1, val2,…, valn are the initial values assigned to the respective variables.

When **let** is executed, each variable is assigned the respective value and at last, the *s-expression* is evaluated. The value of the last expression evaluated is returned.

If you do not include an initial value for a variable, the variable is assigned to **nil.**

Example

Create new source code file named main.lispand type the following code in it:

```
(let ((x 'a)
(y 'b)
(z 'c))
(format t "x = ~a y = ~a z = ~a" x y z))
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result is:

```
x = A  y = B  z = C
```

The **prog** construct also has the list of local variables as its first argument, which is followed by the body of the **prog,** and any number of s-expressions.

1.3.8.Recursion in LISP

In the Lisp programming language, recursion is a commonly used technique for solving problems. Lisp is a functional programming language, which means it is well-suited to recursive solutions. In LISP, recursion is a programming technique in which a function calls itself repeatedly until a certain condition is met. It is a function that is called by itself more than once. This can be a useful way to solve large problems that can be broken down into smaller subproblems. A function is recursive if it calls itself

Boundary condition: then it's not recursive.

Recursive condition: must be a smaller sub-problem to converge the solution.

In recursion, the recursive function ends until its base condition is satisfied. The base condition is a must in recursion otherwise the recursive function may go into an infinite loop which may cause it to memory's stack overflow. A function that uses recursion is called a recursive function. The below figure shows the syntax and flow of the recursion in LISP.

(defunfunction_name(parameters);

body-of-function

if(base-condition)

return;

body-of-function

function_name(parameters)

)

1.3.9.Loops in LISP

There may be a situation, when you need to execute a block of code numbers of times. A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages.

The **loop** construct is the simplest form of iteration provided by LISP. In its simplest form It allows you to execute some statement(s) repeatedly until it finds a **return** statement.

For Loop

The loop for construct allows you to implement a for-loop like iteration as most common in other languages.It allows you to set up variables for iteration specify expression(s) that will conditionally terminate the iteration specify expression(s) for performing some job on each iteration specify expression(s), and expressions for doing some job before exiting the loop

The for loop for construct follows several syntax –

```
(loop for loop-variable in <a list>
   do (action)
)
```

```
(loop for loop-variable from value1 to value2
   do (action)
)
```

Create a new source code file named main.lisp and type the following code in it −

```
( loop for a from 10 to 13
do (print a)
)
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is −

```
10
11
12
13
```

Do:-The do construct is also used for performing iteration using LISP. It provides a structured form of iteration.

The syntax for do statement −

```
(do((variable1 value1 update-value1)
     (variable2 value2 updated-value2)
…..)
(test return-value)
```

The initial values of each variable is evaluated and bound to the respective variable. The updated value in each clause corresponds to an optional update statement that specifies how the values of the variables will be updated with each iteration.

After each iteration, the test is evaluated, and if it returns a non-nil or true, the return-value is evaluated and returned.

The last s-expression(s) is optional. If present, they are executed after every iteration, until the test value returns true.

Create a new source code file named main.lisp and type the following code in it −

```
(do((x 0(+2 x))
  (y 20(- y 2)))
((=x y) (- x y))
(format t "~%x = ~d   y= ~d " x y)
)
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is −

```
x = 0   y = 20
x =2    y =18
x= 4    y=16
x=6    y=14
x = 8  y=12
```

Dotimes

The dotimes construct allows looping for some fixed number of iterations.

For example, Create a new source code file named main.lisp and type the following code in it

```
(dotimes (n 4)
  (print n) (print (* n n))
)
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is −

```
0 0
1 1
2 4
3 9
```

Dolist

The dolist construct allows iteration through each element of a list.

For example, Create a new source code file named main.lisp and type the following code in it

```
(dolist (n ‘(1 2 3 )
(format t “~% Number: ~d” n(* n n))
)
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is −

```
Number : 1 Square: 1
Number : 2 Square: 4
Number : 3 Square: 9
```

1.3.10. Lists

Lists had been the most important and the primary composite data structure in traditional LISP. Present day's Common LISP provides other data structures like, vector, hash table, classes or structures.

Lists are single linked lists. In LISP, lists are constructed as a chain of a simple record structure named **cons** linked together.

Lists in LISP

Although cons cells can be used to create lists, however, constructing a list out of nested **cons** function calls can't be the best solution. The **list** function is rather used for creating lists in LISP.

The list function can take any number of arguments and as it is a function, it evaluates its arguments.

The **first** and **rest** functions give the first element and the rest part of a list. The following examples demonstrate the concepts.

Create a new source code file named main.lisp and type the following code in it.

```
(defun my-library(title author rating availability)
   (list : title  title : author  author :rating   rating :availability)
)
(write (getf (my-library “Hunger Game”  “Collins” 9t) :title))
```

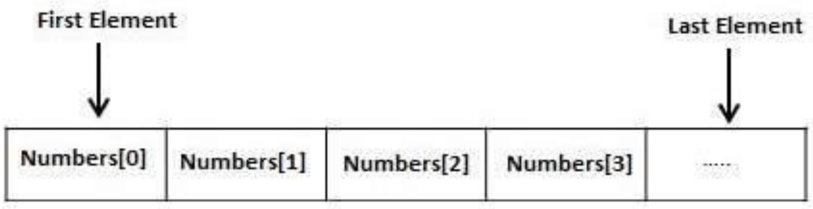
When you execute the code, it returns the following result −

```
“Hunger Game”
```

1.3.11. Array

LISP allows you to define single or multiple-dimension arrays using the **make-array** function. An array can store any LISP object as its elements.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



The number of dimensions of an array is called its rank.

In LISP, an array element is specified by a sequence of non-negative integer indices. The length of the sequence must equal the rank of the array. Indexing starts from zero.

For example, to create an array with 10- cells, named my-array, we can write −

```
(setf my-array (make-array’(10))
```

The aref function allows accessing the contents of the cells. It takes two arguments, the name of the array and the index value.

For example, to access the content of the tenth cell, we write −

```
(arefmy-array 9)
```

Example

Create a new source code file named main.lisp and type the following code in it.

```
(write (setf my-array (make-array '(10))))
(terpri)
(setf (aref my-array 0) 25)
(setf (aref my-array 1) 23)
(setf (aref my-array 3) 10)
(setf (aref my-array 4) 20)
(setf (aref my-array 5) 17)
(setf (aref my-array 6) 25)
(setf (aref my-array 7) 19)
(setf (aref my-array 8) 67)
```

```
(setf (aref my-array 9) 30)
(write my-array)
```

When you execute the code, it returns the following result –

```
#(25 23 45 10 20 17 25 19 67 30)
```

2. PROBLEMS, PROBLEM SPACES AND SEARCH

Problem solving is a processof generating solutions from observed data.

- a "problem" is characterized by a set of *goals*,
- a set of object*s*, and
- a set of operations.

These could be ill-defined and may evolve during problem solving.

- A **'problem space'**is an abstract space.
  - A problem space encompasses all *valid states* that can be generated by the application of any combination of operators on any combination of objects.
  - The problem space may contain one or more *solutions*. A solution is a combination of *operations* and *objects* that achieve the *goals*.
- A '**search**' refers to the search for a solution in a problem space.
  - Search proceeds with different types of '*search control strategies*'.
  - The depth-first search *and* breadth-first searchare the two common search strategies.

2.1. AI - General Problem Solving

Problem solving has been the key area of concern for Artificial Intelligence.

Problem solving is a process of generating solutions from observed or given data. It is however not always possible to use direct methods (i.e. go directly from data to solution). Instead, problem solving often needs to use indirect or model based methods.

General Problem Solver (GPS)was a computer program created in 1957 by Simon and Newellto build a universal problem solver machine. *GPS* was based on Simon and Newell's theoretical work on logic machines. *GPS* in principle can solve any formalized symbolic problem, such as theorems proof and geometric problems and chess playing. *GPS* solved many simple problems, such as the Towers of Hanoi, that could be sufficiently formalized, but **GPS could not solve any real-world problems**.

To build a system to solve a particular problem, we need to:

- Define the problem precisely – find input situations as well as final situations for an acceptable solution to the problem
- Analyze the problem – find few important features that may have impact on the appropriateness of various possible techniques for solving the problem
- Isolate and represent task knowledge necessary to solve the problem
- Choose the best problem-solving technique(s) and apply to the particular problem

Problem definitions

A problem is defined by its 'elements' and their 'relations'. To provide a formal description of a problem, we need to do the following:

a. Define a state space that contains all the possible configurations of the relevant objects, including some impossible ones.
b. Specify one or more states that describe possible situations, from which the problem solving process may start. These states are called initial states.
c. Specify one or more states that would be acceptable solution to the problem.

These states are called goal states.

Specify a set of rules that describe the actions (operators) available.

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem spaceuntil a pathfrom an initial stateto a goal stateis found. This process is known as `search`. Thus:

- Search is fundamental to the problem-solving process.
- Search is a general mechanism that can be used when a more direct

  method is not known.
- Search provides the framework into which more direct methods for

  solving sub parts of a problem can be embedded. A very large number of

  AI problems are formulated as search problems.

Problem space:

A problem space is represented by a directed graph, where *nodes* represent search state and *paths* represent the operators applied to change the *state*.

To simplify search algorithms, it is often convenient to logically and programmatically represent a problem space as a **tree**. A *tree* usually decreases the complexity of a search at a cost. Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph, e.g. node ***B*** and node ***D.***

A tree is a graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph with no cycles is a tree.

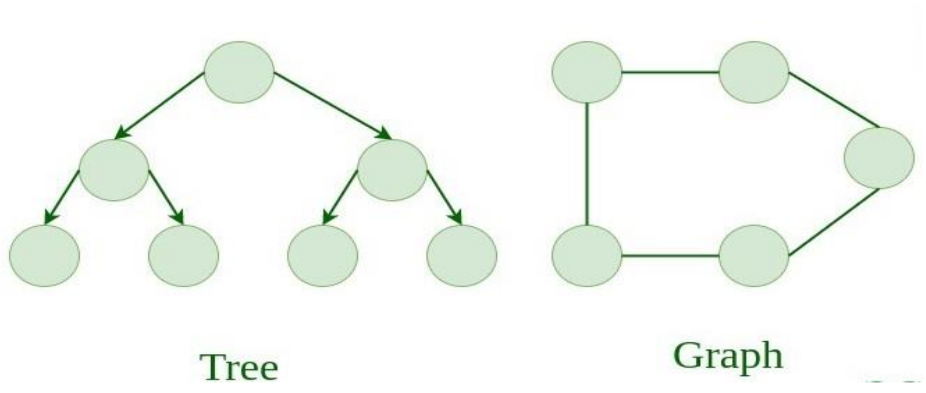


Figure 2.1Tree and Graph

2.2. Forward versus Backward Reasoning

A search procedure must find a path between initial and goal states.

There are two directions in which a search process could proceed.

The two types of search are:

1. Forward search which starts from the start state
2. Backward search that starts from the goal state

We can solve the problem in 2 ways:

Reason forward from the initial state

- **Step 1**. Begin building a tree of move sequences by starting with the initial configuration at the root of the tree.
- **Step 2**. Generate the next level of the tree by finding all rules **whose left-hand** side matches against the root node. The right-hand side is used to create new configurations.
- **Step 3**. Generate the next level by considering the nodes in the previous level and applying it to all rules whose left-hand side match.

Reasoning backward from the goal states:

- **Step 1**. Begin building a tree of move sequences by starting with the goal node configuration at the root of the tree.
- **Step 2**. Generate the next level of the tree by finding all rules **whose right-hand side matches** against the root node. The left-hand side used to create new configurations.
- **Step 3**. Generate the next level by considering the nodes in the previous level and applying it to all rules whose right-hand side match.
- So, The same rules can use in both cases.
- Also, In forwarding reasoning, the left-hand sides of the rules matched against the current state and right sides used to generate the new state.
- Moreover, In backward reasoning, the right-hand sides of the rules matched against the current state and left sides are used to generate the new state.

There are four factors influencing the type of reasoning. They are,

1. Are there more possible start or goal state? We move from smaller set of sets to the length.
2. In what direction is the branching factor greater? We proceed in the direction with the lower branching factor.
3. Will the program be asked to justify its reasoning process to a user? If, so then it is selected since it is very close to the way in which the user thinks.
4. What kind of event is going to trigger a problem-solving episode? If it is the arrival of a new factor, the forward reasoning makes sense. If it is a query to which a response is desired, backward reasoning is more natural.

Example 1 of Forward versus Backward Reasoning

- It is easier to drive from an unfamiliar place from home, rather than from home to an unfamiliar place.Also, If you consider a home as starting place an unfamiliar place as a goal then we have to backtrack from unfamiliar place to home.

Example 2 of Forward versus Backward Reasoning

- Consider a problem of symbolic integration. Moreover, The problem space is a set of formulas, which contains integral expressions. Here START is equal to the given formula with some integrals. GOAL is equivalent to the expression of the formula without any integral. Here we start from the formula with some integrals and proceed to an integral free expression rather than starting from an integral free expression.

Example 3 of Forward versus Backward Reasoning

- The third factor is nothing but deciding whether the reasoning process can justify its reasoning. If it justifies then it can apply. For example, doctors are usually unwilling to accept any advice from diagnostics process because it cannot explain its reasoning.

Example 4 of Forward versus Backward Reasoning

- Prolog is an example of backward chaining rule system. In Prolog rules restricted to Horn clauses. This allows for rapid indexing because all the rules for deducing a given fact share the same rule head. Rules matched with unification procedure. Unification tries to find a set of bindings for variables to equate a sub-goal with the head of some rule. Rules in the Prolog program matched in the order in which they appear.

Combining Forward and Backward Reasoning

Instead of searching either forward or backward, you can search both simultaneously.

Also, That is, start forward from a starting state and backward from a goal state simultaneously until the paths meet.

This strategy called Bi-directional search. The *Figure2.2* shows the reason for a Bidirectional search to be ineffective.

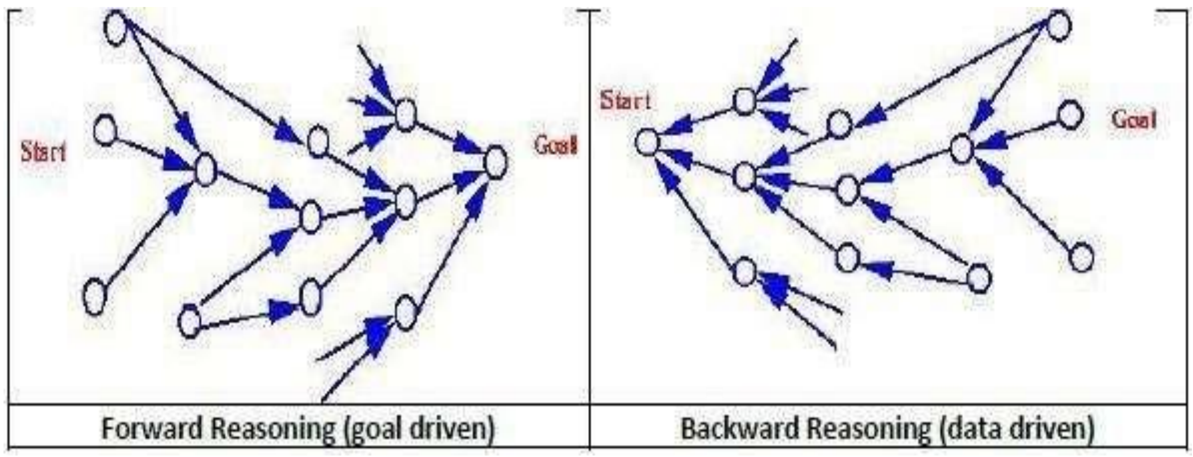

Start
Goal
Start
Goal
Forward Reasoning (goal driven)
Backward Reasoning (data driven)

Figure 2.2 Backward Reasoning and Forward Reasoning

Forward versus Backward Reasoning

- Also, The two searches may pass each other resulting in more work.
- Based on the form of the rules one can decide whether the same rules can apply to both forward and backward reasoning.
- Moreover, If left-hand side and right of the rule contain pure assertions then the rule can reverse.
- And so the same rule can apply to both types of reasoning.
- If the right side of the rule contains an arbitrary procedure then the rule cannot reverse.
- So, In this case, while writing the rule the commitment to a direction of reasoning must make.

2.3. Exhaustive Search Algorithm:

Exhaustive Search is a brute-force algorithm that systematically enumerates all possible solutions to a problem and checks each one to see if it is a valid solution. This algorithm is typically used for problems that have a small and well-defined search space, where it is feasible to check all possible solutions.

Examples:

Input : item[] ={1,2,3,4,5,6};

List of sets ={1,2,3},{4,5},{5,6},{1,4}

Output: Maximum number of sets that can be packed :3

Input: Items[]={1,2};

List of sets ={1},{4, },{5},{1}

Output : Maximum number of sets that can be packed: 1

Approach:

Loop through all the sets and check if the current item is in the current set If the item is in the set, increment the number of sets that can be packed and Update the maximum number of sets that can be packed

Here is a step-by-step description of how the algorithm works in this code:

- The program defines a **maxPackedSets()** function that takes a set of items and a list of sets as input.
- The function initializes the maximum number of sets that can be packed to **0**.
- The function loops through all the sets in the list of sets. For each set, it initializes the number of sets that can be packed to 0.
- The function then loops through all the items in the set of items. For each item, it checks if the item is in the current set. If the item is in the set, the number of sets that can be packed is incremented and the item is removed from the set of items so that it is not counted again.
- The function then updates the maximum number of sets that can be packed by taking the maximum of the current maximum and the number of sets that can be packed for the current set.
- The function repeats steps **3-5** for all the sets in the list of sets.
- Once all the sets have been processed, the function returns the maximum number of sets that can be packed as the result.
- The main() function of the program creates a set of items and a list of sets and then calls the **maxPackedSets()** function to find the maximum number of sets that can be packed into the given set of items.
- The result of the **maxPackedSets()** function is printed to the console, indicating the maximum number of sets that can be packed.

Thus, the code uses the Exhaustive Search algorithm to systematically enumerate and check all the sets in the list of sets to find the maximum number of sets that can be packed into the given set of items. This is a simple and straightforward approach to solving the Set Packing problem, but it can be slow and computationally expensive for problems with a large search space.

2.3.1.Characteristics of Exhaustive Search Algorithm:

- The Exhaustive Search algorithm is a simple and straightforward approach to solving problems.
- However, it can be slow and computationally expensive, especially for problems with a large search space.

- It is also not guaranteed to find the optimal solution to a problem, as it only checks solutions that are explicitly enumerated by the algorithm.
- Despite these limitations, Exhaustive Search can be a useful technique for solving certain types of problems.

Complexity Analysis:

Time Complexity: The complexity of the **maxPackedSets** function is O(nm) where n is the length of the items array and m is the number of sets in the sets list. This is because the function loops through all the items (n) and all the sets (m) in two separate loops.

- The complexity of the contains function is **O(n)** because it loops through all the elements in the array (n).
- The complexity of the remove function is **O(n)** because it loops through all the elements in the array (n) and adds them to a new list if they are not the value to be removed.
- Overall, the complexity of the entire program is **O(n*m)**

AuxiliarySpace: O(n*m)The space complexity of the **maxPackedSets** function is O(nm) because it creates a new list for the result of the **remove** function for each item in the **items** array and each set in the **sets** list.

- The space complexity of the **contains** function is O(1) because it only uses a fixed number of variables regardless of the size of the input.
- The space complexity of the **remove** function is O(n) because it creates a new list with a size equal to the number of elements in the input array that are not the value to be removed.
- Overall, the space complexity of the entire program is **O(n*m)**

2.4. Depth First Search (DFS)

Depth-first search contributes to its effectiveness and optimization in artificial intelligence. From algorithmic insights to real-world implementations, DFS plays a huge role in optimizing AI systems. Let’s dive into the fundamentals of DFS, its significance in artificial intelligence, and its practical applications.

2.4.1. What is a Depth-First Search in AI?

Depth-first search is a traversing algorithm used in tree and graph-like data structures. It generally starts by exploring the deepest node in the frontier. Starting at the root node, the algorithm proceeds to search to the deepest level of the search tree until nodes with no successors are reached. Suppose the node with unexpanded successors is encountered then the search backtracks to the next deepest node to explore alternative paths.

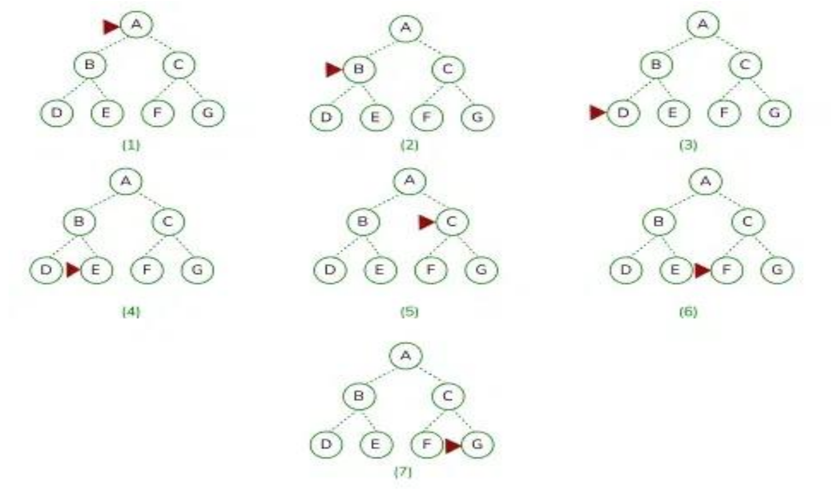


Figure 2.3 DFS Searching

Depth-first search (DFS) explores a graph by selecting a path and traversing it as deeply as possible before backtracking.

- Originally it starts at the root node, then it expands all of its one branch until it reaches a dead end, then backtracks to the most recent unexplored node, repeating until all nodes are visited or a specific condition is met. (From Figure 2.3, starting from node A, DFS explores its successor B, then proceeds to its descendants until reaching a dead end at node D. It then backtracks to node B and explores its remaining successors i.e E.)
- This systematic exploration continues until all nodes are visited or the Search terminates. (In our case after exploring all the nodes of B. DFS explores the right side node i.e C then F and and then G. After exploring the node G. All the nodes are visited. It will terminate.

2.4.2.Key characteristics of DFS

- In simple terms, the DFS algorithms in AI holds the power of extending the current path as deeply as possible before considering the other options.
- DFS is **not cost-optimal** since it doesn't guarantee to find the shortest paths.
- **DFS uses the simple principle to keep track of visited nodes:** It uses a stack to keep track of nodes that have been visited so far which helps in the backtracking of the graph. When the DFS encounters a new node, it adds it to the stack to explore its neighbours. If it reaches a node with no successors (leaf node), it works by backtracking such as popping nodes off the stack to explore the alternative paths.
- **Backtracking search:** The **variant of DFS** is called backtracking search which **uses less memory** than traditional depth-first search. Rather than generating all the successors, the backtracking search enables the DFS to generate only one successor at a time. This approach allows **dynamic state modification**, such as generating successors by directly modifying the current state description instead of allocating the memory to a brand-new state. Thus **reducing the memory requirements** to store one state description and path of actions.

2.4.3.Edge classes in a Depth-first search tree based on a spanning tree

The edges of the depth-first search tree can be divided into four classes based on the spanning tree, they are

- **Forward edges:** The forward edge is responsible for pointing from a **node** of the tree to one of its **successors**.
- **Back edges:** The back edge holds the power of directing its edge from a **node** of the tree to one of its **ancestors**.
- **Tree edges:** When DFS explores the new vertex from the current vertex, the edge connecting them is called a tree edge. It is essential while constructing a DFS spanning tree because it represents the paths to be followed during the traversal.
- **Cross edges:** Cross edges are the edges that connect two vertices that are neither ancestors nor descendants of each other in the DFS tree.

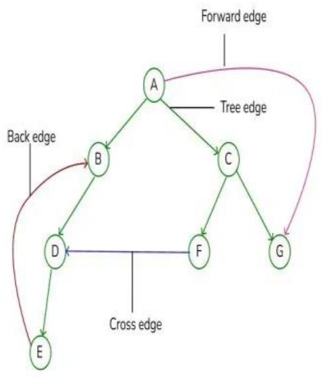


Figure 2.4Depth search

Step wise DFS Pseudocode Explanatioin

1. Initialize an empty **stack** and an empty set for visited vertices.
2. Push the start vertex onto the stack.
3. While the stack is not empty:
   - Pop the current vertex.
   - If it's the goal vertex, return "Path found".
   - Add the current vertex to the visited set.
   - Get the neighbors of the current vertex.
   - For each neighbor not visited, push it onto the stack.
4. If the loop completes without finding the goal, return "Path not found".

Time complexity of DFS:

- **Explicit Time Complexity:** This time complexity arises because DFS traverses each vertex and edge exactly once in the worst-case scenario. This complexity is for explicit traversing of DFS without any repetition. The time complexity of DFS is commonly represented as **O(|V| +|E|)**

  Where, V- represents the number of vertices, E – represents the number of edges**.**

- **Implicit Time Complexity:** This implicit time complexity is appropriate while considering the time complexity in terms of the number of nodes visited in the tree instead of the number of edges and vertices in a graph. For implicit traversing of DFS can be simply depicted as **O(b^d)**

  Where,b – represents the branching factor,d – represents the depth of the search.

Space complexity of DFS:

- **Explicit Space Complexity:** The space complexity of DFS is commonly represented as **O(|V|)**

  Where,V– represents the number of vertices.

This is because DFS typically uses additional data structures like stack or recursion stack to keep track of visited vertices.

- **Implicit Space Complexity:** For an implicit search in a graph, DFS's space complexity can be represented as follows:

  O(bd)

  where,b – represents the branching factor, d – represents the depth of the search.

  This space complexity is said to be considered when the space is required to store the nodes on the current path during the search.

2.5. Breadth-first search(BFS)

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Applications of BFS algorithm

The applications of breadth-first algorithm are given as follows -

- BFS can be used to find the neighboring locations from a given source location.
- In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
- BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.
- BFS is used to determine the shortest path and minimum spanning tree.
- BFS is also used in Cheney's technique to duplicate the garbage collection.
- It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

Algorithm

The steps involved in the BFS algorithm to explore a graph are given as follows -

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

Step 6: EXIT

Example of BFS algorithm

Now, let's understand the working of BFS algorithm by using an example. In the example given below, there is a directed graph having 7 vertices.

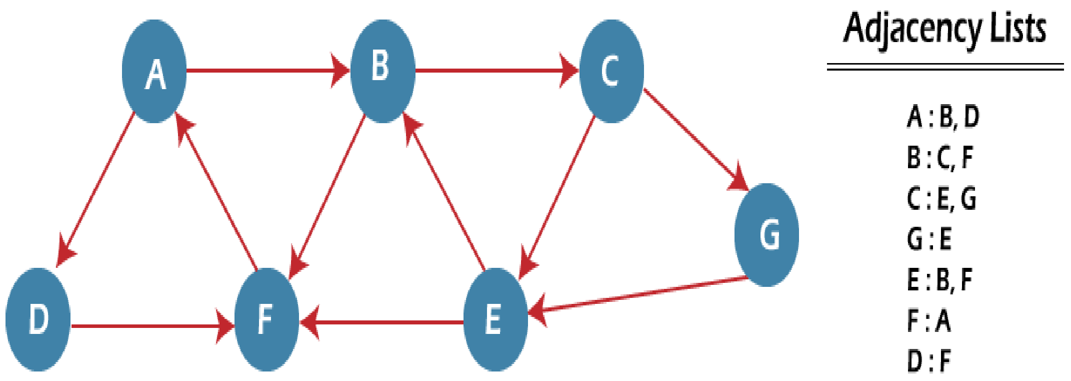


Figure 2.5 BFS Graph

in the Figure 2.5, minimum path 'P' can be found by using the BFS that will start from Node A and end at Node E. The algorithm uses two queues, namely QUEUE1 and QUEUE2. QUEUE1 holds all the nodes that are to be processed, while QUEUE2 holds all the nodes that are processed and deleted from QUEUE1.

Now, let's start examining the graph starting from Node A.

Step 1 - First, add A to queue1 and NULL to queue2.

QUEUE1 = {A}

QUEUE2 = {NULL}

Step 2 - Now, delete node A from queue1 and add it into queue2. Insert all neighbors of node A to queue1.

QUEUE1 = {B, D}

QUEUE2 = {A}

Step 3 - Now, delete node B from queue1 and add it into queue2. Insert all neighbors of node B to queue1.

QUEUE1 = {D, C, F}

QUEUE2 = {A, B}

Step 4 - Now, delete node D from queue1 and add it into queue2. Insert all neighbors of node D to queue1. The only neighbor of Node D is F since it is already inserted, so it will not be inserted again.

QUEUE1 = {C, F}

QUEUE2 = {A, B, D}

Step 5 - Delete node C from queue1 and add it into queue2. Insert all neighbors of node C to queue1.

QUEUE1 = {F, E, G}

QUEUE2 = {A, B, D, C}

Step 5 - Delete node F from queue1 and add it into queue2. Insert all neighbors of node F to queue1. Since all the neighbors of node F are already present, we will not insert them again.

QUEUE1 = {E, G}

QUEUE2 = {A, B, D, C, F}

Step 6 - Delete node E from queue1. Since all of its neighbors have already been added, so we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.

QUEUE1 = {G}

QUEUE2 = {A, B, D, C, F, E}

Complexity of BFS algorithm

Time complexity of BFS depends upon the data structure used to represent the graph. The time complexity of BFS algorithm is **O(V+E)**, since in the worst case, BFS algorithm explores every node and edge. In a graph, the number of vertices is O(V), whereas the number of edges is O(E).

The space complexity of BFS can be expressed as **O(V)**, where V is the number of vertices.

Implementation of BFS algorithm

Now, let's see the implementation of BFS algorithm in java.

In this code, we are using the adjacency list to represent our graph. Implementing the Breadth-First Search algorithm in Java makes it much easier to deal with the adjacency list since we only have to travel through the list of nodes attached to each node once the node is dequeued from the head (or start) of the queue.

In this example, the graph that we are using to demonstrate the code is given as follows –

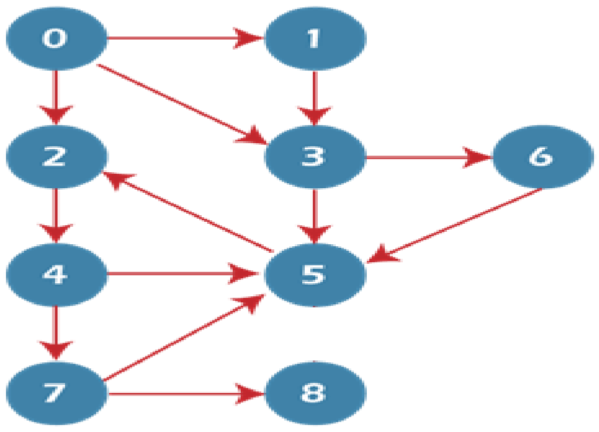


Figure 2.6BFS

2.6. Heuristics Search

To find a solution in proper time rather than a complete solution in unlimited time we use heuristics. ‘A heuristic function is a function that maps from problem state descriptions to measures of desirability, usually represented as numbers’. Heuristic search methods use knowledge about the problem domain and choose promising operators first. These heuristic search methods use heuristic functions to evaluate the next state towards the goal state. For finding a solution, by using the heuristic technique, one should carry out the following steps:

1. Add domain—specific information to select what is the best path to continue searching along.

2. Define a heuristic function h(n) that estimates the ‘goodness’ of a node n. Specifically, h(n) = estimated cost(or distance) of minimal cost path from n to a goal state.

3. The term, heuristic means ‘serving to aid discovery’ and is an estimate, based on domain specific information that is computable from the current state description of how close we are to a goal.

Finding a route from one city to another city is an example of a search problem in which different search orders and the use of heuristic knowledge are easily understood.

1. State: The current city in which the traveller is located.

2. Operators: Roads linking the current city to other cities.

3. Cost Metric: The cost of taking a given road between cities.

4. Heuristic information: The search could be guided by the direction of the goal city from the current city, or we could use airline distance as an estimate of the distance to the goal.

Heuristic search techniques For complex problems, the traditional algorithms, presented above, are unable to find the solution within some practical time and space limits. Consequently, many special techniques are developed, using heuristic functions.

- Blind search is not always possible, because it requires too much time or Space (memory). Heuristics are rules of thumb; they do not guarantee a solution to a problem.
- Heuristic Search is a weak technique but can be effective if applied correctly; it requires domain specific information.

Characteristics of heuristic search

- Heuristics are knowledge about domain, which help search and reasoning in its domain.
- Heuristic search incorporates domain knowledge to improve efficiency over blind search.
- Heuristic is a function that, when applied to a state, returns value as estimated merit of state, with respect to goal.

Heuristics might (for reasons) underestimate or overestimate the merit of a state with respect to goal. Heuristics that underestimate are desirable and called admissible.

- Heuristic evaluation function estimates likelihood of given state leading to goal state.
- Heuristic search function estimates cost from current state to goal, presuming function is efficient

Add domain -specific information to select what is the best path to continue searching along.

Define a heuristic function h(n) that estimates the‘goodness’ of anoden.

Specifically,h(n) = estimated cost(or distance) of minimal cost path from n toa goalstate.

The term, heuristic means ‘serving to aid discovery’ and is an estimate, based on domain specific information that is computable from the current state description of how close wear e to a goal.

Finding a route from one city to another city is an example of a search problem in which different search orders and the use of heuristic knowledge are easily understood.

1. State : The current city in which the traveller islocated.
2. Operators : Roads linking the current city to other cities.
3. Cost Metric : The cost of taking a given road between cities.
4. Heuristic information: The search could be guided by the direction of the goal city from the current city, or we could use airline distance as an estimate of the distance to the goal.

Heuristic search techniques

For complex problems, the traditional algorithms, presented above, are unable to find the solution within some practical time and space limits. Consequently, many special techniques are developed, using heuristic functions.

- Blind search is not always possible, because it requires too much time or Space(memory).
- Heuristics are rules of thumb; they do not guarantee a solution to a problem.
- Heuristic Search is a weak technique but can be effective if applied correctly; it requires domain specific information.

Characteristics of heuristic search

- Heuristics are knowledge about domain, which help search and reasoning in its domain.
- Heuristic search in corporates domain knowledge to improve efficiency over blind search.
- Heuristic is a function that, when applied to a state, returns value as estimated merit of state, with respect to goal.
- Heuristic evaluation function estimates likelihood of given state leading to goal state.
- Heuristic search function estimates cost from current state to goal, presuming function is efficient.

2.7. Hill Climbing

Hill Climbing is heuristic search used for mathematical optimization problems in the field of Artificial Intelligence .

Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

- In the above definition, mathematical optimization problems implies that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example-Travelling salesman problem where weneed to minimize the distance travelled by salesman.
- ‘Heuristic search’ means that this search algorithm may not find the optimal solution tothe problem. However, it will give a good solution in reasonable time.

- A heuristic function is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

2.7.1.Features of Hill Climbing

1. Variant of generate and test algorithm : It is a variant of generate and test algorithm. The generate and test algorithm is as follows :
    - Generate a possible solutions.
    - Test to see if this is the expected solution.
    - If the solution has been found quit else go to step 1.
    - Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from test procedure. Then this feedback is utilized by the generator in deciding the next move insearch space.

2. Uses the Greedy approach : At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution atthe end.

2.7.2.Types of Hill Climbing

1.Simple Hill climbing : It examines the neighbouring nodes one by one and selects the first neighbouring node which optimizes the current cost as next node.

Algorithm for Simple Hill climbing :

Step 1 : Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.

Step 2 **:** Loop until the solution state is found or there are no new operators present which can beapplied to current state.

i) Select a state that has not been yet applied to the current state and apply it to produce a newstate.
ii) Perform these to evaluate new state
    a) If the current state is a goal state, then stop and return success.
    b) If it is better than the current state, then make it current state and proceed further.
    c) If it is not better than the current state, then continue in the loop until a solution is found.

Step 3 : Exit.

2.Steepest-Ascent Hill climbing : It first examines all the neighbouring nodes and thenselects the node closest to the solution state as next node.

Step 1 : Evaluate the initial state. If it is goal state then exit else make the current state as initialstate

Step 2 : Repeat these steps until a solution is found or current state does not change

i. Let ‘target’ be a state such that any successor of the current state will be better than it;
ii. for each operator that applies to the current state
    a) apply the new operator and create a new state
    b) evaluate the new state
    c) if this state is goal state then quit else compare with ‘target’
    d) if this state is better than ‘target’, set this state as ‘target’
    e) if target is better than current state set current state to Target

Step 3 : Exit

3.Stochastic hill climbing : It does not examine all the neighboring nodes before deciding which node to select .It just selects a neighbouring node at random, and decides (based onthe amount of improvement in that neighbour) whether to move to that neighbour or to examine another.

2.7.3.State Space diagram for Hill Climbing

State space diagram is a graphical representation of the set of states our search algorithm can

reach vs the value of our objective function(the function which we wish to maximize).

X-axis : denotes the state space ie states or configuration our algorithm may reach.

Y-axis : denotes the values of objective function corresponding to to a particular state.

The best solution will be that state space where objective function has maximum value(global maximum).

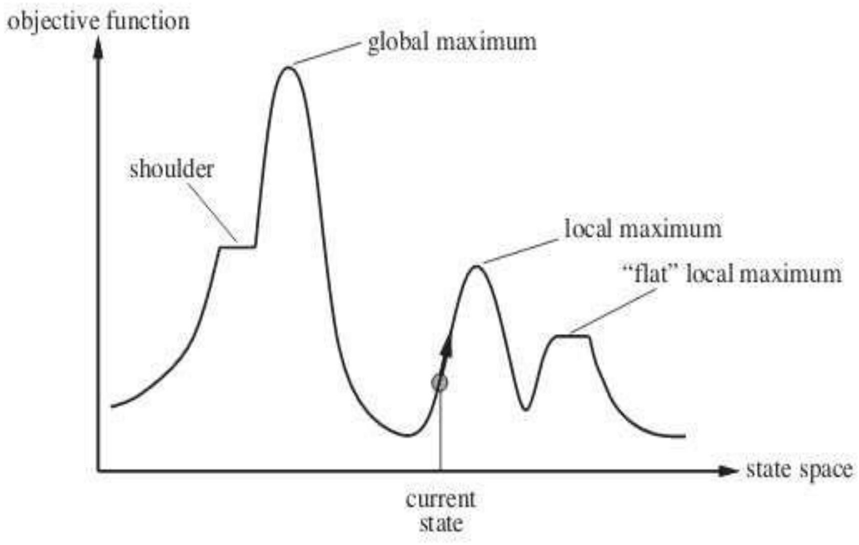


Figure 2.7Hill Climbing

Different regions in the State Space Diagram

Local maximum : It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here value of objective function is higher than its neighbors.

Global maximum : It is the best possible state in the state space diagram. This because atthis state, objective function has highest value.

Plateua/flat local maximum : It is a flat region of state space where neighboring stateshave the same value.

Ridge : It is region which is higher than its neighbours but itself has a slope. It is a specialkind of local maximum.

Current state : The region of state space diagram where we are currently present duringthe search.

Shoulder : It is a plateau that has an uphill edge.

2.7.4.Problems in different regions in Hill climbing

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

Local maximum : At a local maximum all neighbouring states have a values which is worse than than the current state. Since hill climbing uses greedy approach, it will not move to the worse state and terminate itself. The process will end even though a bettersolution may exist.

To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

Plateau : On plateau all neighbours have same value . Hence, it is not possible to select the best direction.

To overcome plateau : Make a big jump. Randomly select a state far away from current state. Chances are that we will land at a non-plateau region

Ridge : Any point on a ridge can look like peak because movement in all possibledirections is downward. Hence the algorithm stops when it reaches this state.

To overcome Ridge : In this kind of obstacle, use two or more rules before testing. Itimplies moving in several distance at once.

2.8. Branch and Bound

Branch and Bound is an algorithmic technique which finds the optimal solution by keeping the best solution found so far. If partial solution can't improve on the best it is abandoned, by this method the number of nodes which are explored can also be reduced. It also deals with the optimization problems over a search that can be presented as the leaves of the search tree. The usual technique for eliminating the sub trees from the search tree is called pruning. For Branch and Bound algorithm we will use stack data structure**.**

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node that is maintaining least distance from the root node.

Step 3: Traverse any neighbour of the neighbour of the root node that is maintaining least distance from the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: PUSH the root node into the stack.

Step 2: If stack is empty, then stop and return failure.

Step 3: If the top node of the stack is a goal node, then stop and return success.

Step 4: Else POP the node from the stack. Process it and find all its successors. Find out the path containing all its successors as well as predecessors and then PUSH the successors which are belonging to the minimum or shortest path.

Step 5**:** Go to step 5.

Step 6**:** Exit.

2.8.1. Implementation:

Let us take the following example for implementing the Branch and Bound algorithm.

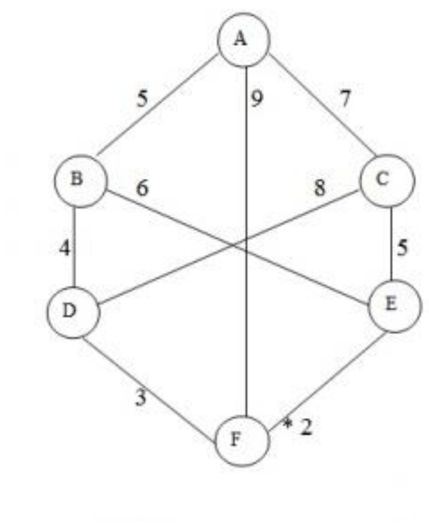
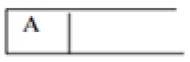


Figure 2.8 Branch and Bound

Step 1:

Consider the node A as our root node. Find its successors i.e B,C,F. Calculate the distance from the root and PUSH them according to least distance.

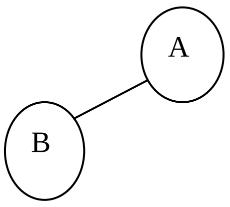


B: 0+5=5 (The cost of A is 0 as it is the starting node)

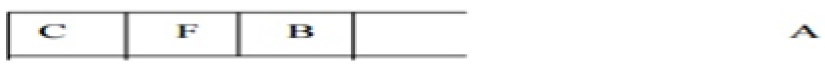
F: 0+9= 9

C: 0+7=7

Here B(5) is the least distance



Step 2: Now stack will be

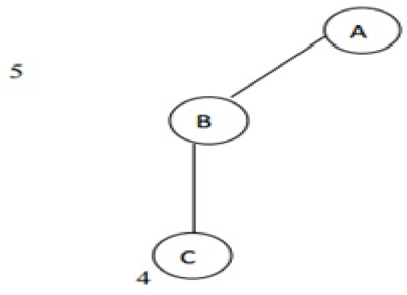


As B is on the top of the stack so calculate the neighbours of B.

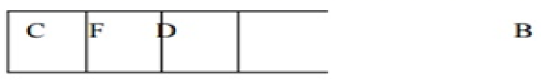
D: 0+5+4=9

E: 0+5+6=11

The least distance is D from B.So it will be on the top of the stack.



Step 3: As the top of the stack is D. So calculate neighbours of D.

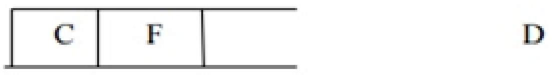


C: 0+5+4+8=17

F: 0+5+4+3=12

The least distance is F from D and it is our goal node. So stop and return success

Step 4:



Hence the searching path will be A-B -D-F

Advantages:

- As it finds the minimum path instead of finding the minimum successor so there should not be any repetition.
- The time complexity is less compared to other algorithms.

Disadvantages:

- The load balancing aspects for Branch and Bound algorithm make it parallelization difficult.
- The Branch and Bound algorithm is limited to small size network. In the problem of large networks, where the solution search space grows exponentially with the scale of the network, the approach becomes relatively prohibitive.

2.9. Best First Search

In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

We use a priority queue to store costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to Priority Queue.

Algorithm:

```
Best-First-Search(Grah g, Node start)

    1)  Create an empty PriorityQueue
        PriorityQueue pq;
    2)  Insert "start" in pq.
        pq.insert(start)
    3)  Until PriorityQueue is empty
        u = PriorityQueue.DeleteMin
        If u is the goal
        Exit
          Else
        Foreach neighbor v of u
                If v "Unvisited"
                Mark v "Visited"
                pq.insert(v)
        Mark v "Examined"
        End procedure
        Let us consider below example.
```

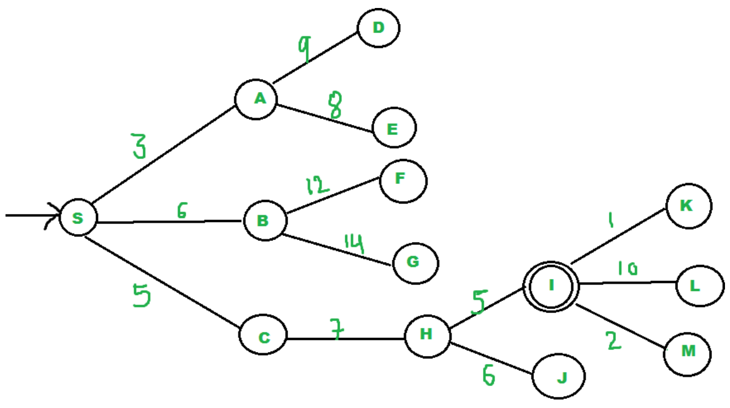


Figure 2.9 Best Search

Step 1:-We start from source "S" and search for goal "I" using given costs and Best First search.

Step 2 :-pq initially contains S We remove s from and process unvisitedneighbors of S to pq.

pq now contains {A, C, B} (C is putbefore B because C has lesser cost)

Step 3 :-We remove A from pq and process unvisitedneighbors of A to pq.

pq now contains {C, B, E, D}

Step 4:-We remove C from pq and process unvisitedneighbors of C to pq.

pq now contains {B, H, E, D}

Step 5:-We remove B from pq and process unvisitedneighbors of B to pq.

pq now contains {H, E, D, F, G}

We remove H from pq. Since our goal"I" is a neighbor of H, we return.

Analysis :

- The worst case time complexity for Best First Search is O(n * Log n) where n is numberof nodes. In worst case, we may have to visit all nodes before we reach goal. Note that priority queue is implemented using Min(or Max) Heap, and insert and remove operations take O(log n) time.
- Performance of the algorithm depends on how well the cost or evaluation function is designed.

2.10. A* Algorithm

A* (pronounced "A-star") is a powerful graph traversal and pathfinding algorithm widely used in artificial intelligence and computer science. It is mainly used to find the shortest path between two nodes in a graph, given the estimated cost of getting from the current node to the destination node. The main advantage of the algorithm is its ability to provide an optimal path by exploring the graph in a more informed way compared to traditional search algorithms such as Dijkstra's algorithm.

Algorithm A* combines the advantages of two other search algorithms: Dijkstra's algorithm and Greedy Best-First Search. Like Dijkstra's algorithm, A* ensures that the path found is as short as possible but does so more efficiently by directing its search through a heuristic similar to Greedy Best-First Search. A heuristic function, denoted h(n), estimates the cost of getting from any given node n to the destination node.

The main idea of A* is to evaluate each node based on two parameters:

1) **g(n):** the actual cost to get from the initial node to node n. It represents the sum of the costs of node n outgoing edges.
2) **h(n):** Heuristic cost (also known as "estimation cost") from node n to destination node n. This problem-specific heuristic function must be acceptable, meaning it never overestimates the actual cost of achieving the goal. The evaluation function of node n is defined as f(n) = g(n) h(n).

Algorithm A* selects the nodes to be explored based on the lowest value of f(n), preferring the nodes with the lowest estimated total cost to reach the goal. The A* algorithm works:

1) Add a starting node to the open list with an initial value of g
2) Create an open list of found but not explored nodes.
3) Create a closed list to hold already explored nodes.
4) Repeat the following steps until the open list is empty or you reach the target node:
5) Find the node with the smallest f-value (i.e., the node with the minor g(n) h(n)) in the open list.
6) Move the selected node from the open list to the closed list.
7) Create all valid descendants of the selected node.
8) For each successor, calculate its g-value as the sum of the current node's g value and the cost of moving from the current node to the successor node. Update the g-value of the tracker when a better path is found.
9) If the follower is not in the open list, add it with the calculated g-value and calculate its h-value. If it is already in the open list, update its g value if the new path is better.
10) Repeat the cycle. Algorithm A* terminates when the target node is reached or when the open list empties, indicating no paths from the start node to the target node. The A* search algorithm is widely used in various fields such as robotics, video games, network routing, and design problems because it is efficient and can find optimal paths in graphs or networks

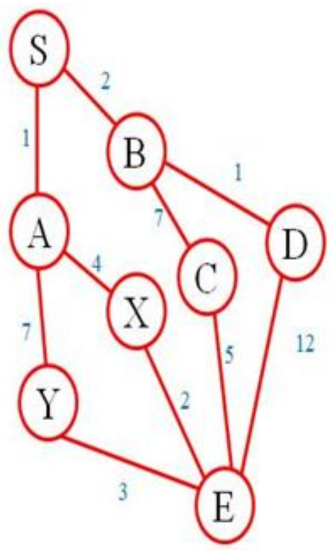


Figure 2.10 A Algorithm Graph*

Values for h:

A:5, B:6, C:4, D:15, X:5, Y:8

Solution:- Expand S

{S,A} f=1+5=6

{S,B} f=2+6=8

Expand A

{S,B} f=2+6=8

{S,A,X} f=(1+4)+5=10

{S,A,Y} f=(1+7)+8=16

Expand B

{S,A,X} f=(1+4)+5=10

{S,B,C} f=(2+7)+4=13

{S,A,Y} f=(1+7)+8=16

{S,B,D} f=(2+1)+15=18

Expand X

{S,A,X,E} is the best path…. (costing 7)

2.10.1. Advantages of A* Search Algorithm

The A* search algorithm offers several advantages in artificial intelligence and problem-solving scenarios.

1) **Optimal solution:** A* ensures finding the optimal (shortest) path from the start node to the destination node in the weighted graph given an acceptable heuristic function. This optimality is a decisive advantage in many applications where finding the shortest path is essential.

2) **Completeness:** If a solution exists, A* will find it, provided the graph does not have an infinite cost This completeness property ensures that A* can take advantage of a solution if it exists.
3) **Efficiency:** A* is efficient ifan efficient and acceptable heuristic function is used. Heuristics guide the search to a goal by focusing on promising paths and avoiding unnecessary exploration, making A* more efficient than non-aware search algorithms such as breadth-first search or depth-first search.
4) **Versatility:** A* is widely applicable to variousproblem areas, including wayfinding, route planning, robotics, game development, and more. A* can be used to find optimal solutions efficiently as long as a meaningful heuristic can be defined.
5) **Optimized search:** A* maintains a priority order to select the nodes with the minor f(n) value (g(n) and h(n)) for expansion. This allows it to explore promising paths first, which reduces the search space and leads to faster convergence.
6) **Memory efficiency:** Unlike some other search algorithms, such as breadth-first search, A* stores only a limited number of nodes in the priority queue, which makes it memory efficient, especially for large graphs.
7) **Tunable Heuristics:** A*'s performancecan be fine-tuned by selecting different heuristic functions. More educated heuristics can lead to faster convergence and less expanded nodes.
8) **Extensively researched:** A* is a well-established algorithm with decades of research and practical applications. Many optimizations and variations have been developed, making it a reliable and well-understood troubleshooting tool.
9) **Web search:** A* can be used for web-based path search, where the algorithm constantly updates the path according to changes in the environment or the appearance of new It enables real-time decision-making in dynamic scenarios.

2.10.2. Disadvantages of A* Search Algorithm

Although the A* (letter A) search algorithm is a widely used and powerful technique for solving AI path finding and graph traversal problems, it has disadvantages and limitations. Here are some of the main disadvantages of the search algorithm:

1) **Heuristic accuracy:** The performance of the A* algorithm depends heavily on the accuracy of the heuristic function used to estimate the cost from the current node to the If the heuristic is unacceptable (never overestimates the actual cost) or inconsistent (satisfies the triangle inequality), A* may not find an optimal path or may explore more nodes than necessary, affecting its efficiency and accuracy.
2) **Memory usage:** A* requires that all visited nodes be kept in memory to keep track of explored paths. Memory usage can sometimes become a significant issue, especially when dealing with an ample search space or limited memory resources.
3) **Time complexity:** Although A* is generally efficient, its time complexity can be a concern for vast search spaces or graphs. In the worst case, A* can take exponentially longer to find the optimal path if the heuristic is inappropriate for the problem.
4) **Bottleneck at the destination:** In specific scenarios, the A* algorithm needs to explore nodes far from the destination before finally reaching the destination region. This the problem occurs when the heuristic needs to direct the search to the goal early effectively.
5) **Cost Binding:** A* faces difficulties when multiple nodes have the same f-value (the sum of the actual cost and the heuristic cost). The strategy used can affect the optimality and efficiency of the discovered path. If not handled correctly, it can lead to unnecessary nodes being explored and slow down the algorithm.
6) **Complexity in dynamic environments:** In dynamic environments where the cost of edges or nodes may change during the search, A* may not be suitable because it does not adapt well to such changes. Reformulation from scratch can be computationally expensive, and D* (Dynamic A*) algorithms were designed to solve this

7) **Perfection in infinite space :** A* may not find a solution in infinite state space. In such cases, it can run indefinitely, exploring an ever-increasing number of nodes without finding a solution. Despite these shortcomings, A* is still a robust and widely used algorithm because it can effectively find optimal paths in many practical situations if the heuristic function is well-designed and the search space is manageable. Various variations and variants of A* have been proposed to alleviate some of its limitations.

2.10.3. Applications of the A* Search

The search algorithm A* (letter A) is a widely used and robust path finding algorithm in artificial intelligence and computer science. Its efficiency and optimality make it suitable for various applications. Here are some typical applications of the A* search algorithm in artificial intelligence:

1) **Path finding in Games:** A* is often used in video games for character movement, enemy AI navigation, and finding the shortest path from one location to another on the game map. Its ability to find the optimal path based on cost and heuristics makes it ideal for real-time applications such as games.
2) **Robotics and Autonomous Vehicles:** A* is used in robotics and autonomous vehicle navigation to plan an optimal route for robots to reach a destination, avoiding obstacles and considering terrain costs. This is crucial for efficient and safe movement in natural environments.
3) **Maze solving:** A* can efficiently find the shortest path through a maze, making it valuable in many maze-solving applications, such as solving puzzles or navigating complex structures.
4) **Route planning and navigation:** In GPS systems and mapping applications, A* can be used to find the optimal route between two points on a map, considering factors such as distance, traffic conditions, and road network topology.
5) **Puzzle-solving:** A* can solve various diagram puzzles, such as sliding puzzles, Sudoku, and the 8-puzzle problem. Resource Allocation: In scenarios where resources must be optimally allocated, A* can help find the most efficient allocation path, minimizing cost and maximizing efficiency.
6) **Network Routing:** A* can be used in computer networks to find the most efficient route for data packets from a source to a destination node.
7) **Natural Language Processing (NLP):** In some NLP tasks, A* can generate coherent and contextual responses by searching for possible word sequences based on their likelihood and relevance.
8) **Path planning in robotics:** A* can be used to plan the path of a robot from one point to another, considering various constraints, such as avoiding obstacles or minimizing energy consumption.
9) **Game AI:** A* is also used to make intelligent decisions for non-player characters (NPCs), such as determining the best way to reach an objective or coordinate movements in a team-based game.
10) These are just a few examples of how the A* search algorithm finds applications in various areas of artificial intelligence. Its flexibility, efficiency, and optimization make it a valuable tool for many problems.

2.10.4. A* Search Algorithm Complexity

The A* (pronounced "A-star") search algorithm is a popular and widely used graph traversal and path search algorithm in artificial intelligence. Finding the shortest path between two nodes in a graph or grid-based environment is usually common. The algorithm combines Dijkstra's and greedy best-first search elements to explore the search space while ensuring optimality efficiently. Several factors determine the complexity of the A* search algorithm.

Graph size (nodes and edges): A graph's number of nodes and edges greatly affects the algorithm's complexity. More nodes and edges mean more possible options to explore, which can increase the execution time of the algorithm.

Heuristic function: A* uses a heuristic function (often denoted h(n)) to estimate the cost from the current node to the destination node. The precision of this heuristic greatly affects the efficiency of the A* search. A good heuristic can help guide the search to a goal more quickly, while a bad heuristic can lead to unnecessary searching.

Data Structures: A* maintains two main data structures: an open list (priority queue) and a closed list (or visited pool). The efficiency of these data structures, along with the chosen implementation (e.g., priority queue binary heaps), affects the algorithm's performance.

Branch factor: The average number of followers for each node affects the number of nodes expanded during the search. A higher branching factor can lead to more exploration, which increases

Optimality and completeness: A* guarantees both optimality (finding the shortest path) and completeness (finding a solution that exists). However, this guarantee comes with a trade-off in terms of computational complexity, as the algorithm must explore different paths for optimal performance. Regarding time complexity, the chosen heuristic function affects A* in the worst case. With an accepted heuristic (which never overestimates the true cost of reaching the goal), A* expands the fewest nodes among the optimization algorithms. The worst-case time complexity of A * is exponential in the worst-case O(b ^ d), where "b" is the effective branching factor (average number of followers per node) and "d" is the optimal

2.11. AO* Search: (And-Or) Graph

Best-first search is what the AO* algorithm does. The AO* method **divides** any given difficult **problem into a smaller group** of problems that are then resolved **using the AND-OR** graph concept. AND OR graphs are specialized graphs that are used in problems that can be divided into smaller problems. The AND side of the graph represents a set of tasks that must be completed to achieve the main goal, while the OR side of the graph represents different methods for accomplishing the same main goal.

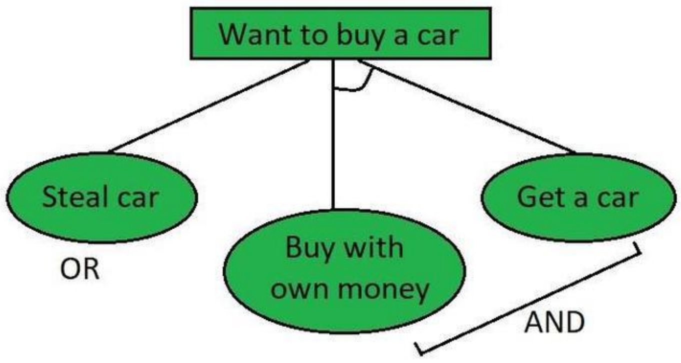


Figure 2.11 AO Tree*

In the above figure, the **buying of a car** may be broken down into smaller problems or tasks that can be accomplished **to achieve the main goal** in the above figure, which is an example of a simple AND-OR graph. The other task is to either steal a car that will help us accomplish the main goal or use your own money to purchase a car that will accomplish the main goal. The AND symbol is used to indicate the AND part of the graphs, which refers to the need that all subproblems containing the AND to be resolved before the preceding node or issue may be finished.

The start state and the target state are already known in the knowledge-based search strategy known as the **AO* algorithm**, and the best path is identified by heuristics. The informed search technique considerably reduces the algorithm's **time complexity**. The AO* algorithm is far more effective in searching AND-OR trees **than** the A* algorithm.

Working of AO* algorithm:

The evaluation function in AO* looks like this:

f(n)= g(n) + h(n)

f(n)= Actual cost + Estimated cost

here,

f(n)= The actual cost of traversal.

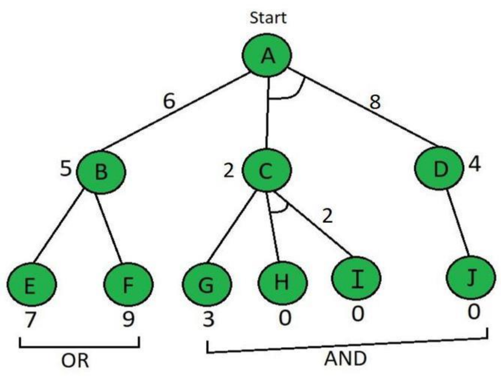
g(n)= The cost from the initial node to the current node.

h(n) = Estimated cost from the current node to the goal state.

Difference between the A* Algorithm and AO* algorithm

- A* algorithm and AO* algorithm both works on the **best first search**.
- They are both **informed search** and works on given heuristics values.
- **A*** always **gives** the **optimal solution** but AO* doesn't guarantee to give the optimal solution.
- Once AO* got a solution **doesn't explore** all possible paths but A* explores all paths.
- When compared to the A* algorithm, the AO* algorithm uses **less memory**.
- opposite to the A* algorithm, the AO* algorithm cannot go into an endless **loop**.

Example:



:

Figure 2.12 AO Algorithm*

Here in the above example below the Node which is given is the heuristic value i.e **h(n)**. Edge length is considered as **1**.

Step 1

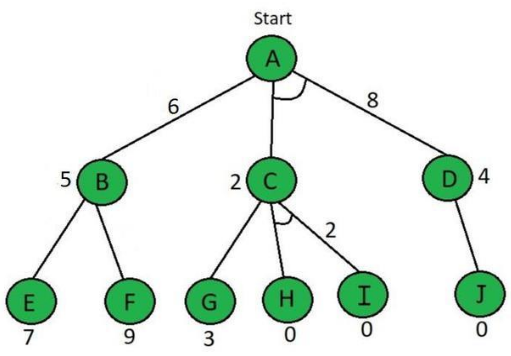


Figure 2.13 Step 1 AO Tree*

With help of **f(n) = g(n) + h(n)** evaluation function,

Start from node A

F(A→B) = g(B) + h(B)

= 1+5 …..here g(n)= 1 is taken by default for path cost

= 6

F(A→C+D)= g(C) +h(C) + g(D) +h(D)

= 1+ 2+1+4 ……..here we have added C & D

Because they are in AND

= 8

So, by calculation A→B path is chosen which is the minimum path , i.e f(A→B)

Step 2:-

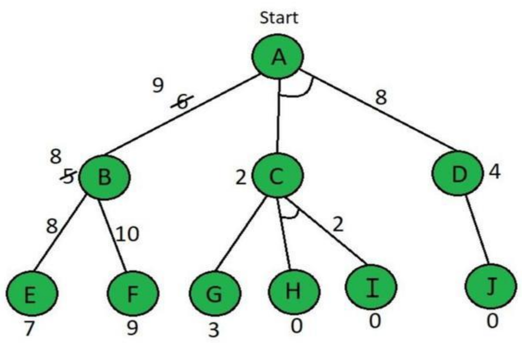


Figure 2.14 Step2 AO Tree*

According to the answer of step 1, explore node B

Here the value of E & F are calculated as follows,

f(B→E) = g(E) + h(E)

f(B→E) = 1+7

=8

f(B→F) = g(F) + h(F)

= 1 + 9

= 10

So, by above calculation B→ E path is chosen which is minimum path,

i.e f(B→E)

because B's heuristic value is different from its actual value the heuristic is updated and the minimum cost path is selected. The minimum value in our situation is 8.

Therefore, the heuristic for A must be updated due to the change in B's heuristic. So we need to calculate it again

f(A→B) = g(B) + updated h(B)

= 1+ 8

= 9

We have updated all values of the Tree.

By comparing f(A→B) & f(A→ C+D)

f(A→ C+D) is shown to be smaller. i.e 8 < 9

Now explore f(A→ C+D) so, the current node is C

F(C→G)= g(G) + h(G)

F(C→G)= 1+ 3

= 4

f(C→ H+I) = g(H) + h(H) + g(I) + h(I)

f(C→ H+I)= 1+ 0 +1+0 =2

f(C→ H+I) is selected as the path with the lowest cost and the heuristic is also left unchanged. Because it matches the actual cost. Paths H& I are solved because the heuristic for those path is 0.

But path A→D needs to be calculated because it has an AND.

f(D→J) = g(J) + h(J)

f(D→J) = 1+ 0 = 1

The heuristic of node D needs to be updated to 1.

f(A→ C+D) = g(C) + h(C) + g(D) + h(D)

= 1+ 2+ 1+1 = 5

As we can see that path f(A→ C+D) is get solved and this Tree has become a solved tree now.

Advantages:

- It is an optimal algorithm.
- If traverse according to the ordering of nodes. It can be used for both OR and AND graph.

Disadvantages:

- Sometimes for unsolvable nodes, it can't find the optimal path. Its complexity is than other algorithms.

2.12. CONSTRAINT SATISFACTION:-

Many problems in AI can be considered as problems of constraint satisfaction, in which the goalstate satisfies a given set of constraint. constraint satisfaction problems can be solved by using any of the search strategies. The general form of the constraint satisfaction procedure is as follows:

Until a complete solution is found or until all paths have led to lead ends, do

1. select an unexpanded node of the search graph.
2. Apply the constraint inference rules to the selected node to generate all possible new constraints.
3. If the set of constraints contains a contradiction, then report that this path is a dead end.
4. If the set of constraints describes a complete solution then report success.
5. If neither a constraint nor a complete solution has been found then apply the rules to generate new partial solutions. Insert these partial solutions into the search graph.

2.12.1. CONSTRAINTS:-

1. no two digit can be assigned to same letter.
2. only single digit number can be assign to a letter.
3. no two letters can be assigned same digit.
4. Assumption can be made at various levels such that they do not contradict each other.
5. The problem can be decomposed into secured constraints. A constraint satisfaction approachmay be used.
6. Any of search techniques may be used.
7. Backtracking may be performed as applicable us applied search techniques.

There are mainly three basic components in the constraint satisfaction problem:

Variables: The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

Domains: The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.

Constraints: The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Constraint Satisfaction Problems (CSP) representation:

- The finite set of variables V_1, V_2, V_3 ……………..V_n.
- Non-empty domain for every single variable D_1, D_2, D_3 …………..D_n.
- The finite set of constraints C_1, C_2 …….…, Cm.
- where each constraint C_i restricts the possible values for variables,
- e.g., $V_1 \neq V_2$
- Each constraint C_i is a pair <scope, relation>
- Example: <(V_1, V_2), V_1 not equal to V_2>
- Scope = set of variables that participate in constraint.
- Relation = list of valid variable value combinations.
- There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.

2.12.2. Constraint Satisfaction Problems (CSP) algorithms:

- The **backtracking algorithm** is a depth-first search algorithm that methodically investigates the search space of potential solutions up until a solution is discovered that satisfies all the restrictions. The method begins by choosing a variable and giving it a value before repeatedly attempting to give values to the other variables. The method returns to the prior variable and tries a different value if at any time a variable cannot be given a value that fulfills the requirements. Once all assignments have been tried or a solution that satisfies all constraints has been discovered, the algorithm ends.
- The **forward-checking algorithm** is a variation of the backtracking algorithm that condenses the search space using a type of local consistency. For each unassigned variable, the method keeps a list of remaining values and applies local constraints to eliminate inconsistent values from these sets. The algorithm examines a variable's neighbors after it is given a value to see whether any of its remaining values become inconsistent and removes them from the sets if they do. The algorithm goes backward if, after forward checking, a variable has no more values.
- Algorithms for **propagating constraints** are a class that uses local consistency and inference to condense the search space. These algorithms operate by propagating restrictions between variables and removing inconsistent values from the variable domains using the information obtained.

2.12.3. Real-world Constraint Satisfaction Problems (CSP):

Scheduling: A fundamental CSP problem is how to efficiently and effectively schedule resources like personnel, equipment, and facilities. The constraints in this domain specify the availability and capacity of each resource, whereas the variables indicate the time slots or resources.

Vehicle routing: Another example of a CSP problem is the issue of minimizing travel time or distance by optimizing a fleet of vehicles' routes. In this domain, the constraints specify each vehicle's capacity, delivery locations, and time windows, while the variables indicate the routes taken by the vehicles.

Assignment: Another typical CSP issue is how to optimally assign assignments or jobs to humans or machines. In this field, the variables stand in for the tasks, while the constraints specify the knowledge, capacity, and workload of each person or machine.

Sudoku: The well-known puzzle game Sudoku can be modeled as a CSP problem, where the variables stand in for the grid's cells and the constraints specify the game's rules, such as prohibiting the repetition of the same number in a row, column, or area.

Constraint-based image segmentation: The segmentation of an image into areas with various qualities (such as color, texture, or shape) can be treated as a CSP issue in computer vision, where the variables represent the regions and the constraints specify how similar or unlike neighboring regions are to one another.

Constraint Satisfaction Problems (CSP) benefits:

- conventional representation patterns
- generic successor and goal functions
- Standard heuristics (no domain-specific expertise).

3. Knowledge Representation

3.1. Knowledge Representation:-

For the purpose of solving complex problems c\encountered in AI, we need both a large amount of knowledge and some mechanism for manipulating that knowledge to create solutions to new problems. A variety of ways of representing knowledge (facts) have been exploited in AI programs. In all variety of knowledge representations , we deal with two kinds of entities.

A. Facts: Truths in some relevant world. These are the things we want to represent.
B. Representations of facts in some chosen formalism .

these are things we will actually be able to manipulate.

One way to think of structuring these entities is at two levels : (a) the knowledge level, at whichfacts are described, and (b) the symbol level, at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

The facts and representations are linked with two-way mappings. This link is called representation mappings. The forward representation mapping maps from facts to representations. The backward representation mapping goes the other way, from representationsto facts.

One common representation is natural language (particularly English) sentences. Regardless of the representation for facts we use in a program , we may also need to be concerned with an English representation of those facts in order to facilitate getting information into and out of the system. We need mapping functions from English sentences to the representation we actually useand from it back to sentences.

3.1.1.Representations and Mappings

In order to solve complex problems encountered in artificial intelligence, one needs both a large amount of knowledge and some mechanism for manipulating that knowledge to create solutions.

- Knowledge and Representation are two distinct entities. They play central but distinguishable roles in the intelligent system.
- Knowledge is a description of the world. It determines a system’s competence by what itknows.
- Moreover, Representation is the way knowledge is encoded. It defines a system’s performance in doing something.
- Different types of knowledge require different kinds of representation.

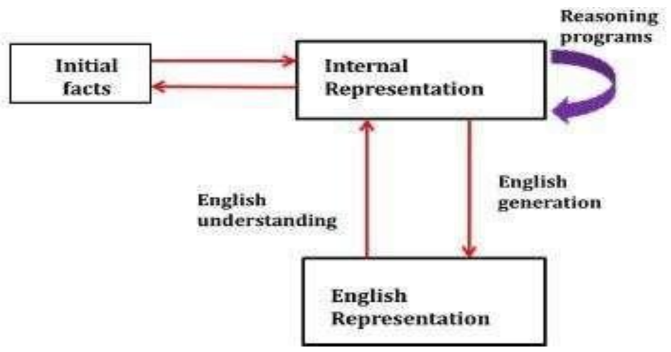


Figure 3.1 Knowledge Representation

Fig: Mapping between Facts and Representations

The Knowledge Representation models/mechanisms are often based on:

- Logic
- Rules
- Frames
- Semantic Net

Knowledge is categorized into two major types:

1. Tacit corresponds to “informal” or “implicit“
    - Exists within a human being;
    - It is embodied.
    - Difficult to articulate formally.
    - Difficult to communicate or share.
    - Moreover, Hard to steal or copy.
    - Drawn from experience, action, subjective insight
2. Explicit formal type of knowledge, Explicit
    - Explicit knowledge
    - Exists outside a human being;
    - It is embedded.
    - Can be articulated formally.
    - Also, Can be shared, copied, processed and stored.
    - So, Easy to steal or copy
    - Drawn from the artifact of some type as a principle, procedure, process, concepts.

A variety of ways of representing knowledge have been exploited in AI programs.

There are two different kinds of entities, we are dealing with.

1. Facts: Truth in some relevant world. Things we want to represent.
2. Also, Representation of facts in some chosen formalism. Things we will actually be ableto manipulate.

These entities structured at two levels:

1. The knowledge level, at which facts described.
2. Moreover, The symbol level, at which representation of objects defined in terms of symbols that can manipulate by programs

3.1.2.Framework of Knowledge Representation

- The computer requires a well-defined problem description to process and provide a well-defined acceptable solution.
- Moreover, To collect fragments of knowledge we need first to formulate a description inour spoken language and then represent it in formal language so that computer can understand.
- Also, The computer can then use an algorithm to compute an answer.So, This process

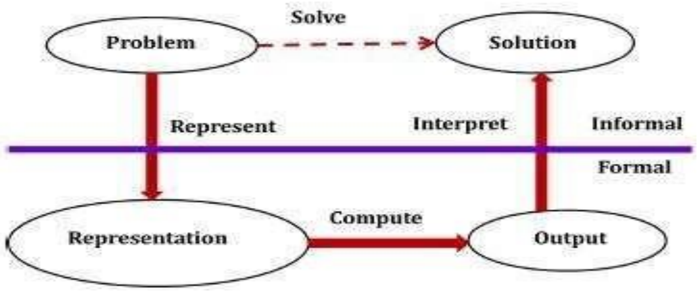


Figure 3.2 Knowledge Representation Mapping

illustrated as,

The steps are:

- The informal formalism of the problem takes place first.
- It then represented formally and the computer produces an output.
- This output can then represented in an informally described solution that user understandsor checks for consistency.

The Problem solving requires,

- Formal knowledge representation, and
- Moreover, Conversion of informal knowledge to a formal knowledge that is the conversion of implicit knowledge to explicit knowledge.

Mapping between Facts and Representation

- Knowledge is a collection of facts from some domain.
- Also, We need a representation of "facts" that can manipulate by a program.
- Moreover, Normal English is insufficient, too hard currently for a computer program todraw inferences in natural languages.
- Thus some symbolic representation is necessary.

A good knowledge representation enables fast and accurate access to knowledge and understanding of the content.

A knowledge representation system should have following properties.

1. Representational Adequacy
   - The ability to represent all kinds of knowledge that are needed in that domain.
2. Inferential Adequacy
   - Also, The ability to manipulate the representational structures to derive new structures corresponding to new knowledge inferred from old.
3. Inferential Efficiency
   - The ability to incorporate additional information into the knowledge structure that can be used to focus the attention of the inference mechanisms in the most promising direction.
4. Acquisitional Efficiency
   - Moreover, The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

*3.1.3.*Relational Knowledge

- The simplest way to represent declarative facts is a set of relations of the same sort usedin the database system.
- Provides a framework to compare two objects based on equivalent attributes. o Any instance in which two different objects are compared is a relational type of knowledge.
- The table below shows a simple way to store facts.
  - Also, The facts about a set of objects are put systematically in columns.
  - This representation provides little opportunity for inference.

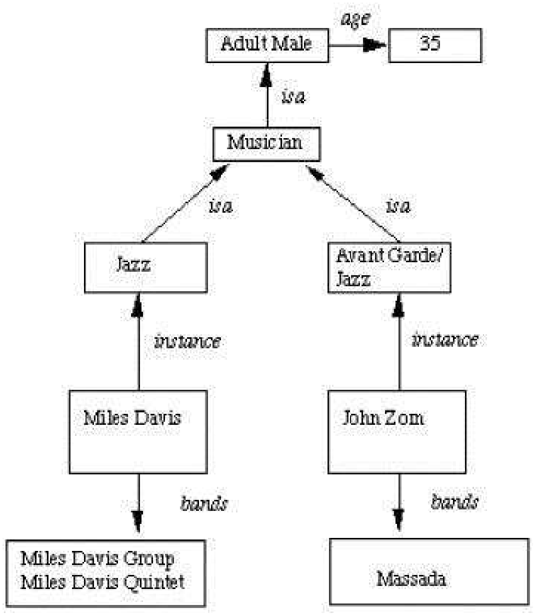
Player	Height	Weight	Bats-Throws
Aaron	6-0	180	Right-Right
Mays	5-10	170	Right-Right
Ruth	6-2	215	Left-Left
Williams	6-3	205	Left-Right

Table 3.1 Relational Knowledge Table

- Given the facts, it is not possible to answer a simple question such as: “Who is the heaviest player?”
- Also, But if a procedure for finding the heaviest player is provided, then these facts willenable that procedure to compute an answer.
- Moreover, We can ask things like who “bats – left” and “throws – right”.

*3.1.4.*Inheritable Knowledge

- Here the knowledge elements inherit attributes from their parents.
- The knowledge embodied in the design hierarchies found in the functional, physical andprocess domains.
- Within the hierarchy, elements inherit attributes from their parents, but in many cases, notall attributes of the parent elements prescribed to the child elements.
- Also, The inheritance is a powerful form of inference, but not adequate.
- Moreover, The basic KR (Knowledge Representation) needs to augment with inferencemechanism.
- Property inheritance: The objects or elements of specific classes inherit attributes and values from more general classes.
- So, The classes organized in a generalized hierarchy.



- Boxed nodes — objects and values of attributes of objects.
- Arrows — the point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection offrames.

The steps to retrieve a value for an attribute of an instance object:

1. Find the object in the knowledge base
2. If there is a value for the attribute report it
3. Otherwise look for a value of an instance, if none fail
4. Also, Go to that node and find a value for the attribute and then report it
5. Otherwise, search through using is until a value is found for the attribute.

*3.1.5.*Inferential Knowledge

- This knowledge generates new information from the given information.

- This new information does not require further data gathering form source but does require analysis of the given information to generate new knowledge.
- Example: given a set of relations and values, one may infer other values or relations. Apredicate logic (a mathematical deduction) used to infer from a set of attributes. Moreover, Inference through predicate logic uses a set of logical operations to relate individual data.
- Represent knowledge as formal logic: All dogs have tails ∀x: dog(x) → hastail(x)

Advantages:

- A set of strict rules.
- Can use to derive more facts.
- Also, Truths of new statements can be verified.
- Guaranteed correctness.
- So, Many inference procedures available to implement standard rules of logic popular inAI systems. e.g Automated theorem proving.

*3.1.6.*Procedural Knowledge

A representation in which the control information, to use the knowledge, embedded in the knowledge itself. For example, computer programs, directions, and recipes; these indicate specific use or implementation;

Moreover, Knowledge encoded in some procedures, small programs that know how to do specific things, how to proceed.

Advantages:

- Heuristic or domain-specific knowledge can represent.
- Moreover, Extended logical inferences, such as default reasoning facilitated.
- Also, Side effects of actions may model. Some rules may become false in time. Keeping track of this in large systems may be tricky.

Disadvantages:

- Completeness — not all cases may represent.
- Consistency — not all deductions may be correct. e.g If we know that Fred is a bird we might deduce that Fred can fly. Later we might discover that Fred is an emu.
- Modularity sacrificed.
- Cumber some control information.

3.2. Using Predicate Logic

Representation of Simple Facts in Logic

Propositional logic is useful because it is simple to deal with and a decision procedure for it exists.

Also, In order to draw conclusions, facts are represented in a more convenient way as,

1. Marcus is a man.
   - man(Marcus)
2. Plato is a man.
   - man(Plato)

3. All men are mortal.

   - mortal(men)

But propositional logic fails to capture the relationship between an individual being a man and that individual being a mortal.

- How can these sentences be represented so that we can infer the third sentence from thefirst two?
- Also, Propositional logic commits only to the existence of facts that may or may not bethe case in the world being represented.
- Moreover, It has a simple syntax and simple semantics. It suffices to illustrate the processof inference.
- Propositional logic quickly becomes impractical, even for very small worlds.

***3.2.1.*Predicate logic**

First-order Predicate logic (FOPL) models the world in terms of

- Objects, which are things with individual identities
- Properties of objects that distinguish them from other objects
- Relations that hold among sets of objects
- Functions, which are a subset of relations where there is only one "value" for any given"input"

First-order Predicate logic (FOPL) provides

- Constants: a, b, dog33. Name a specific object.
- Variables: X, Y. Refer to an object without naming it.
- Functions: Mapping from objects to objects.
- Terms: Refer to objects
- Atomic Sentences: in(dad-of(X), food6) Can be true or false, Correspond to propositionalsymbols P, Q.

A well-formed formula (wff) is a sentence containing no "free" variables. So, That is, all variables are "bound" by universal or existential quantifiers.

(∀x)P(x, y) has x bound as a universally quantified variable, but y is free.

Quantifiers

Universal quantification

- (∀x)P(x) means that P holds for all values of x in the domain associated with that variable
- E.g., (∀x) dolphin(x) → mammal(x)Existential quantification
- (∃ x)P(x) means that P holds for some value of x in the domain associated with thatvariable
- E.g., (∃ x) mammal(x) Ʌ lays-eggs(x)

Also, Consider the following example that shows the use of predicate logic as a way of representing knowledge.

3. Marcus was a man.
4. Marcus was a Pompeian.
5. All Pompeians were Romans.
6. Caesar was a ruler.
7. Also, All Pompeians were either loyal to Caesar or hated him.
8. Everyone is loyal to someone.
9. People only try to assassinate rulers they are not loyal to.
10. Marcus tried to assassinate Caesar.

The facts described by these sentences can be represented as a set of well-formed formulas (*wffs*)as follows:

1. Marcus was a man.
   - man(Marcus)
2. Marcus was a Pompeian.
   - Pompeian(Marcus)
3. All Pompeians were Romans.
   - ∀x: Pompeian(x) → Roman(x)
4. Caesar was a ruler.
   - ruler(Caesar)
5. All Pompeians were either loyal to Caesar or hated him.
   - inclusive-or
   - ∀x: Roman(x) → loyal to(x, Caesar) ∨ hate(x, Caesar)
   - exclusive-or
   - ∀x: Roman(x) → (loyal to(x, Caesar) Ʌ¬ hate(x, Caesar)) ∨
   - (¬loyal to(x, Caesar) Ʌ hate(x, Caesar))
6. Everyone is loyal to someone.
   - ∀x: ∃y: loyal to(x, y)
7. People only try to assassinate rulers they are not loyal to.
   - ∀x: ∀y: person(x) Ʌ ruler(y) Ʌ try assassinate(x, y)
   - →¬loyal to(x, y)
8. Marcus tried to assassinate Caesar.
   - Try assassinate(Marcus,Caesar)

Now suppose if we want to use these statements to answer the question: **Was Marcus loyal to Caesar?**

Also, Now let’s try to produce a formal proof, reasoning backward from the desired goal: ¬ Ioyal to(Marcus, Caesar)

In order to prove the goal, we need to use the rules of inference to transform it into another goal (or possibly a set of goals) that can, in turn, transformed, and so on, until there are no unsatisfied goals remaining.

$$\begin{tikzcd}
\neg\text{loyal to(Marcus, Caesar)} \\
\text{Person(Marcus)}\wedge\text{ ruler(Caesar)}\wedge\text{ tryassassinate(Marcus,Caesar)} \arrow[u] \\
\text{Person(Marcus) tryassassinate (Marcus,Caesar)} \arrow[u] \\
\text{Person(Marcus)} \arrow[u]
\end{tikzcd}$$

Figure: An attempt to prove ¬loyal to(Marcus, Caesar).

- The problem is that, although we know that Marcus was a man, we do not have any wayto conclude from that that Marcus was a person. Also, We need to add the representation of another fact to our system, namely***: ∀ man(x) → person(x)***
- Now we can satisfy the last goal and produce a proof that Marcus was not loyal toCaesar.
- Moreover, From this simple example, we see that three important issues must be addressed in the process of converting English sentences into logical statements and then using those statements to deduce new ones:

1. Many English sentences are ambiguous (for example, 5, 6, and 7 above).Choosing the correct interpretation may be difficult.
2. Also, There is often a choice of how to represent the knowledge. Simple representations are desirable, but they may exclude certain kinds of reasoning.
3. Similalry, Even in very simple situations, a set of sentences is unlikely to contain all the information necessary to reason about the topic at hand. In order to be ableto use a set of statements effectively. Moreover, It is usually necessary to have access to another set of statements that represent facts that people consider too obvious to mention.

3.3. Resolutional Propositional

1. Convert all the propositions of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made:
    - Select two clauses. Call these the parent clauses.
    - Resolve them together. The resulting clause, called the resolvent, will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals L and ¬ *L* such that one of the parent clauses contains *L* and the other contains ¬*L,* then select one such pair and eliminate both *L* and ¬ *L* from the resolvent.
    - If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of classes available to the procedure.

3.3.1.The Unification Algorithm

- In propositional logic, it is easy to determine that two literals cannot both be true at thesame time.
- Simply look for L and ¬L in predicate logic, this matching process is more complicatedsince the arguments of the predicates must be considered.
- For example, man(John) and ¬man(John) is a contradiction, while the man(John) and
- ¬man(Spot) is not.
- Thus, in order to determine contradictions, we need a matching procedure that compares two literals and discovers whether there exists a set of substitutions that makes them identical.
- There is a straightforward recursive procedure, called the unification algorithm, that doesit.

Algorithm: Unify(L1, L2)

1. If L1 or L2 are both variables or constants, then:
    - If L1 and L2 are identical, then return NIL.
    - Else if L1 is a variable, then if L1 occurs in L2 then return {FAIL}, else return(L2/L1).
    - Also, Else if L2 is a variable, then if L2 occurs in L1 then return {FAIL}, else return (L1/L2). d. Else return {FAIL}.
2. If the initial predicate symbols in L1 and L2 are not identical, then return {FAIL}.
3. If LI and L2 have a different number of arguments, then return {FAIL}.
4. Set SUBST to NIL. (At the end of this procedure, SUBST will contain all the substitutions used to unify L1 and L2.)
5. For I ← 1 to the number of arguments in L1 :
    - Call Unify with the ith argument of L1 and the ith argument of L2, putting the result in S.
    - If S contains FAIL then return {FAIL}.
    - If S is not equal to NIL then:

    a) Apply S to the remainder of both L1 and L2.
    b) SUBST: = APPEND(S, SUBST).
6. Return SUBST.

3.3.2. Resolution in Predicate Logic

We can now state the resolution algorithm for predicate logic as follows, assuming a set of givenstatements F and a statement to be proved P:

Algorithm: Resolution

1. Convert all the statements of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in 1.
3. Repeat until a contradiction found, no progress can make, or a predetermined amount ofeffort has expanded.
    a) Select two clauses. Call these the parent clauses.
    b) Resolve them together. The resolvent will the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals T1 and ¬T2 such that one of the parent clauses contains T2 and the other contains T1 and if T1 and T2 are unifiable, thenneither T1 nor T2 should appear in the resolvent. We call T1 and T2 Complementary literals. Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complementary literals, only one pair should omit from the resolvent.
    c) If the resolvent is an empty clause, then a contradiction has found. Moreover, If it is not, then add it to the set of classes available to the procedure.

3.3.3. Resolution Procedure

- Resolution is a procedure, which gains its efficiency from the fact that it operates on statements that have been converted to a very convenient standard form.
- Resolution produces proofs by refutation.
- In other words, to prove a statement (i.e., to show that it is valid), resolution attempts to show that the negation of the statement produces a contradiction with the known statements (i.e., that it is unsatisfiable).
- The resolution procedure is a simple iterative process: at each step, two clauses, called the parent clauses, are compared (resolved), resulting in a new clause that has inferred from them. The new clause represents ways that the two parent clauses interact with eachother. Suppose that there are two clauses in the system:

  winter V summer

      ¬ winter V cold

- Now we observe that precisely one of winter and ¬ winter will be true at any point.
- If winter is true, then cold must be true to guarantee the truth of the second clause. If ¬winter is true, then summer must be true to guarantee the truth of the first clause.
- Thus we see that from these two clauses we can deduce **summer V cold**
- This is the deduction that the resolution procedure will make.
- Resolution operates by taking two clauses that each contains the same literal, in this example, **winter**.
- Moreover, The literal must occur in the positive form in one clause and in negative form in the other. The resolvent obtained by combining all of the literals of the two parent clauses except the ones that cancel.
- If the clause that produced is the empty clause, then a contradiction has found.

- For example, the two clauses
  winter

  ¬ winter

  will produce the empty clause.

3.4. Evolution Frames

- As seen in the previous example, there are certain problems which are difficult to solvewith Semantic Nets.
- Although there is no clear distinction between a semantic net and frame system, more structured the system is, more likely it is to be termed as a frame system.
- A frame is a collection of attributes (called slots) and associated values that describe some entities in the world. Sometimes a frame describes an entity in some absolute sense;
- Sometimes it represents the entity from a particular point of view only.
- A single frame taken alone is rarely useful; we build frame systems out of collections of frames that connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame.

Frames as Sets and Instances

- The set theory is a good basis for understanding frame systems.
- Each frame represents either a class (a set) or an instance (an element of class)
- Both *isa* and *instance* relations have inverse attributes, which we call subclasses & all instances.
- As a class represents a set, there are 2 kinds of attributes that can be associated with it.
  1. Its own attributes &
  2. Attributes that are to be inherited by each element of the set.

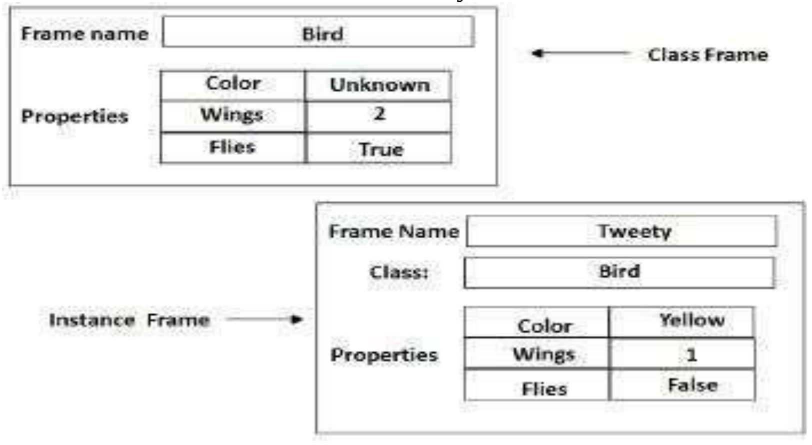


Figure 3.3Frames as Sets and Instances

- Sometimes, the difference between a set and an individual instance may not be clear.
- Example: Team India is an instance of the class of Cricket Teams and can also think of asthe set of players.
- Now the problem is if we present Team India as a subclass of Cricket teams, then Indianplayers automatically become part of all the teams, which is not true.
- So, we can make Team India a subclass of class called Cricket Players.
- To do this we need to differentiate between regular classes and meta-classes.
- Regular Classes are those whose elements are individual entities whereas Meta-classesare those special classes whose elements are themselves, classes.

- The most basic meta-class is the class *CLASS*.
- It represents the set of all classes.
- All classes are instances of it, either directly or through one of its subclasses.
- The class *CLASS* introduces the attribute cardinality, which is to inherited by all instancesof CLASS. Cardinality stands for the number.

Other ways of Relating Classes to Each Other

  - We have discussed that a class1 can be a subset of class2.
  - If Class2 is a meta-class then Class1 can be an instance of Class2.
  - Another way is the mutually-disjoint-with relationship, which relates a class to one ormore other classes that guaranteed to have no elements in common with it.
  - Another one is, *is-covered-by* which relates a class to a set of subclasses, the union ofwhich is equal to it.
  - If a class is-covered-by a set S of mutually disjoint classes, then S called a partition of theclass.

Slots as Full-Fledged Objects (Frames)

Till now we have used attributes as slots, but now we will represent attributes explicitly and describe their properties.

Some of the properties we would like to be able to represent and use in reasoning include,

- The class to which the attribute can attach.
- Constraints on either the type or the value of the attribute.
- A default value for the attribute. Rules for inheriting values for the attribute.
- To be able to represent these attributes of attributes, we need to describe attributes (slots)as frames.
- These frames will organize into an *isa* hierarchy, just as any other frames, and that hierarchy can then used to support inheritance of values for attributes of slots.

3.5. Semantic Nets

- Inheritance property can represent using **isa** and **instance**
- Monotonic Inheritance can perform substantially more efficiently with such structures than with pure logic, and non-monotonic inheritance is also easily supported.
- The reason that makes Inheritance easy is that the knowledge in slot and filler systems isstructured as a set of entities and their attributes.

These structures turn out to be useful as,

- It indexes assertions by the entities they describe. As a result, retrieving the value for anattribute of an entity is fast.
- Moreover, It makes easy to describe properties of relations. To do this in a purely logicalsystem requires higher-order mechanisms.
- It is a form of object-oriented programming and has the advantages that such systems normally include modularity and ease of viewing by people.

Here we would describe two views of this kind of structure – Semantic Nets & Frames.

Semantic Nets

- There are different approaches to knowledge representation include semantic net, frames,and script.
- The semantic net describes both objects and events.

- In a semantic net, information represented as a set of nodes connected to each other by aset of labeled arcs, which represents relationships among the nodes.
- It is a directed graph consisting of vertices which represent concepts and edges which represent semantic relations between the concepts.
- It is also known as associative net due to the association of one node with other.
- The main idea is that the meaning of the concept comes from the ways in which it connected to other concepts.
- We can use inheritance to derive additional relations.

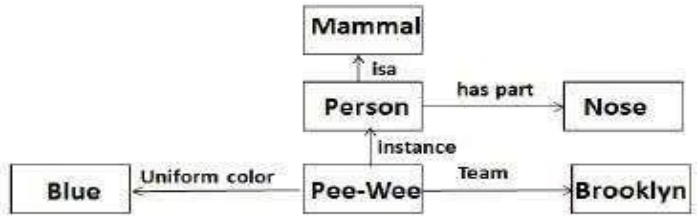


Figure 3.4 A Semantic Network

Intersection Search Semantic Nets

- We try to find relationships among objects by spreading activation out from each of twonodes. And seeing where the activation meets.
- Using this we can answer the questions like, what is the relation between India and Blue.
- It takes advantage of the entity-based organization of knowledge that slot and filler representation provides.

Representing Non-binary Predicates Semantic Nets

- Simple binary predicates like isa(Person, Mammal) can represent easily by semantic nets but other non-binary predicates can also represent by using general-purpose predicates such as *isa* and *instance*.
- Three or even more place predicates can also convert to a binary form by creating one new object representing the entire predicate statement and then introducing binary predicates to describe a relationship to this new object.

3.6. Conceptual Dependency

Introduction to Strong Slot and Filler Structures

- The main problem with semantic networks and frames is that they lack formality; there isno specific guideline on how to use the representations.
- In frame when things change, we need to modify all frames that are relevant – this can betime-consuming.
- Strong slot and filler structures typically represent links between objects according to more rigid rules, specific notions of what types of object and relations between them areprovided and represent knowledge about common situations.
- Moreover, We have types of strong slot and filler structures:
    1. Conceptual Dependency (CD)
    2. Scripts
    3. Cyc

Conceptual Dependency originally developed to represent knowledge acquired from natural language input.

The goals of this theory are:

To help in the drawing of the inference from sentences.

- To be independent of the words used in the original input.
- That is to say: For any 2 (or more) sentences that are identical in meaning there should beonly one representation of that meaning.

Moreover, It has used by many programs that portend to understand English (MARGIE, SAM,PAM).

Conceptual Dependency (CD) provides:

- A structure into which nodes representing information can be placed.
- Also, A specific set of primitives.
- A given level of granularity.

Sentences are represented as a series of diagrams depicting actions using both abstract and realphysical situations.

- The agent and the objects represented.
- Moreover, The actions are built up from a set of primitive acts which can modify by tense.

CD is based on events and actions. Every event (if applicable) has:

- an ACTOR o an ACTION performed by the Actor
- Also, an OBJECT that the action performs on
- A DIRECTION in which that action is oriented

These are represented as slots and fillers. In English sentences, many of these attributes left out.

3.6.1.A Simple Conceptual Dependency Representation

For the sentences, "I have a book to the man" CD representation is as follows:

$$\begin{tikzcd}
 & & & & \text{man} \\
I \arrow[r, Leftrightarrow, "P"] & \textbf{ATRANS} & \textbf{book} \arrow[l, "O"'] & {} \arrow[l, "R"'] \arrow[ur, "\text{to}"] & \\
 & & & & I \arrow[ul, "\text{From}"]
\end{tikzcd}$$

Where the symbols have the following meaning.

Arrows indicate directions of dependency.

Moreover, The double arrow indicates the two-way link between actor and action.

O — for the object case relation

R – for the recipient case relation

P – for past tense

D – destination

Primitive Acts of Conceptual Dependency Theory

ATRANS

- Transfer of an abstract relationship (i.e. give)

PTRANS

- Transfer of the physical location of an object (e.g., go)

PROPEL

- Also, Application of physical force to an object (e.g. push)

MOVE

- Moreover, Movement of a body part by its owner (e.g. kick)

GRASP

- Grasping of an object by an action (e.g. throw)

INGEST

- Ingesting of an object by an animal (e.g. eat)

EXPEL

- Expulsion of something from the body of an animal(e.g. cry).

MTRANS

- Transfer of mental information (e.g. tell)

MBUILD

- Building new information out of old (e.g decide)

SPEAK

- Producing of sounds (e.g. say)

ATTEND

- Focusing of a sense organ toward a stimulus (e.g. listen)

There are four conceptual categories. These are,

ACT

- Actions {one of the CD primitives}

PP

- Also, Objects {picture producers}

AA

- Modifiers of actions {action aiders}

PA

- Modifiers of PP's {picture aiders}

Advantages of Conceptual Dependency

- Using these primitives involves fewer inference rules.
- So, Many inference rules already represented in CD structure.
- Moreover, The holes in the initial structure help to focus on the points still to established.

Disadvantages of Conceptual Dependency

- Knowledge must decompose into fairly low-level primitives.
- Impossible or difficult to find the correct set of primitives.
- Also, A lot of inference may still require.

- Representations can be complex even for relatively simple actions.
- Consider: Dave bet Frank five pounds that Wales would win the Rugby World Cup.
- Moreover, Complex representations require a lot of storage.

4. Natural Language Processing

4.1. Introduction to Natural Language Processing

- Language meant for communicating with the world.
- Also, By studying language, we can come to understand more about the world.
- If we can succeed at building computational mode of language, we will have a powerfultool for communicating with the world.
- Also, We look at how we can exploit knowledge about the world, in combination with linguistic facts, to build computational natural language systems.

Natural Language Processing (NLP) problem can divide into two tasks:

1. Processing written text, using lexical, syntactic and semantic knowledge of the languageas well as the required real-world information.
2. Processing spoken language, using all the information needed above plus additional knowledge about phonology as well as enough added information to handle the furtherambiguities that arise in speech.

4.1.1.Steps Natural Language Processing

Morphological Analysis

- Suppose we have an English interface to an operating system and the following sentencetyped: I want to print Bill’s .init file.
- The morphological analysis must do the following things:
- Pull apart the word “Bill’s” into proper noun “Bill” and the possessive suffix “’s”
- Recognize the sequence “.init” as a file extension that is functioning as an adjective in thesentence.
- This process will usually assign syntactic categories to all the words in the sentence.

Syntactic Analysis

- A syntactic analysis must exploit the results of the morphological analysis to build a structural description of the sentence.
- The goal of this process, called parsing, is to convert the flat list of words that form thesentence into a structure that defines the units that represented by that flat list.
- The important thing here is that a flat sentence has been converted into a hierarchical structure. And that the structure corresponds to meaning units when a semantic analysisperformed.
- Reference markers (set of entities) shown in the parenthesis in the parse tree.
- Each one corresponds to some entity that has mentioned in the sentence.
- These reference markers are useful later since they provide a place in which to accumulate information about the entities as we get it.

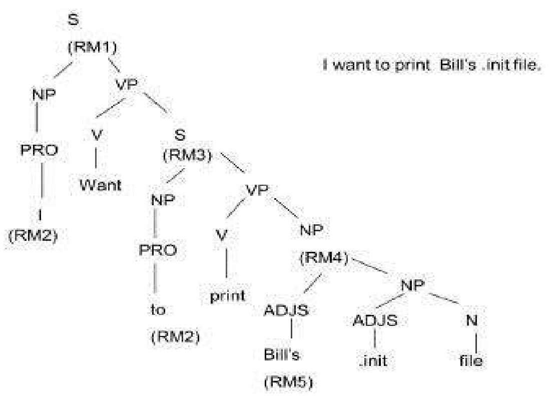


Figure 4.1 Parsing Tree

Semantic Analysis

- The semantic analysis must do two important things:

1. It must map individual words into appropriate objects in the knowledge base or database.
2. It must create the correct structures to correspond to the way the meanings of the individual words combine with each other.

Discourse Integration

- Specifically, we do not know whom the pronoun “I” or the proper noun “Bill” refers to.
- To pin down these references requires an appeal to a model of the current discourse context, from which we can learn that the current user is USER068 and that the only person named “Bill” about whom we could be talking is USER073.
- Once the correct referent for Bill known, we can also determine exactly which file referred to.

Pragmatic Analysis

- The final step toward effective understanding is to decide what to do as a result.
- One possible thing to do to record what was said as a fact and done with it.
- For some sentences, a whose intended effect is clearly declarative, that is the preciselycorrect thing to do.
- But for other sentences, including this one, the intended effect is different.
- We can discover this intended effect by applying a set of rules that characterize cooperative dialogues.
- The final step in pragmatic processing to translate, from the knowledge-based representation to a command to be executed by the system.

Syntactic Processing

- Syntactic Processing is the step in which a flat input sentence converted into a hierarchical structure that corresponds to the units of meaning in the sentence. This process called parsing.
- It plays an important role in natural language understanding systems for two reasons:
    1. Semantic processing must operate on sentence constituents. If there is no syntactic parsing step, then the semantics system must decide on its own constituents. If parsing is done, on the other hand, it constrains the number of constituents that semantics can consider.
    2. Syntactic parsing is computationally less expensive than is semantic processing. Thus it can play a significant role in reducing overall system complexity.
- Although it is often possible to extract the meaning of a sentence without using grammatical facts, it is not always possible to do so.
- Almost all the systems that are actually used have two main components:
    1. A declarative representation, called a grammar, of the syntactic facts about the language.
    2. A procedure, called parser that compares the grammar against input sentences to produce parsed structures.

Grammars and Parsers

- The most common way to represent grammars is a set of production rules.
- The first rule can read as “A sentence composed of a noun phrase followed by Verb Phrase”; the Vertical bar is OR; ε represents the empty string.
- Symbols that further expanded by rules called non-terminal symbols.

- Symbols that correspond directly to strings that must found in an input sentence calledterminal symbols.
- Grammar formalism such as this one underlies many linguistic theories, which in turn provide the basis for many natural language understanding systems.
- Pure context-free grammars are not effective for describing natural languages.
- NLPs have less in common with computer language processing systems such as compilers.
- Parsing process takes the rules of the grammar and compares them against the input sentence.
- The simplest structure to build is a Parse Tree, which simply records the rules and howthey matched.
- Every node of the parse tree corresponds either to an input word or to a non-terminal inour grammar.
- Each level in the parse tree corresponds to the application of one grammar rule.

4.2. Augmented Transition Network (ATN)

- An augmented transition network is a top-down parsing procedure that allows various kinds of knowledge to incorporated into the parsing system so it can operate efficiently.
- ATNs build on the idea of using finite state machines (Markov model) to parse sentences.
- Instead of building an automaton for a particular sentence, a collection of transition graphs built.
- A grammatically correct sentence parsed by reaching a final state in any state graph.
- Transitions between these graphs simply subroutine calls from one state to any initial state on any graph in the network.
- A sentence determined to be grammatically correct if a final state reached by the last word in the sentence.
- The ATN is similar to a finite state machine in which the class of labels that can attach tothe arcs that define the transition between states has augmented.

Arcs may label with:

- Specific words such as “in’.
- Word categories such as noun.
- Procedures that build structures that will form part of the final parse.
- Procedures that perform arbitrary tests on current input and sentence components that have identified.

Semantic Analysis

- The structures created by the syntactic analyzer assigned meanings.
- A mapping made between the syntactic structures and objects in the task domain.
- Structures for which no such mapping is possible may rejected.
- The semantic analysis must do two important things:
    1. It must map individual words into appropriate objects in the knowledge base ordatabase.
    2. It must create the correct structures to correspond to the way the meanings of theindividual words combine with each other. Semantic Analysis AI
- Producing a syntactic parse of a sentence is only the first step toward understanding it.
- We must produce a representation of the meaning of the sentence.
- Because understanding is a mapping process, we must first define the language intowhich we are trying to map.

- There is no single definitive language in which all sentence meaning can describe.
- The choice of a target language for any particular natural language understanding program must depend on what is to do with the meanings once they constructed.
- Choice of the target language in Semantic Analysis AI
    1. There are two broad families of target languages that used in NL systems, depending on the role that the natural language system playing in a larger system:
    2. When natural language considered as a phenomenon on its own, as for example when one builds a program whose goal is to read the text and then answer questions about it. A target language can design specifically to support languageprocessing.
    3. When natural language used as an interface language to another program (such as a db query system or an expert system), then the target language must legal input to that other program. Thus the design of the target language driven by the backend program.

4.2.1.Discourse and Pragmatic Processing

To understand a single sentence, it is necessary to consider the discourse and pragmatic contextin which the sentence was uttered.

There are a number of important relationships that may hold between phrases and parts of theirdiscourse contexts, including:

1. Identical entities. Consider the text:
    - Bill had a red balloon. o John wanted it.
    - The word “it” should identify as referring to the red balloon. These types of references called anaphora.
2. Parts of entities. Consider the text:
    - Sue opened the book she just bought.
    - The title page was torn.
    - The phrase “title page” should be recognized as part of the book that was just bought.
3. Parts of actions. Consider the text:
    - John went on a business trip to New York.
    - He left on an early morning flight.
    - Taking a flight should recognize as part of going on a trip.
4. Entities involved in actions. Consider the text:
    - My house was broken into last week.
    - Moreover, They took the TV and the stereo.
    - The pronoun “they” should recognize as referring to the burglars who broke into the house.
5. Elements of sets. Consider the text:
    - The decals we have in stock are stars, the moon, item and a flag.
    - I’ll take two moons.
    - Moons mean moon decals.
6. Names of individuals:
    - Dev went to the movies.
7. Causal chains
    - There was a big snow storm yesterday.

So, The schools closed today.

8. Planning sequences:
    - Sally wanted a new car

- She decided to get a job.

9. Implicit presuppositions:
   - Did Joe fail CS101?

The major focus is on using following kinds of knowledge:

- The current focus of the dialogue.
- Also, A model of each participant's current beliefs.
- Moreover, The goal-driven character of dialogue.
- The rules of conversation shared by all participants.

4.3. Statistical Natural Language Processing

Formerly, many language-processing tasks typically involved the direct hand coding of rules, which is not in general robust to natural-language variation. The machine-learning

paradigm calls instead for using statistical inference to automatically learn such rules through theanalysis of large **corpora** of typical real-world examples (a corpus (plural, "corpora") is a set of documents, possibly with human or computer annotations).

Many different classes of machine learning algorithms have been applied to natural-language processing tasks. These algorithms take as input a large set of "features" that are generated from the input data. Some of the earliest-used algorithms, such as **decision trees,** produced systems of hard if-then rules similar to the systems of hand-written rules that were then common.

Increasingly, however, research has focused on **statistical models,** which make

soft, **probabilistic** decisions based on attaching **real-valued** weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possibleanswers rather than only one, producing more reliable results when such a model is included as acomponent of a larger system.

Systems based on machine-learning algorithms have many advantages over hand-produced rules:

- The learning procedures used during machine learning automatically focus on the mostcommon cases, whereas when writing rules by hand it is often not at all obvious wherethe effort should be directed.
- Automatic learning procedures can make use of statistical inference algorithms to produce models that are robust to unfamiliar input (e.g. containing words or structures that have not been seen before) and to erroneous input (e.g. with misspelled words or words accidentally omitted). Generally, handling such input gracefully with hand-written rules—or more generally, creating systems of hand-written rules that make soft decisions—is extremely difficult, error-prone and time-consuming.
- Systems based on automatically learning the rules can be made more accurate simply by supplying more input data. However, systems based on hand-written rules can only be made more accurate by increasing the complexity of the rules, which is a much more difficult task. In particular, there is a limit to the complexity of systems based on hand- crafted rules, beyond which the systems become more and more unmanageable.

4.3.1.Spell Checking

Spell checking is one of the applications of natural language processing that impacts billions of users daily. A good introduction to spell checking can be found on Peter Norvig's webpage. The article introduces a simple 21-line spell checker implementation in Python combining simple language and error models to predict the word a user intended to type. The language model estimates how likely a given word `c` is in the language for which the spell checker is designed, this can be written as `P(C)`. The error model estimates the probability `P(w|c)` of typing the misspelled version `w` conditionally to the intention of typing the correctly spelled word `c`.The spell checker then returns word `c` corresponding to the highest value of`P(w|c)P(c)` among all possible words in the language.

4.4. Game Playing

- Charles Babbage, the nineteenth-century computer architect thought about programming his analytical engine to play chess and later of building a machine to play tic-tac-toe.
- There are two reasons that games appeared to be a good domain.
  1. They provide a structured task in which it is very easy to measure success or failure.
  2. They are easily solvable by straightforward search from the starting state to a winning position.
- The first is true is for all games bust the second is not true for all, except simplest games.
- For example, consider chess.
- The average branching factor is around 35. In an average game, each player might make50.
- So in order to examine the complete game tree, we would have to examine 35^{100}
- Thus it is clear that a simple search is not able to select even its first move during the lifetime of its opponent.
- It is clear that to improve the effectiveness of a search based problem-solving programtwo things can do.
  1. Improve the generate procedure so that only good moves generated.
  2. Improve the test procedure so that the best move will recognize and explored first.
- If we use legal-move generator then the test procedure will have to look at each of thembecause the test procedure must look at so many possibilities, it must be fast.
- Instead of the legal-move generator, we can use plausible-move generator in which onlysome small numbers of promising moves generated.
- As the number of lawyers available moves increases, it becomes increasingly importantin applying heuristics to select only those moves that seem more promising.
- The performance of the overall system can improve by adding heuristic knowledge intoboth the generator and the tester.
- In game playing, a goal state is one in which we win but the game like chess. It is not possible. Even we have good plausible move generator.
- The depth of the resulting tree or graph and its branching factor is too great.
- It is possible to search tree only ten or twenty moves deep then in order to choose the best move. The resulting board positions must compare to discover which is most advantageous.
- This is done using static evolution function, which uses whatever information it has to evaluate individual board position by estimating how likely they are to lead eventually toa win.
- Its function is similar to that of the heuristic function h’ in the A* algorithm: in the absence of complete information, choose the most promising position.

4.5. MINIMAX Search Procedure

- The minimax search is a depth-first and depth limited procedure.
- The idea is to start at the current position and use the plausible-move generator to generate the set of possible successor positions.

- Now we can apply the static evolution function to those positions and simply choose thebest one.
- After doing so, we can back that value up to the starting position to represent our evolution of it.
- Here we assume that static evolution function returns larger values to indicate good situations for us.
- So our goal is to maximize the value of the static evaluation function of the next boardposition.
- The opponents' goal is to minimize the value of the static evaluation function.
- The alternation of maximizing and minimizing at alternate ply when evaluations areto be pushed back up corresponds to the opposing strategies of the two players is called MINIMAX.
- It is the recursive procedure that depends on two procedures
    - MOVEGEN(position, player)— The plausible-move generator, which returns alist of nodes representing the moves that can make by Player in Position.
    - STATIC(position, player)– static evaluation function, which returns a number representing the goodness of Position from the standpoint of Player.
- With any recursive program, we need to decide when recursive procedure should stop.
- There are the variety of factors that may influence the decision they are,
    - Has one side won?
    - How many plies have we already explored? Or how much time is left?
    - How stable is the configuration?
- We use DEEP-ENOUGH which assumed to evaluate all of these factors and to return TRUE if the search should be stopped at the current level and FALSE otherwise.
- It takes two parameters, position, and depth, it will ignore its position parameter and simply return TRUE if its depth parameter exceeds a constant cut off value.
- One problem that arises in defining MINIMAX as a recursive procedure is that it needs toreturn not one but two results.
    - The backed-up value of the path it chooses.
    - The path itself. We return the entire path even though probably only the first element, representing the best move from the current position, actually needed.
- We assume that MINIMAX returns a structure containing both results and we have twofunctions, VALUE and PATH that extract the separate components.
- Initially, It takes three parameters, a board position, the current depth of the search, andthe player to move,
    - MINIMAX(current,0,player-one) If player –one is to move
    - MINIMAX(current,0,player-two) If player –two is to move

4.6. Adding alpha-beta cutoffs

- Minimax procedure is a depth-first process. One path is explored as far as time allows,the static evolution function is applied to the game positions at the last step of the path.
- The efficiency of the depth-first search can improve by branch and bound technique inwhich partial solutions that clearly worse than known solutions can abandon early.
- It is necessary to modify the branch and bound strategy to include two bounds, one foreach of the players.
- This modified strategy called alpha-beta pruning.
- It requires maintaining of two threshold values, one representing a lower bound on that amaximizing node may ultimately assign (we call this alpha).
- And another representing an upper bound on the value that a minimizing node may assign(this we call beta).

- Each level must receive both the values, one to use and one to pass down to the next levelto use.
- The MINIMAX procedure as it stands does not need to treat maximizing and minimizinglevels differently. Since it simply negates evaluation each time it changes levels.
- Instead of referring to alpha and beta, MINIMAX uses two values, USE-THRESH andPASSTHRESH.
- USE-THRESH used to compute cutoffs. PASS-THRESH passed to next level as its USETHRESH.
- USE-THRESH must also pass to the next level, but it will pass as PASS-THRESH so thatit can be passed to the third level down as USE-THRESH again, and so forth.
- Just as values had to negate each time they passed across levels.
- Still, there is no difference between the code required at maximizing levels and that required at minimizing levels.
- PASS-THRESH should always the maximum of the value it inherits from above and thebest move found at its level.
- If PASS-THRESH updated the new value should propagate both down to lower levels. And back up to higher ones so that it always reflects the best move found anywhere in thetree.
- The MINIMAX-A-B requires five arguments, position, depth, player, Use-thresh, and passThresh.
- MINIMAX-A-B(current,0,player-one,maximum value static can compute, minimum value static can compute).

Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS)

There are two common ways to traverse a graph, BFS and DFS. Considering a Tree (or Graph)of huge height and width, both BFS and DFS are not very efficient due to following reasons.

1. **DFS** first traverses nodes going through one adjacent of root, then next adjacent. The problem with this approach is, if there is a node close to root, but not in first few subtreesexplored by DFS, then DFS reaches that node very late. Also, DFS may not find shortestpath to a node (in terms of number of edges).

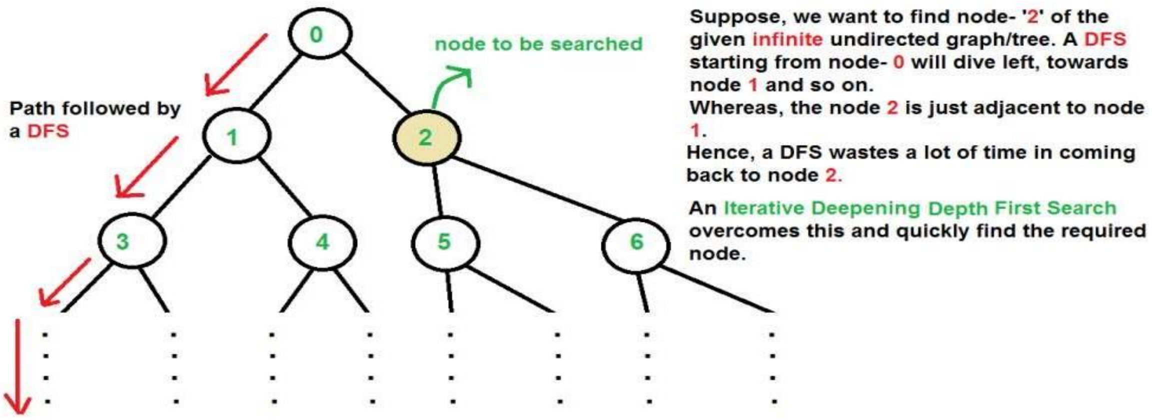

Path followed by a DFS
node to be searched
0
1
2
3
4
5
6
Suppose, we want to find node- '2' of the given infinite undirected graph/tree. A DFS starting from node- 0 will dive left, towards node 1 and so on.
Whereas, the node 2 is just adjacent to node 1.
Hence, a DFS wastes a lot of time in coming back to node 2.
An Iterative Deepening Depth First Search overcomes this and quickly find the required node.

Figure 4.2 Alpha Beta Cutoff

2. **BFS** goes level by level, but requires more space. The space required by DFS is O(d) where d is depth of tree, but space required by BFS is O(n) where n is number of nodes intree (Why? Note that the last level of tree can have around n/2 nodes and second last level n/4 nodes and in BFS we need to have every level one by one in queue).

4.7. Planning

Blocks World Problem

In order to compare the variety of methods of planning, we should find it useful to look at all ofthem in a single domain that is complex enough that the need for each of the mechanisms is apparent yet simple enough that easy-to-follow examples can be found.

- There is a flat surface on which blocks can be placed.
- There are a number of square blocks, all the same size.
- They can be stacked one upon the other.
- There is robot arm that can manipulate the blocks.

Actions of the robot arm

1. UNSTACK(A, B): Pick up block A from its current position on block B.
2. STACK(A, B): Place block A on block B.
3. PICKUP(A): Pick up block A from the table and hold it.
4. PUTDOWN(A): Put block A down on the table. Notice that the robot arm can hold only one block at a time.**Predicates**

- In order to specify both the conditions under which an operation may be performed andthe results of performing it, we need the following predicates:
  1. ON(A, B): Block A is on Block B.
  2. ONTABLES(A): Block A is on the table.
  3. CLEAR(A): There is nothing on the top of Block A.
  4. HOLDING(A): The arm is holding Block A.
  5. ARMEMPTY: The arm is holding nothing.
- Robot problem-solving systems (STRIPS)
  1. List of new predicates that the operator causes to become true is ADD List
  2. Moreover, List of old predicates that the operator causes to become false is DELETE List
  3. PRECONDITIONS list contains those predicates that must be true for the operator to beapplied.

STRIPS style operators for BLOCKs World

STACK(x, y)

P: CLEAR(y)^HOLDING(x)

D: CLEAR(y)^HOLDING(x)

A: ARMEMPTY^ON(x, y)

UNSTACK(x, y)

PICKUP(x)

P: CLEAR(x) ^ ONTABLE(x) ^ARMEMPTY

D: ONTABLE(x) ^ ARMEMPTY

A: HOLDING(x)

PUTDOWN(x)

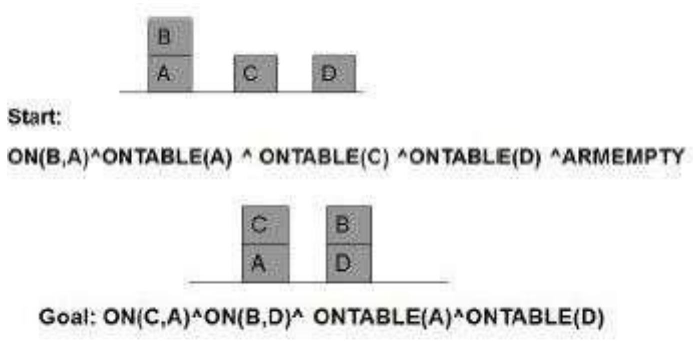
4.7.1.Goal Stack Planning

To start with goal stack is simply:

- ON(C,A)^ON(B,D)^ONTABLE(A)^ONTABLE(D)

This problem is separate into four sub-problems, one for each component of the goal.

Two of the sub-problems ONTABLE(A) and ONTABLE(D) are already true in the initial state.


B
A
C
D
Start:
ON(B,A)^ONTABLE(A) ^ ONTABLE(C) ^ONTABLE(D) ^ARMEMPTY
C
A
B
D
Goal: ON(C,A)^ON(B,D)^ ONTABLE(A)^ONTABLE(D)

Alternative 1: Goal Stack:

- ON(C,A)
- ON(B,D)
- ON(C,A)^ON(B,D)^OTAD

Alternative 2: Goal stack:

- ON(B,D)
- ON(C,A)
- ON(C,A)^ON(B,D)^OTAD

Exploring Operators

- Pursuing alternative 1, we check for operators that could cause ON(C, A)
- Out of the 4 operators, there is only one STACK. So it yields:
    - STACK(C,A)
    - ON(B,D)
    - ON(C,A)^ON(B,D)^OTAD
- Preconditions for STACK(C, A) should be satisfied, we must establish them as sub-goals:
    - CLEAR(A)
    - HOLDING(C)
    - CLEAR(A)^HOLDING(C)
    - STACK(C,A) o ON(B,D)
    - ON(C,A)^ON(B,D)^OTAD
- Here we exploit the Heuristic that if HOLDING is one of the several goals to be achieved at once, it should be tackled last.
- Next, we see if CLEAR(A) is true. It is not. The only operator that could make it true is UNSTACK(B, A). Also, This produces the goal stack:
    - ON(B, A)
    - CLEAR(B)
    - ON(B,A)^CLEAR(B)^ARMEMPTY
    - UNSTACK(B, A)
    - HOLDING(C)
    - CLEAR(A)^HOLDING(C)
    - STACK(C,A)
    - ON(B,D)
    - ON(C,A)^ON(B,D)^OTAD
- We see that we can pop predicates on the stack till we reach HOLDING(C) for which we need to find a suitable operator.
- Moreover, The operators that might make HOLDING(C) true: PICKUP(C) and UNSTACK(C, x). Without looking ahead, since we cannot tell which of these operatorsis

appropriate. Also, we create two branches of the search tree corresponding to the Table 4.1 goal stacks:

ALT 1:	ALT 2:
ONTABLE(C)	ON(C,x)
CLEAR(C)	CLEAR(C)
ARM EMPTY	ARM EMPTY
ONTABLE(C)	ON(C,x)^CLEAR(C) ^ARMEMPTY
PICKUP(C)	UNSTACK(C,x)
CLEAR(A)^ HOLDING(C)	CLEAR(A)^ HOLDING(C)
STACK(C,A)	STACK(C,A)
ON(B,D)	ON(B,D)
ON(C,A)^ON(B,D) ^OTAD	ON(C,A)^ON(B,D) ^OTAD

Table 4.1 Process of Goal Stack Planning

Complete plan

1. UNSTACK(C, A)
2. PUTDOWN(C)
3. PICKUP(A)
4. STACK(A, B)
5. UNSTACK(A, B)
6. PUTDOWN(A)
7. PICKUP(B)
8. STACK(B, C)
9. PICKUP(A)
10. STACK(A,B)

4.8. Planning Components

- Methods which focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interactions among the subparts as they are detected during the problem-solving process are often called as planning.
- Planning refers to the process of computing several steps of a problem-solving procedurebefore executing any of them.

Components of a planning system

Choose the best rule to apply next, based on the best available heuristic information.

- The most widely used technique for selecting appropriate rules to apply is first to isolatea set of differences between desired goal state and then to identify those rules that are relevant to reduce those differences.
- If there are several rules, a variety of other heuristic information can be exploited to choose among them.

Apply the chosen rule to compute the new problem state that arises from its application.

- In simple systems, applying rules is easy. Each rule simply specifies the problem statethat would result from its application.
- In complex systems, we must be able to deal with rules that specify only a small part ofthe complete problem state.

- One way is to describe, for each action, each of the changes it makes to the state description.

Detect when a solution has found.

- A planning system has succeeded in finding a solution to a problem when it has found asequence of operators that transform the initial problem state into the goal state.
- How will it know when this has done?
- In simple problem-solving systems, this question is easily answered by a straightforwardmatch of the state descriptions.
- One of the representative systems for planning systems is predicate logic. Suppose that asa part of our goal, we have the predicate P(x).
- To see whether P(x) satisfied in some state, we ask whether we can prove P(x) given theassertions that describe that state and the axioms that define the world model.

Detect dead ends so that they can abandon and the system’s effort directed in more fruitful directions.

- As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to asolution.
- The same reasoning mechanisms that can use to detect a solution can often use fordetecting a dead end.
- If the search process is reasoning forward from the initial state. It can prune any path thatleads to a state from which the goal state cannot reach.
- If search process reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot reach or because little progress made.

Detect when an almost correct solution has found and employ special techniques to make it totally correct.

- The kinds of techniques discussed are often useful in solving nearly decomposableproblems.
- One good way of solving such problems is to assume that they are completely decomposable, proceed to solve the sub-problems separately. And then check that whenthe sub-solutions combined. They do in fact give a solution to the original problem.

4.8.1.Goal Stack Planning Method

- In this method, the problem solver makes use of a single stack that contains both goalsand operators. That have proposed to satisfy those goals.
- The problem solver also relies on a database that describes the current situation and a setof operators described as PRECONDITION, ADD and DELETE lists.
- The goal stack planning method attacks problems involving conjoined goals by solvingthe goals one at a time, in order.
- A plan generated by this method contains a sequence of operators for attaining the firstgoal, followed by a complete sequence for the second goal etc.
- At each succeeding step of the problem-solving process, the top goal on the stack willpursue.
- When a sequence of operators that satisfies it, found, that sequence applied to the statedescription, yielding new description.
- Next, the goal that then at the top of the stack explored. And an attempt made to satisfy it,starting from the situation that produced as a result of satisfying the first goal.

- This process continues until the goal stack is empty.
- Then as one last check, the original goal compared to the final state derived from theapplication of the chosen operators.
- If any components of the goal not satisfied in that state. Then those unsolved parts of thegoal reinserted onto the stack and the process resumed.

4.9. Nonlinear Planning using Constraint Posting

Difficult problems cause goal interactions.

The operators used to solve one sub-problem may interfere with the solution to a previoussub-problem.

Most problems require an intertwined plan in which multiple sub-problems worked on simultaneously.

Such a plan is called nonlinear plan because it is not composed of a linear sequence of complete sub-plans.

Constraint Posting

- The idea of constraint posting is to build up a plan by incrementally hypothesizing operators, partial orderings between operators, and binding of variables within operators.
- At any given time in the problem-solving process, we may have a set of useful operators but perhaps no clear idea of how those operators should order with respect to each other.
- A solution is a partially ordered, partially instantiated set of operators to generate anactual plan. And we convert the partial order into any number of total orders.

Constraint Posting versus State Space search

State Space Search

- Moves in the space: Modify world state via operator
- Model of time: Depth of node in search space
- Plan stored in Series of state transitions

Constraint Posting Search

- Moves in the space: Add operators, Oder Operators, Bind variables Or Otherwise constrain plan
- Model of Time: Partially ordered set of operators
- Plan stored in Single node

Algorithm: Nonlinear Planning (TWEAK)

- Initialize S to be the set of propositions in the goal state.
- Remove some unachieved proposition P from S.
- Moreover, Achieve P by using step addition, promotion, DE clobbering, simple establishment or separation.
- Review all the steps in the plan, including any new steps introduced by step addition, tosee if any of their preconditions unachieved. Add to S the new set of unachieved preconditions.
- Also, If S is empty, complete the plan by converting the partial order of steps into a totalorder, instantiate any variables as necessary.

4.10. Hierarchical Planning

- In order to solve hard problems, a problem solver may have to generate long plans.

- It is important to be able to eliminate some of the details of the problem until a solutionthat addresses the main issues is found.
- Then an attempt can make to fill in the appropriate details.
- Early attempts to do this involved the use of macro operators, in which larger operatorswere built from smaller ones.
- In this approach, no details eliminated from actual descriptions of the operators.

5. Uncertainty and Certainty in Probability Theory

In probability theory, **uncertainty** and **certainty** describe the likelihood of events occurring. These concepts are fundamental to understanding how probability quantifies the randomness and variability in various scenarios.

Certainty:***Certainty** refers to the absolute surety of an event occurring. In probability terms, an event is certain if it is guaranteed to happen. The probability of a certain event is 1.*

Examples:

In a fair six-sided die, the probability that one of the numbers 1, 2, 3, 4, 5, or 6 will land face up is 1.

The probability that the sun will rise tomorrow (assuming the natural laws continue as we know them) is considered to be 1.

Mathematical Representation:

If E is an event, then P(E)=1 indicates certainty.

Uncertainty:***Uncertainty refers to a situation in which the outcome of an event is not guaranteed. In probability theory, uncertainty is quantified by assigning a probability between 0 and 1 to an event. The closer the probability is to 0, the less likely the event is to occur; the closer to 1, the more likely the event is to occur.***

Examples:

In a fair coin toss, the probability of getting heads is 0.5, indicating an uncertain outcome, as there is an equal chance of getting tails.

The probability of rain tomorrow might be 0.3, indicating some uncertainty about whether it will rain.

Mathematical Representation:

For an event E, the probability P(E) represents the degree of uncertainty, where $0 \leq P(E) \leq 1$.

Bayes theorem and Bayesian networks

Bayes' Theorem:Bayes' Theorem is a fundamental concept in probability theory and statistics. It provides a way to update the probability of a hypothesis based on new evidence. It describes the probability of an event based on prior knowledge of conditions that might be related to the event.

Formula:

The formula for Bayes' Theorem is:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where:

- P(A|B) is the **posterior probability**: the probability of event A occurring given that event B has occurred.
- P(B|A) is the **likelihood**: the probability of event B occurring given that A is true.
- P(A) is the **prior probability**: the initial probability of event A.
- P(B) is the **marginal probability**: the total probability of event B.

Bayesian Networks:**Bayesian Networks** (also known as Belief Networks or Bayes Nets) are graphical models that represent the probabilistic relationships among a set of variables. They are a powerful tool for reasoning under uncertainty, combining principles from graph theory and probability theory.

Structure:

A Bayesian Network consists of:

1. **Nodes:** Represent random variables. Each node is associated with a probability distribution.
2. **Directed Edges:** Indicate conditional dependencies between variables. An edge from node A to node B means that A has a direct influence on B.
3. The graph structure is a **Directed Acyclic Graph (DAG)**, meaning there are no cycles, and the direction of the edges encodes the flow of influence.

Conditional Probability Tables (CPTs)Each node in a Bayesian Network has an associated Conditional Probability Table (CPT) that quantifies the effect of the parent nodes on the node. The CPT specifies the probability of each possible state of the node, given the states of its parents.

Bayes' Theorem provides a way to update the probability of a hypothesis based on new evidence, forming the foundation for Bayesian inference. **Bayesian Networks** extend this concept by representing complex probabilistic relationships among multiple variables using a graphical model. They are widely used in various fields, including artificial intelligence, genetics, and risk analysis, for decision-making under uncertainty.

1. Data visualization

Data visualization is a crucial component of the machine learning workflow. It involves the graphical representation of data and model results, making complex information more accessible, understandable, and usable. Visualization aids in exploring and understanding data, communicating findings, and making informed decisions throughout the machine learning process.

Types of Data Visualizations:

- **Univariate Visualizations:**
  - **Histograms:** Display the distribution of a single variable, showing the frequency of different ranges of values.
  - **Box Plots:** Summarize the distribution of a variable by showing its quartiles and potential outliers.
  - **Density Plots:** Show the distribution of a variable using a continuous curve.
- **Bivariate and Multivariate Visualizations:**
  - **Scatter Plots:** Show the relationship between two continuous variables, often used to identify correlations.
  - **Heatmaps:** Represent data in matrix form, with colors indicating the magnitude of values, useful for showing correlations or distances.
  - **Pair Plots:** Display pairwise relationships between variables in a dataset, often using scatter plots or histograms.
  - **Violin Plots:** Combine box plots and density plots to show the distribution of data across different categories.
- **Categorical Data Visualizations:**
  - **Bar Charts:** Show the frequency or proportion of different categories.
  - **Pie Charts:** Display the relative proportions of different categories within a whole.
  - **Count Plots:** Similar to bar charts but show the count of occurrences for each category.

- **Time Series Visualizations:**
  - **Line Plots:** Show data points over time, useful for identifying trends and patterns in time series data.
  - **Lag Plots:** Help in identifying autocorrelation in time series data.
- **Model Evaluation Visualizations:**
  - **Confusion Matrix:** A table showing the true positives, false positives, true negatives, and false negatives for classification models.
  - **ROC Curve and AUC:** Plot the true positive rate against the false positive rate, providing a measure of model performance.
  - **Precision-Recall Curve:** Shows the trade-off between precision and recall for different thresholds.
  - **Residual Plots:** Show the residuals (errors) of a regression model, useful for diagnosing issues like heteroscedasticity or non-linearity.

5.1. Introduction to expert system and application of expert systems:

Expert system is an artificial intelligence program that has expert-level knowledge about a particular domain and knows how to use its knowledge to respond properly. Domain refers to the area within which the task is being performed. Ideally the expert systems should substitute a human expert. Edward Feigenbaum of Stanford University has defined expert system as an intelligent computer program that uses knowledge and inference procedures to solveproblems that are difficult enough to require significant human expertise for their solutions.It is a branch of artificial intelligence introduced by researchers in the Stanford Heuristic Programming Project.The expert systems is a branch of AI designed to work within a particular domain. As an expert is a person who can solve a problem with the domain knowledge in hands it should be able to solve problems at the level of a human expert. The source of knowledge may come from a human expert and/or from books, magazines and internet. As knowledge play a key role in the functioning of expert systems, they are also known as knowledge-based systems and knowledge-based expert systems. The expert ‘s knowledge about solving the given specific problems is called knowledge domain of the expert.

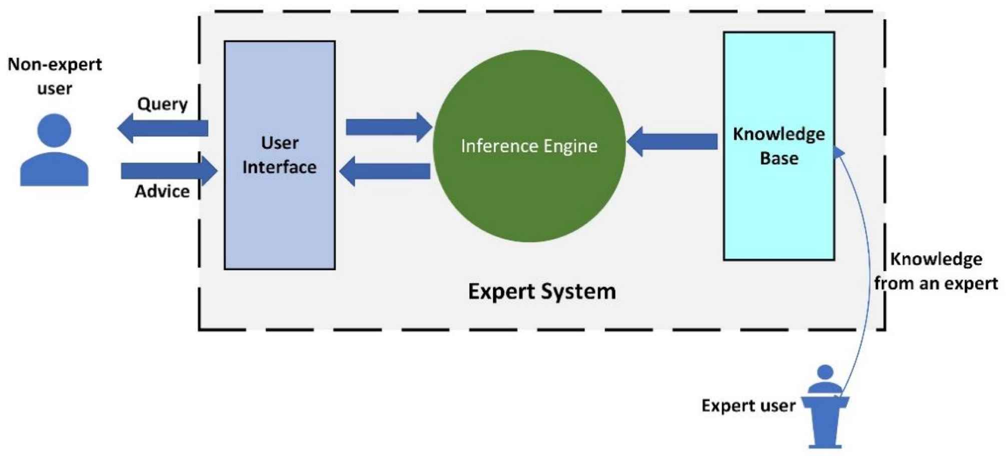


Figure 5.1 Expert System Architecture.

The structure of an expert system is depicted inFigure 5.1, and each of its components is briefly introduced in the following subsection.

5.1.1.Knowledgebase:

The knowledgebase is considered the core of the expert system; it comprises rules and facts that model expert knowledge about the problem domain. It is a domain-specific information repository gathered from the human expert via the acquisition of knowledge. An expert system's knowledgebase includes both empirical and heuristic knowledge. It replicates the expert's knowledge in the form of rules of development, frames logic, etc.

5.1.2.Inference Engine:

A significant element of any expert system is the inference engine. The inference engine is the component that determines how the expert system suitably interprets the information in the knowledgebase. Its function is to work with the system and user data available to generate solutions to problems; extract knowledge and produce answers, predictions and recommendations from the database just as a human expert does.

The Inference Engine generally uses two strategies for acquiring knowledge from the Knowledge Base, namely –

- Forward Chaining
- Backward Chaining

Forward Chaining –

Forward Chaining is a strategic process used by the Expert System to answer the questions – What will happen next. This strategy is mostly used for managing tasks like creating a conclusion as shown in Figure 5.2, result or effect. Example – prediction or share market movement status.

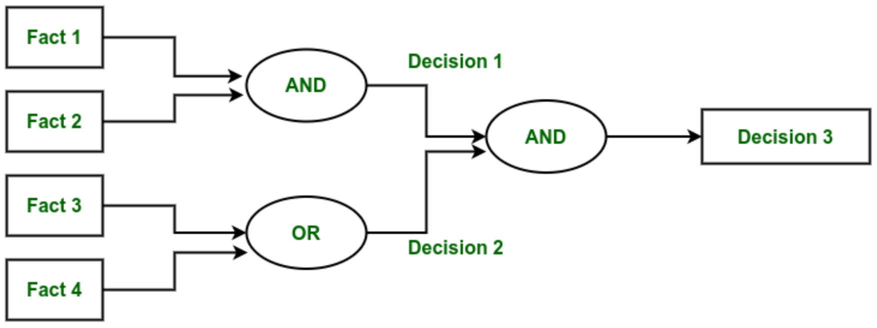


Figure 5.2 Illustration of Forward Chaining

Backward Chaining –

Backward Chaining is a strategy used by the Expert System to answer the questions – Why this has happened. This strategy is mostly used to find out the root cause or reason behind it, considering what has already happenedas shown inFigure 5.3. Example – diagnosis of stomach pain, blood cancer or dengue, etc.

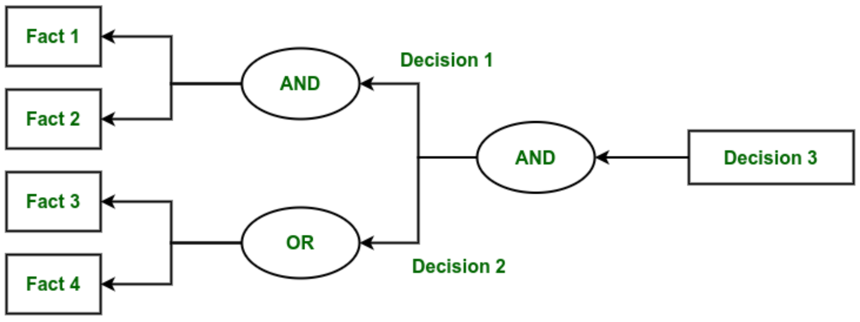


Figure 5.3Illustration of Backward Chaining

5.1.3.User Interface:

The user interface component of the expert system controls the user-system interaction and communication. It provides facilities such as menus, GUI and so on that enable a non-expert user to query the system and receive feedback. Thus, it is an agent that enables the user to communicate with the expert system. The user interface function is to ease the use by developers, users and administrators of an expert system.

5.2. Various expert system shells:

Expert system shells are software environments that provide the necessary tools to develop and implement expert systems. These shells usually include a knowledge base, an inference engine, and a user interface. Expert system shells are the main choice for building small ESs due primarily totheir ease of use. The shell is really a ready-made ES without a knowledge base.All the programming components are there, waiting for rules to be entered intothe system.

Below are some well-known expert system shells:

5.2.1.CLIPS (C Language Integrated Production System)

CLIPS stands for C-Language Integrated Production System. As the meaning suggests the CLIPS expert system shell is written in the procedural langue C. CLIPS was developed in 1985 at NASA’s Johnson Space Centre. It is a rule-based programming language that is used for creating an expert system. CLIPS are used in systems where the heuristic solution is easier to implement and maintain than a traditional algorithmic approach.CLIPS provides 03 different tools for knowledge representation in the form of programming methodologies/ programming paradigms. The 03 methods are:Procedural, Object-oriented and Rule-based programming.

Since the CLIPS is written in C language, the expert system developed by CLIPS requires ANSI compiler also. Since the systems that already have the ANSI compiler can easily run the expert system without changing the source code.

An example of a simple expert system created using CLIPS. This system will diagnose a basic set of medical conditions based on symptoms provided by the user.

Setting Up CLIPS

First, ensure you have CLIPS installed on your system. You can download it from the official CLIPS website.

Writing the Rules and Facts

Create a file named diagnosis.clp with the following content:

```
(defrule start
   =>
   (printout t "Welcome to the Medical Diagnosis System" crlf)
   (printout t "Please enter your symptoms one by one followed by 'done':" crlf))


(defrule input-symptom
   (declare (salience 100))
   (not (symptom ?))
   =>
   (printout t "Symptom: ")
   (bind ?symptom (read))
   (if (neq ?symptom done)
      then
      (assert (symptom ?symptom))
      (retract (input-symptom))
      (assert (input-symptom))))

(defrule flu
   (symptom fever)
   (symptom headache)
   (symptom cough)
   =>
   (printout t "Diagnosis: You might have the flu." crlf))

(defrule cold
   (symptom cough)
   (symptom sneezing)
   =>
   (printout t "Diagnosis: You might have a cold." crlf))

(defrule allergy
   (symptom sneezing)
   (symptom itchy-eyes)
   =>
   (printout t "Diagnosis: You might have allergies." crlf))

(defrule no-diagnosis
   (declare (salience -100))
   =>
   (printout t "Diagnosis: No matching condition found." crlf))
```

Running the Expert System

- Open CLIPS from your command line or terminal.
- Load the rules and facts by typing (load "diagnosis.clp").
- Start the rule engine by typing (reset) followed by (run).

Interaction Example:

```
CLIPS> (load "diagnosis.clp")

CLIPS> (reset)

CLIPS> (run)

Welcome to the Medical Diagnosis System

Please enter your symptoms one by one followed by 'done':

Symptom: fever

Symptom: headache

Symptom: cough

Symptom: done

Diagnosis: You might have the flu.
```

Here's what an interaction might look like:

<u>In this example:</u>

- The start rule welcomes the user.
- The input-symptom rule continuously prompts the user to enter symptoms until they type 'done'.
- The flu, cold, and allergy rules diagnose the condition based on the symptoms entered.
- The no-diagnosis rule is a catch-all for when no conditions match the provided symptoms.
- This is a basic example to demonstrate how CLIPS can be used to create an expert system. You can expand it by adding more rules and symptoms to handle a wider range of diagnoses.

5.2.2.Jess (Java Expert System Shell)

Jess is a rule engine and scripting environment written in Java. It is based on the CLIPS expert system shell but is fully integrated with the Java platform.

A simple example of an expert system using Jess (Java Expert System Shell) to diagnose basic medical conditions based on symptoms.

<u>Setting Up Jess</u>

First, ensure you have Jess installed. You can download it from the official Jess website.

<u>Writing the Rules and Facts</u>

Create a file named diagnosis.clp with the following content:

```
(defrule start

   =>

   (printout t "Welcome to the Medical Diagnosis System" crlf)

   (printout t "Please enter your symptoms one by one followed by 'done':" crlf))


(defrule input-symptom
   (declare (salience 100))
   (not (symptom ?))
   =>
   (printout t "Symptom: ")
   (bind ?symptom (read))
   (if (neq ?symptom "done")
       then
       (assert (symptom ?symptom))
       (retract (input-symptom))
       (assert (input-symptom))))

(defrule flu
   (symptom "fever")
   (symptom "headache")
   (symptom "cough")
   =>
   (printout t "Diagnosis: You might have the flu." crlf))

(defrule cold
   (symptom "cough")
   (symptom "sneezing")
   =>
   (printout t "Diagnosis: You might have a cold." crlf))

(defrule allergy
   (symptom "sneezing")
   (symptom "itchy-eyes")
   =>
   (printout t "Diagnosis: You might have allergies." crlf))

(defrule no-diagnosis
   (declare (salience -100))
   =>
   (printout t "Diagnosis: No matching condition found." crlf))
```

Writing the Java Code

Create a Java file named DiagnosisSystem.java with the following content:

```
import jess.JessException;
import jess.Rete;
import jess.Console;

public class DiagnosisSystem {
  public static void main(String[] args) {
    try {
      // Create a Rete object
      Rete engine = new Rete();

      // Load the CLIPS file
engine.batch("diagnosis.clp");

      // Run the rules
engine.reset();
engine.run();

      // Console interface for user input
      Console console = new Console(engine);
console.execute(true);
    } catch (JessException e) {
e.printStackTrace();
    }
  }
}
```

Compiling and Running the Program

Compile the Java program:

```
javac -cp jess.jar DiagnosisSystem.java
```

Run the program:

```
java -cp .:jess.jarDiagnosisSystem
```

Interaction Example

Here's what an interaction might look like:

```
Welcome to the Medical Diagnosis System

Please enter your symptoms one by one followed by 'done':

Symptom: fever

Symptom: headache

Symptom: cough

Symptom: done

Diagnosis: You might have the flu.
```

In this example:

- The start rule welcomes the user.
- The input-symptom rule continuously prompts the user to enter symptoms until they type 'done'.
- The flu, cold, and allergy rules diagnose the condition based on the symptoms entered.
- The no-diagnosis rule is a catch-all for when no conditions match the provided symptoms.
- This simple expert system in Jess demonstrates how you can create a rule-based system to diagnose medical conditions based on user input. You can expand it by adding more rules and symptoms to handle a wider range of diagnoses.

5.2.3.Prolog

Various Prolog-based shells have been developed, leveraging the logicprogrammingcapabilities of Prolog to create expert systems. These shells are known for their strongpatternmatching and symbolic reasoning abilities.Prolog is a logic programming language usedextensively for building expert systems. Prolog provides mechanisms for pattern matching,tree-based data structuring, and automatic backtracking, making it suitable for creatingcomplex rule-based systems.

A simple example of an expert system written in Prolog. This system will diagnose a basic set of medical conditions based on symptoms provided by the user.

Writing the Prolog Rules and Facts

Create a file named diagnosis.pl with the following content:

```
% Define the symptoms and their associated diseases

disease(flu) :-

   symptom(fever),

   symptom(headache),

   symptom(cough).
```

```
disease(cold) :-
    symptom(cough),
    symptom(sneezing).

disease(allergy) :-
    symptom(sneezing),
    symptom(itchy_eyes).

% Ask the user for symptoms
ask_symptom(Symptom) :-
format('Do you have ~w? (yes/no): ', [Symptom]),
read(Reply),
    (Reply == yes -> assert(symptom(Symptom)); assert(not(symptom(Symptom))), fail).

% Check for symptoms
symptom(Symptom) :-
ask_symptom(Symptom).

symptom(Symptom) :-
    not(ask_symptom(Symptom)), !, fail.

% Start diagnosis
start_diagnosis :-
retractall(symptom(_)),
retractall(not(_)),
    (disease(Disease) -> format('You might have ~w.', [Disease]);
format('No matching condition found.')).
```

<u>Running the Expert System</u>

- Ensure you have SWI-Prolog installed. You can download it from the official SWI-Prolog website.
- Open your terminal or command line and navigate to the directory where diagnosis.pl is located.
- Start the Prolog interpreter by typing swipl.
- Load the Prolog file by typing [diagnosis]..
- Start the diagnosis by typing start_diagnosis.

Interaction ExampleHere's what an interaction might look like:

```
?- [diagnosis].

true.

?-start_diagnosis.

 Do you have fever? (yes/no): yes.

Do you have headache? (yes/no): yes.

Do you have cough? (yes/no): yes.

You might have flu.

true.
```

In this example:

- The disease/1 predicates define the conditions based on symptoms.
- The ask_symptom/1 predicate asks the user whether they have a specific symptom and asserts it if the answer is yes.
- The symptom/1 predicate checks for the presence of a symptom by asking the user.
- The start_diagnosis/0 predicate initializes the diagnosis process, retracting any previously asserted symptoms or negative assertions, and then checks for diseases based on the symptoms provided.
- This simple Prolog-based expert system demonstrates how you can create a rule-based system to diagnose medical conditions based on user input. You can expand it by adding more rules and symptoms to handle a wider range of diagnoses.

5.3. Vidwan frame work

Vidwan is an expert system shell developed at the National Centre for Software Technology,Bombay. It enables knowledge of a domain to be encodedin the form of if-then rules. It supports backward chaining as the inferenceprocedure. It enables the user to represent the uncertainty associated withthe domain knowledge using certainty factors. It provides explanation facilities such as why and how. Vidwan allows one to create a rule base using anystandard text editor or the system’s built-in interactive editor.We briefly look at the mainfeatures of Vidwan and examine the syntax ofits rule language.

Some special features such asthe report generator andcommand file directed invocation of Vidwan are also discussed briefly. Formore details, please refer to the user manual of the software [NCST, 1993].

An Overview of Features

Vidwan prototypes serve various applications, such as medical advisors (cardiology, respiratory diseases, rheumatism), troubleshooting systems (electric motors, electronics, printers), and financial advisors (credit worthiness, share investment, tax planning).The fundamental elements of a rule base for Vidwan are the domain's parameters of interest (individual age, reported symptoms, etc.) and their interrelationships. In Vidwan, the qualities of the domain are referred to as parameters. The relationships between them are depicted using if-then rules.

Rule

The form (structure) of a Vidwan rule is shown below:

rule (<id> , <attr> , <value> , <cf>) :- <anteset>.

The symbols with the angular brackets indicate generic terms which will bedescribed in the following paragraphs. All other non-blank characters area mandatory part of the syntax. Each rule has a unique identifier, <id>.<attr> is the name of the attribute and <value> is its value. <cf> is thecertainty factor, which can be associated with the attribute.

The <anteset> consists of one or more antecedents separated by commas.The last antecedent should be followed by a full stop. Each antecedent is ofthe form:

<opr>(<attr> , <value>)

where, <opr> is one of a pre-defined set of operators. For example, a rulefrom the Hruday rule base looks like this:

rule(chp2, chest pain type, cardiac, 0.25) :-

is(complaint, chest pain),

is(chest pain location, left side of chest).

This rule says that:

If the patient’s complaint is chest pain

and the site of the chest pain is the left side of the chest

then there is suggestive evidence (0.25) thatthe type of the chest pain is cardiac.

Each rule can have only one consequent. The antecedents of a rule areconjunctively connected (anded) together. Rules can have zero or moreantecedents. A rule with zero antecedents is applicable (i.e., can be fired)in all cases. (This can be used to define default values for parameters.) Forexample, the following,

rule (bpl, bp-status, normal, 0.3).

says that unless modified by other rules, blood pressure status is normal withcertainty 0.3.

In the description above, all symbols, except the symbols with angular brackets (generic terms),are part of the syntax and have to be typed in as given.One or more blanks are allowed wherever there is at least one blank in thedescription.

The core of an antecedent or consequent is an attribute-value tuple. In the example above, the attributes are *chest_ pain_ type*, *complaint*and *chest_pain_location* and their values are *cardiac*, *chest_pain* and*left_side_of_chest* respectively.

5.4. Case studies: MYCIN.

MYCIN is the name of a decision support system developed by Stanford University in the early- to mid-seventies, built to assist physicians in the diagnosis of infectious diseases. The system (also known as an" expert system") would ask a series of questions designed to emulate the thinking of an expert in the field of infectious disease (hence the "expert-"), and from the responses to these questions give a list of possible diagnoses, with probability, as well as recommend treatment (hence the "decision support-"). The name "MYCIN" actually comes from antibiotics, many of which have the suffix "-mycin".

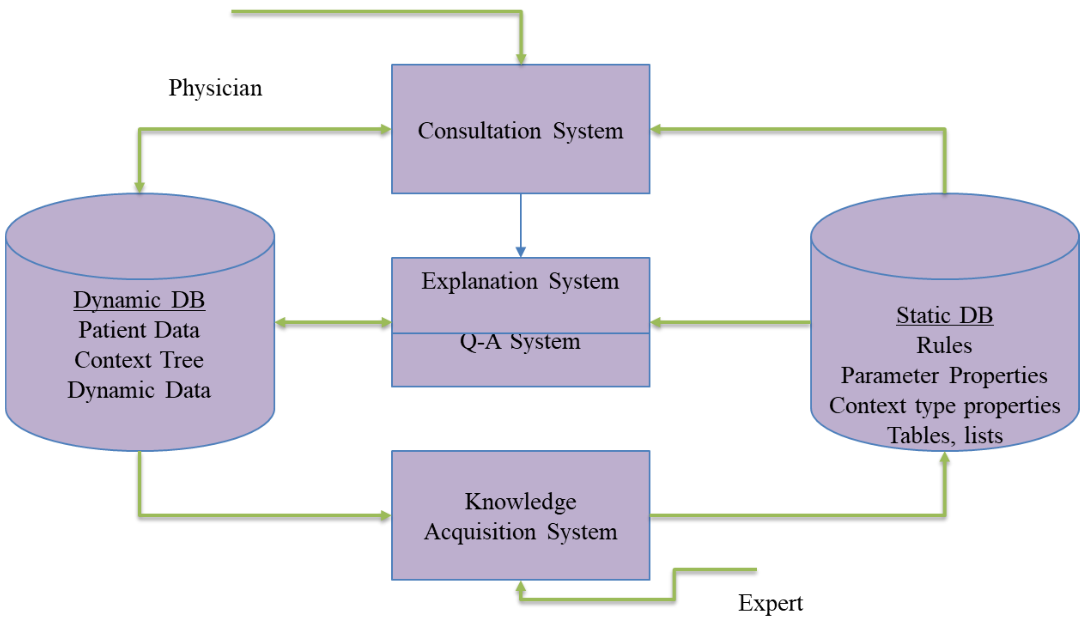


Figure 5.4MYCIN Architecture

MYCIN has three sub-systems:

- Consultation system
- Explanation System
- Rule Acquisition system

5.4.1.Mycin's Consultation System

- works out possible organisms and suggests treatments

Rule base

- Rules stored as premise -action pairs
- Premises are conjunctions and disjunctions of conditions
- Conditions normally evaluate to True or False, with some certainty factor on the evidence. Rules also have certainty factors. Combined to form new certainty factors
- Certainty factor - degree of belief attached to information
- Actions are either conclusions (e.g. microbe is type X) or instruction (e.g. remove drug from list of possible value)

Static and dynamic data structures

Static data structures: These store medical knowledge not suitable for storage as inferential rules: includes lists of organisms, knowledge tables with features of bacteria types, list of parameters

Parameters = features of patients, bacterial cultures, drugs…

Parameters can be Y/N (e.g. FEBRILE), single value (e.g. IDENTITY - if it's salmonella it can't be another organism as well) or multi-value (e.g. INFECT - patient can have more than one infection) Parameter properties include:

EXPECT range of possible values

PROMPT English sentence to elicit response

LABDATA can be known for certain from Lab data

LOOKAHEAD lists rules mentioning the parameter in their premise (e.g. a rule might need to know whether or not a patient is febrile)

UPDATED-BY lists rules mentioning the parameter in their action (i.e. they may draw a conclusion about the value of the parameter, such as the IDENTITY parameter)

Dynamic data structures store information about the evolving case - the patient details, possible diagnoses, rules consulted:

Control structure: MYCIN first attempts to create “patient context” containing information about the case, then tries to compile a list of therapies for the context. It uses a backward chaining mechanism, reasoning back from the goals it wants to prove to the data it has, rather than vice versa. The overall goal is "compile a list of therapies”. Questions can be prompted by the invocation of rules, to find out necessary data, to avoid unnecessary questions.

5.4.2. The Explanation System

MYCIN can answer questions about HOW a conclusion was reached and WHY a question was asked, either after a consultation or while it is going on. It does this by manipulating its record of the rules it invoked, the goal it was trying to achieve, the information it was trying to discover. Can also answer general questions (e.g. what would you prescribe for organism X?) by consulting its static data structures.

5.4.3. The Rule Acquisition System

Experts can enter new rules or edit existing rules. The system automatically adds the new rule tothe LOOKAHEAD list for all parameters mentioned in its premise, and to the UPDATED-BY list of all parameters mentioned in its action.

5.4.4.History

MYCIN was originally developed by Edward Shortliffe for Stanford Medical School in the early- and mid-1970's. Written in Lisp, a language (a set of languages, actually) geared towards artificial intelligence, MYCIN was one of the pioneering expert systems, and was the first such system implemented for the medical field. The Goal of MYCIN was to compete in an experiment conducted at Stanford Medical similar to the Turing Test. The case histories of ten patients with different types of meningitis were submitted to MYCIN as well as to eight human physicians, including a resident, a research fellow, and five faculty specialists in infectious disease. Both MYCIN and the human physicians were given the same information. Both MYCIN's and the human physician's recommendations (as well as a record of the treatment actually received by the patients) were sent to eight non-Stanford specialists, completely unidentified as to which recommendation was MYCIN's and which were authored by the physicians. The outside specialists gave MYCIN the highest score as far as accuracy of diagnosis and effectiveness of treatment.

The framework for MYCIN was derived from an earlier expert system called DENDRAL, created to find new chemical compounds in the field of organic chemistry.

5.4.5.How it works

MYCIN is an expert system comprised of two major components:

1. A knowledge base which stores the information the expert system "knows", much of which is derived from other information in the knowledge base.
2. An inference engine to derive knowledge from the presently known knowledge in the knowledge base.

Humans interface with MYCIN by answering a series of diagnostic questions akin to what a physician may ask a patient, as well as prompting for relevant test results. MYCIN takes this data as input and either arrives at a set of answers with respective probabilities, or branches to other questions in order to narrow its search. Researchers at Stanford found MYCIN to have an approximate correctness rate of 65%, which is better than the majority of physicians who are not specialists in diagnosing infections, and only slightly worse than physicians who were experts in that field (who had an average correctness of approximately 80%).

MYCIN's knowledge base is small relative to those used by most rules-based systems today; it is on the order of ~500 rules. The science of the generation of these rules is known as "knowledge engineering". MYCIN uses a modification of the method of reasoning called "backward chaining" to search its knowledge base.

The core of MYCIN, its inference engine, is called EMYCIN ("Essential MYCIN"). EMYCIN is the framework for MYCIN, a semi-separate system which could be used to create other rules-based expert systems to face problems similar to what MYCIN faces. In some cases, this can be affected merely by changing the knowledge base.

5.4.6.Logical layout of MYCIN

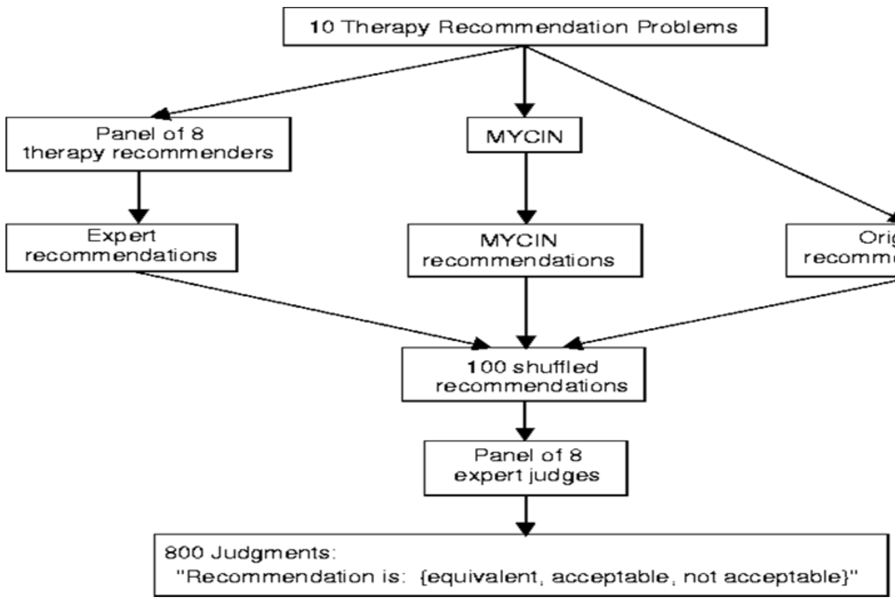

10 Therapy Recommendation Problems
Panel of 8 therapy recommenders
MYCIN
Expert recommendations
MYCIN recommendations
Original recommendations
100 shuffled recommendations
Panel of 8 expert judges
800 Judgments: "Recommendation is: {equivalent, acceptable, not acceptable}"

Figure 5.5Layout of MYCIN

5.4.7.Practical use of MYCIN

MYCIN was never actually used in practice. This wasn't because of any weakness in its performance. As mentioned, in tests it outperformed members of the Stanford medical school faculty. Some observers raised ethical and legal issues related to the use of computers in medicine — if a program gives the wrong diagnosis or recommends the wrong therapy, who should be held responsible? However, the greatest problem, and the reason that MYCIN was not used in routine practice, was the state of technologies for system integration, especially at the time it was developed. MYCIN was a stand-alone system that required a user to enter all relevant information about a patient by typing in response to questions that MYCIN would pose. The program ran on a large time-shared system, available over the early Internet (ARPANet), before personal computers were developed. In the modern era, such a system would be integrated with medical record systems, would extract answers to questions from patient databases, and would be much less dependent on physician entry of information. In the 1970s, a session with MYCIN could easily consume 30 minutes or more—an unrealistic time commitment for a busy clinician.

MYCIN's greatest influence was accordingly its demonstration of the power of its representation and reasoning approach. Rule-based systems in many non-medical domains were developed in the years that followed MYCIN's introduction of the approach. In the 1980s, expert system "shells" were introduced (including one based on MYCIN, known as E-MYCIN (followed by KEE)) and supported the development of expert systems in a wide variety of application areas.

A difficulty that rose to prominence during the development of MYCIN and subsequent complex expert systems has been the extraction of the necessary knowledge for the inference engine to use from the human expert in the relevant fields into the rule base (the so-called knowledge engineering).

5.5. Knowledge acquisition:

Knowledge acquisition involves adding new knowledge to a knowledge base and refining or improving previously acquired knowledge. This process is typically goal-oriented, aiming to expand a system's capabilities or enhance its performance for specific tasks. Knowledge can include facts, rules, concepts, procedures, heuristics, formulas, relationships, statistics, or other valuable information. The sources of this knowledge may include:

- Experts in the relevant domain
- Textbooks
- Technical papers
- Databases
- Reports
- The environment

For the newly acquired knowledge to be effective, it must be meaningfully integrated with existing knowledge, enabling nontrivial inferences. This knowledge should be accurate, non-redundant, consistent, and sufficiently complete to allow reliable reasoning about the intended conclusions.

In other words, Knowledge acquisition can be defined as the situation in which the organization and individuals obtain the required knowledge that helps them accomplish their work efficiently, easily, and at the lowest possible cost.

5.5.1.Knowledge Acquisition process:

Before explaining the steps of acquiring knowledge, we must know what the raw materials of knowledge are?The raw materials of knowledge, as shown in Figure 5.6, refer to the raw materials that you can use to build knowledge. The following paragraphs explain the primary forms of the raw material of knowledge.

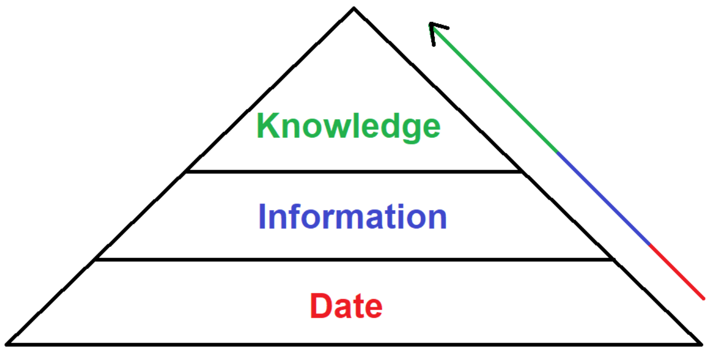


Figure 5.6 Raw Materials of the Knowledge.

1. Data: The data indicates the facts about things. In other words, it describes things. For example, length = 1.70 meters, weight = 80 kg, Old = 35 years. And so on.
2. Information: Data is a collection of individual facts without any relationship between them. Information refers to the process of extracting a relationship between specific facts.When studying and analyzing data, discovering the relationship between that data is information. For example, someone who is 1.70 meters tall would be 35 years old.
3. Knowledge: As a result of the above, the organization obtains knowledge by linking information together and identifying recurring patterns.The company considers this repetition of patterns as the basis for making various decisions. In short, this is the knowledge that organizations are looking for.Knowledge refers to discovering a particular pattern through the study and analysis of information and data. The organization can use these patterns as bases for making various decisions.

5.5.2.Steps of knowledge acquisition process:

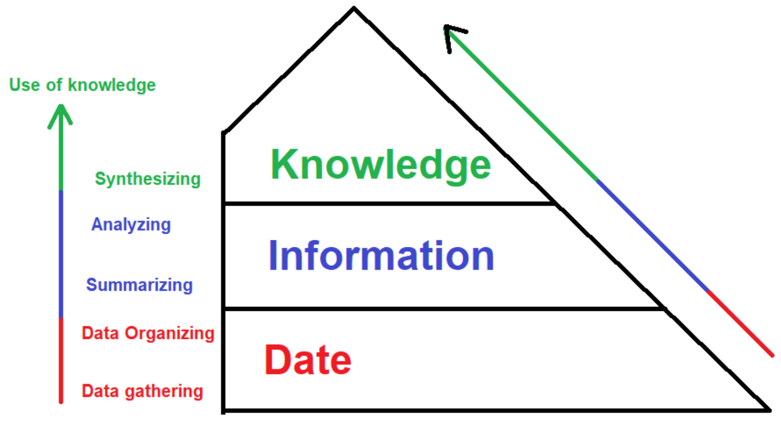


Figure 5.7 Knowledge Acquisition Process

The organization transforms raw materials into knowledge so that they can be used and utilized. In short, the organization creates knowledge using raw materials.

The knowledge acquisition process consists of five main steps, as shown in Figure 5.7. The following paragraphs explain these steps in more detail:

1. <u>Data gathering</u>: Data gatheringis the first step in the knowledge acquisition process. Determining exactly what data will be gathered, how it will be obtained, and where it will come from are all essential phases in this process. Planning ahead is essential before beginning any data gathering process. The kind of data needed and how accurate and precise it is are important considerations in the overall precision and quality of the information that is gained. Consequently, it is essential that this procedure be carried out

carefully. Furthermore, knowledge acquisition might make use of data that has already been gathered for other objectives, including financial statements from organizations or client purchase histories.

2. Data Organizing: The data in the previous step is often not well-organized. Consequently, the knowledge management team must reorganize the data to extract valuable information. For instance, data with related attributes should be collected in one place, such as grouping customer data by geographic area or age group. Usually, the knowledge team keeps this information in specialized databases. Afterward, a variety of information technology techniques are used to methodically arrange the data.
3. Summarizing: In this step, various statistics are extracted from databases. These statistics are presented in tables and graphs in multiple forms.It should be noted here that upon completion of this step, the raw data has been converted into information that can be used further.
4. Analyzing:The information is analyzed, looking for recurring patterns that can be considered a new characteristic or a new knowledge.
   For example, it may be observed that a specific age group exhibits interest in a particular product, such as individuals aged 8-28 displaying a preference for purchasing video games.Another example could be observing that the majority of customers in a certain geographic region are women. Such insights help in understanding customer preferences and behaviours more effectively.
5. Synthesizing:This step depends on joining statistics and patterns and coming out of them with fixed concepts that can be relied upon. These concepts are the knowledge you are looking for.Knowledge management stores these results in rules or laws in the organization’s knowledge databases. So that everyone who wants it can reach it.After this stage, knowledge management completes other knowledge management processes such as storing and distributing knowledge.

5.6. Learning

Learning is the improvement of performance with experience over time.Learning element isthe portion of a learning AI system that decides how to modify theperformance element and implements those modifications.We all learn new knowledge through different methods,depending on the type of material to belearned, the amount of relevant knowledge we already possess, and the environment in which thelearning takes place.

Learning in expert systems involves developing methodologies that allow these systems to improve their performance over time by acquiring new knowledge or refining existing knowledge. Expert systems, which are AI programs that mimic the decision-making abilities of a human expert, can benefit from learning in several ways

5.6.1.Rote learning

- When a computer stores a piece of data, it is performing a rudimentary form of learning.
- In case of data caching, we store computed values so that we do not have to recompute them later.
- When computation is more expensive than recall, this strategy can save a significant amount of time.
- Caching has been used in AI programs to produce some surprising performance improvements.
- Such caching is known as rote learning.
- Rote learning does not involve any sophisticated problem-solving capabilities.
- It shows the need for some capabilities required of complex learning systems such as:
  – Organized Storage of information
  – Generalization

5.6.2.Learning by induction

- Classification is the process of assigning, to a particular input, the name of a class to which it belongs.
- The classes from which the classification procedure can choose can be described in a variety of ways.
- Their definition will depend on the use to which they are put.
- Classification is an important component of many problem-solving tasks.
- Before classification can be done, the classes it will use must be defined:
  - Isolate a set of features that are relevant to the task domain. Define each class by a weighted sum of values of these features. Ex: task is weather prediction, the parameters can be measurements such as rainfall, location of cold fronts etc.
  - Isolate a set of features that are relevant to the task domain. Define each class as a structure composed of these features. Ex: classifying animals, various features can be such things as color, length of neck etc

  The idea of producing a classification program that can evolve its own class definitions is called concept learning or induction

5.6.3.Explanation based learning

- Learning complex concepts using Induction procedures typically requires a substantial number of training instances.
- But people seem to be able to learn quite a bit from single examples.
- We don't need to see dozens of positive and negative examples of fork(chess) positions in order to learn to avoid this trap in the future and perhaps use it to our advantage.
- What makes such single-example learning possible? The answer is knowledge.
- Much of the recent work in machine learning has moved away from the empirical, data intensive approach described in the last section toward this more analytical knowledge intensive approach.
- A number of independent studies led to the characterization of this approach as explanation-base learning(EBL).
- An EBL system attempts to learn from a single example x by explaining why x is an example of the target concept.
- The explanation is then generalized, and then system's performance is improved through the availability of this knowledge.
- We can think of EBL programs as accepting the following as input:
  - A training example
  - A goal concept: A high level description of what the program is supposed to learn
  - An operational criterion- A description of which concepts are usable.
  - A domain theory: A set of rules that describe relationships between objects and actions in a domain.
- From this EBL computes a generalization of the training example that is sufficient to describe the goal concept, and also satisfies the operationality criterion.
- Explanation-based generalization (EBG) is an algorithm for EBL and has two steps: (1) explain, (2) generalize
- During the explanation step, the domain theory is used to prune away all the unimportant aspects of the training example with respect to the goal concept. What is left is an explanation of why the training example is an instance of the goal concept. This explanation is expressed in terms that satisfy the operationality criterion.
- The next step is to generalize the explanation as far as possible while still describing the goal concept.

6. Machine Learning

6.1. Introduction to machine learning

Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms that allow computers to learn from and make decisions based on data. It involves the use of statistical techniques to enable machines to improve their performance on tasks through experience. Machine learning is the process of using data to train models that can make predictions or decisions without being explicitly programmed for specific tasks. The purpose is to allow computers to autonomously learn and adapt to new data.

In traditional programming, a computer takes data and a program as inputs to produce the desired output. The programmer explicitly defines the rules and logic in the program, which the computer follows to process the data and generate the output. In contrast, machine learning reverses this process. Here, the computer takes data and the expected output as inputs and learns to produce the program (model) that can generate similar outputs from new, unseen data. The learning process involves finding patterns and relationships in the data, allowing the computer to create a model that can make predictions or decisions without being explicitly programmed with specific rules. This approach enables computers to handle more complex and dynamic problems where traditional programming might fall short.

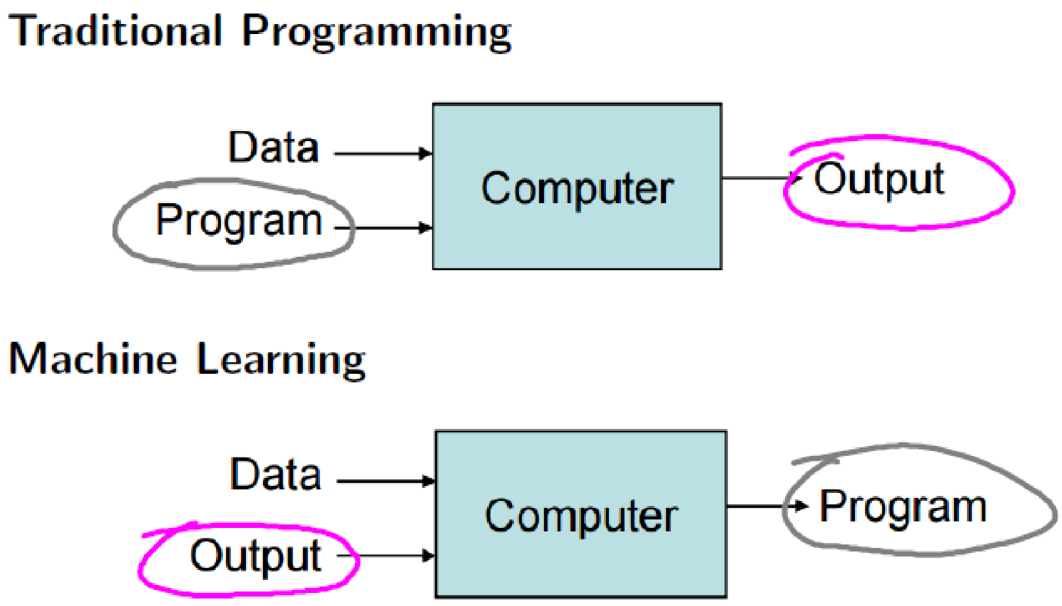


Fig. 1 Traditional programming vs. Machine learning

At the broadest level, AI represents the capability of machines to perform tasks that typically require human intelligence, operating autonomously without human intervention. Within this broad field, ML is a specialized subset that employs statistical algorithms to learn patterns from data over time, thereby improving the machine's performance on specific tasks. Nested within ML, DL is a further specialized subset that uses neural networks with multiple layers (often referred to as deep neural networks) to filter and analyze data. This layered approach allows DL models to handle more complex data and achieve higher levels of accuracy in tasks such as image and speech recognition.

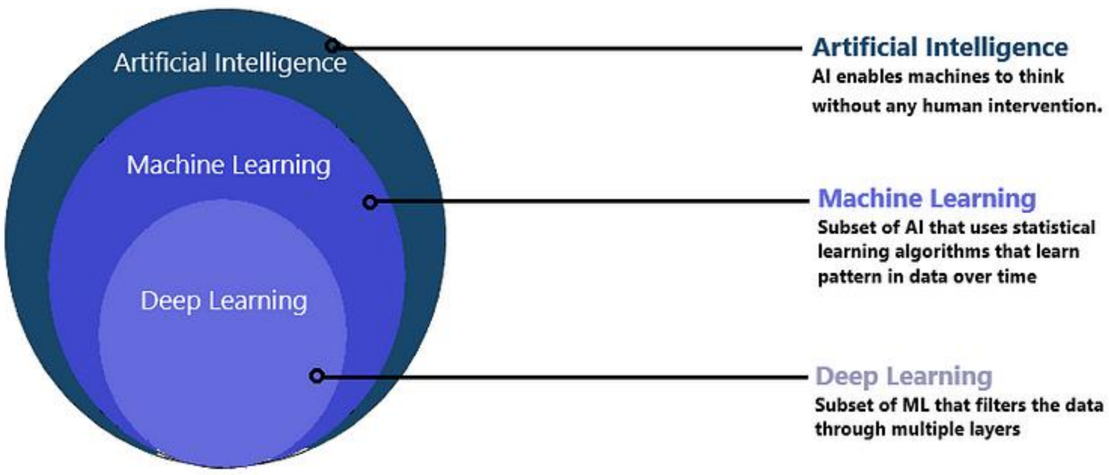

Artificial Intelligence
Machine Learning
Deep Learning
Artificial Intelligence
AI enables machines to think without any human intervention.
Machine Learning
Subset of AI that uses statistical learning algorithms that learn pattern in data over time
Deep Learning
Subset of ML that filters the data through multiple layers

Figure 6.1 AI vs. ML vs. DL

6.1.1.A metaphorical sense

Machine learning can be compared to farming or gardening in a metaphorical sense. Think of the algorithms as seeds that you plant. Just as seeds need nutrients to grow, algorithms require data to learn and develop. You, as the gardener, play a crucial role in this process, tending to the garden by selecting the right seeds (algorithms) and providing the necessary nutrients (data). With proper care and attention, these seeds grow into plants, which represent the programs or models that are capable of making predictions or decisions. This analogy emphasizes the importance of choosing the right algorithms and feeding them with high-quality data, along with the active role of the gardener in nurturing the growth process to yield fruitful results.

6.1.2.Key Concepts

- **Data**: The raw input for machine learning models, which can be in the form of numbers, text, images, or other formats.
- **Features**: Individual measurable properties or characteristics of the data.
- **Labels**: The outcome or target variable that the model aims to predict (primarily in supervised learning).
- **Training**: The process of feeding data into an algorithm to help it learn patterns.
- **Testing**: The phase where the learned model is evaluated using a separate dataset to assess its performance.
- **Algorithms**: The specific methods used to find patterns in data and make predictions.

6.1.3.Types of Machine Learning

- **Supervised Learning:**
  - **Description:** In supervised learning, the model is trained on a labeled dataset, which means that each training example is paired with an output label. The model learns to make predictions or decisions by finding patterns in the input-output pairs.
  - **Examples:**
    - Classification: Identifying spam emails.
    - Regression: Predicting house prices.
    - **Neural Networks:** Feedforward neural networks for tasks like handwriting recognition.

        - **CNNs:** Convolutional neural networks for image classification tasks such as recognizing objects in images.
- **Unsupervised Learning:**
    - **Description:** Unsupervised learning involves training the model on data without labeled responses. The model tries to identify patterns and relationships within the data.
    - **Examples:**
        - Clustering: Grouping customers based on purchasing behavior.
        - Dimensionality Reduction: Reducing the number of features in a dataset while preserving its essential characteristics.
        - **Neural Networks:** Autoencoders for anomaly detection or data compression.
        - **CNNs:** Unsupervised feature learning for image clustering.
- **Reinforcement Learning:**
    - **Description:** In reinforcement learning, an agent learns to make decisions by performing actions in an environment to achieve a goal. The agent receives feedback in the form of rewards or penalties and adjusts its actions to maximize the cumulative reward over time.
    - **Examples:**
        - Game Playing: AlphaGo.
        - Robotic Control: Robots learning to navigate obstacles.
        - **Neural Networks:** Deep Q-Networks (DQN) for playing video games.
        - **CNNs:** Using CNNs in the state representation of reinforcement learning agents, especially in visual environments.
- **Semi-Supervised Learning:**
    - **Description:** Semi-supervised learning is a hybrid approach that uses both labeled and unlabeled data for training. It is particularly useful when obtaining a fully labeled dataset is expensive or time-consuming.
    - **Examples:**
        - Image Recognition: Labeling a few images in a large dataset and using the model to infer labels for the rest.
        - Text Classification.
        - **Neural Networks:** Semi-supervised learning with generative adversarial networks (GANs).
        - **CNNs:** Semi-supervised image classification using CNNs with a mix of labeled and unlabeled data.

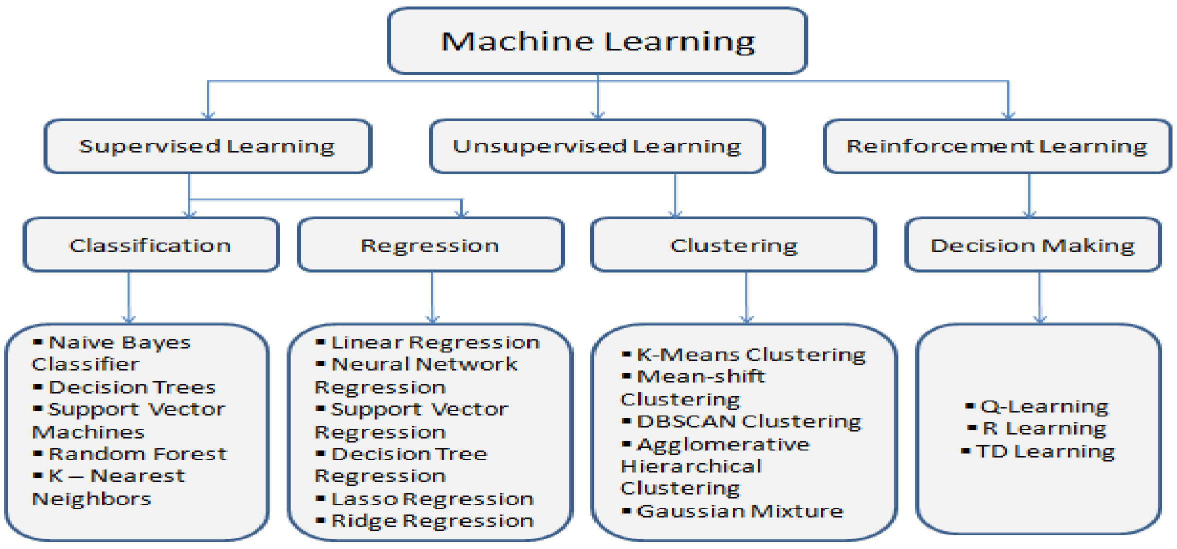


Figure 6.2Taxonomy of types of ML

6.2. Extended Types as Techniques of Machine Learning

- **Self-Supervised Learning:**
  - **Description:** Self-supervised learning is a form of unsupervised learning where the data provides its own supervision. The model generates its own labels from the input data, enabling it to learn representations without external labels.
  - **Examples:**
    - Language Models: Predicting the next word in a sentence.
    - Computer Vision: Predicting the rotation angle of an image.
    - **Neural Networks:** Self-supervised learning with neural networks for natural language processing tasks.
    - **CNNs:** Using CNNs for self-supervised learning tasks such as colorization of black and white images.
- **Transfer Learning:**
  - **Description:** Transfer learning involves taking a pre-trained model on a large dataset and fine-tuning it on a smaller, task-specific dataset. This approach leverages the knowledge gained from one domain to improve performance in another.
  - **Examples:**
    - Using a pre-trained image recognition model for medical image analysis.
    - Fine-tuning a language model for sentiment analysis.
    - **Neural Networks:** Fine-tuning pre-trained neural networks for specific tasks.
    - **CNNs:** Applying transfer learning with pre-trained CNNs like VGG, ResNet for specific image classification problems.

6.3. Examples and Scope of Machine Learning Applications

- **Email Spam Filtering**:
  - **Problem**: Identify and filter out spam emails from a user's inbox.
  - **Approach**: Use a supervised learning algorithm such as Naive Bayes, which can be trained on a dataset of labeled emails (spam and not spam). The model learns patterns and keywords associated with spam and can then predict whether new emails are spam based on these learned patterns.
- **Image Recognition**:
  - **Problem**: Automatically identify objects within an image.
  - **Approach**: Use a convolutional neural network (CNN) that has been trained on a large dataset of labeled images (e.g., identifying cats, dogs, cars). The model extracts feature from images and learns to recognize objects based on their visual characteristics.
- **Customer Segmentation**:
  - **Problem**: Group customers into distinct segments based on their behavior and demographics.
  - **Approach**: Apply clustering algorithms like K-Means on customer data, including purchase history, age, location, and preferences. The model finds patterns and groups similar customers together, which helps in targeted marketing strategies.
- **Predictive Maintenance**:
  - **Problem**: Predict when equipment is likely to fail so that maintenance can be performed proactively.
  - **Approach**: Use a combination of time-series analysis and machine learning algorithms (e.g., random forests) on sensor data from machinery. By identifying patterns that precede failures, the model can predict and alert for maintenance needs before a breakdown occurs.

- **Speech Recognition**:
  - **Problem**: Convert spoken language into text.
  - **Approach**: Use recurrent neural networks (RNNs) or more advanced models like transformers, trained on large datasets of audio recordings and their corresponding transcriptions. These models learn to recognize speech patterns and convert spoken words into text.
- **Recommendation Systems**:
  - **Problem**: Recommend products, movies, or content to users based on their preferences.
  - **Approach**: Collaborative filtering techniques like matrix factorization or content-based filtering that analyze user behavior and item characteristics. For example, Netflix recommends shows based on users' viewing history and ratings.
- **Medical Diagnosis**:
  - **Problem**: Assist in diagnosing diseases based on patient data.
  - **Approach**: Train models on medical records, imaging data, and other diagnostic information. For example, using deep learning to analyze MRI scans and detect early signs of brain tumors.

6.4. Limitations of ML

- **Data Dependency:**
  - **Challenge:** ML models require large volumes of high-quality data to train effectively.
  - **Impact:** Insufficient or poor-quality data can lead to inaccurate models and unreliable predictions.
- **Overfitting:**
  - **Challenge:** ML models can become too tailored to the training data, capturing noise instead of underlying patterns.
  - **Impact:** Overfitted models perform well on training data but poorly on new, unseen data.
- **Complexity and Interpretability:**
  - **Challenge:** Some ML models, particularly deep learning models, are complex and difficult to interpret.
  - **Impact:** Lack of interpretability can hinder trust and understanding, especially in critical applications like healthcare.
- **Bias and Fairness:**
  - **Challenge:** ML models can inadvertently learn and perpetuate biases present in the training data.
  - **Impact:** Biased models can lead to unfair and discriminatory outcomes, affecting decision-making in areas like hiring and lending.
- **Resource Intensive:**
  - **Challenge:** Training and deploying ML models, especially deep learning models, require significant computational resources and time.
  - **Impact:** High costs associated with computational resources can limit accessibility and scalability.
- **Ethical and Privacy Concerns:**
  - **Challenge:** The use of ML raises ethical issues related to privacy, consent, and data security.
  - **Impact:** Misuse of ML technologies can lead to privacy breaches and ethical dilemmas, necessitating robust governance and regulation.
- **Generalization:**
  - **Challenge:** ML models may struggle to generalize from training data to real-world scenarios, especially in dynamic environments.
  - **Impact:** Limited generalization capability can reduce the effectiveness of models in practical applications.

6.5. Probability, statistics, and linear algebra for machine learning

Understanding and applying probability, statistics, and linear algebra is crucial for developing and implementing machine learning models. These mathematical foundations provide the necessary tools for analyzing data, modeling relationships, making predictions, and optimizing solutions.

By leveraging probability, we handle uncertainty and make informed decisions. Statistics helps us analyze and infer from data, ensuring our models are valid and reliable. Linear algebra enables efficient computation and manipulation of data, essential for working with large datasets and complex models. Together, these disciplines form the bedrock upon which machine learning is built, driving innovations across various applications and industries.

- **Probability**
  - **Scope:** Probability is fundamental to machine learning, providing the theoretical foundation for understanding and modeling uncertainty, making predictions, and drawing inferences from data.
  - **Examples:**
    - **Bayesian Networks:**
      - **Scope:** Used for probabilistic inference in complex models.
      - **Example:** Diagnosing diseases based on symptoms and medical history.
    - **Markov Decision Processes (MDPs):**
      - **Scope:** Used for modeling decision-making in environments with stochastic outcomes.
      - **Example:** Optimizing policies in reinforcement learning.
    - **Probability Distributions:**
      - **Scope:** Describe the likelihood of different outcomes.
      - **Example:** Gaussian distributions for modeling continuous data.
    - **Expectation-Maximization (EM) Algorithm:**
      - **Scope:** Used for finding maximum likelihood estimates in models with latent variables.
      - **Example:** Clustering data with Gaussian Mixture Models (GMMs).
    - **Naive Bayes Classifier:**
      - **Scope:** A simple probabilistic classifier based on Bayes' theorem.
      - **Example:** Email spam filtering.
- **Statistics**
  - **Scope:** Statistics is essential for analyzing data, estimating model parameters, evaluating models, and validating results. It provides the tools for hypothesis testing, inference, and descriptive analysis.
  - **Examples:**
    - **Descriptive Statistics:**
      - **Scope:** Summarize and describe features of a dataset.
      - **Example:** Mean, median, standard deviation of data.
    - **Inferential Statistics:**
      - **Scope:** Make inferences about populations from samples.
      - **Example:** Confidence intervals, hypothesis tests.
    - **Regression Analysis:**
      - **Scope:** Model relationships between variables.
      - **Example:** Linear regression for predicting house prices.
    - **Principal Component Analysis (PCA):**
      - **Scope:** Reduce dimensionality of data while preserving variance.

            - **Example:** Data compression and visualization.
        - **Hypothesis Testing:**
            - **Scope:** Test assumptions and theories about data.
            - **Example:** t-tests for comparing means of two groups.
- **Linear Algebra**
    - **Scope:** Linear algebra is the backbone of many machine learning algorithms, providing tools for handling high-dimensional data, matrix operations, and transformations.
    - **Examples:**
        - **Vector and Matrix Operations:**
            - **Scope:** Fundamental operations for manipulating data.
            - **Example:** Dot product, matrix multiplication used in neural networks.
        - **Singular Value Decomposition (SVD):**
            - **Scope:** Factorize matrices to identify important features.
            - **Example:** Recommender systems for latent factor models.
        - **Eigenvalues and Eigenvectors:**
            - **Scope:** Analyze properties of matrices.
            - **Example:** PCA for reducing dimensionality.
        - **Linear Transformations:**
            - **Scope:** Transform data into different spaces.
            - **Example:** Transforming input features in linear regression.
        - **Optimization:**
            - **Scope:** Solve for parameters that minimize or maximize an objective function.
            - **Example:** Gradient descent for training machine learning models.

6.6. Data preprocessing in ML

Data preprocessing is a critical step that ensures the data fed into machine learning models is clean, consistent, and suitable for analysis. By addressing issues such as missing values, inconsistencies, and scaling, data preprocessing helps improve the accuracy and reliability of machine learning models. Proper preprocessing steps include data cleaning, integration, transformation, reduction, discretization, and splitting, each contributing to the overall effectiveness of the machine learning process.

- **Data Cleaning**
    - **Scope:** Data cleaning involves handling missing values, correcting errors, and removing inconsistencies in the data.
    - **Examples:**
        - **Handling Missing Values:**
            - **Techniques:** Imputation (filling missing values with mean, median, mode), removal of missing entries, or using algorithms that support missing values.
            - **Example:** Filling missing ages in a dataset with the median age.
        - **Removing Duplicates:**
            - **Techniques:** Identifying and removing duplicate rows to avoid bias in the model.
            - **Example:** Removing duplicate entries in a customer database.
        - **Correcting Errors:**
            - **Techniques:** Identifying and correcting erroneous data points.
            - **Example:** Fixing incorrect data entries, such as negative values for age.

- **Data Integration**
  - **Scope:**Data integration combines data from different sources to provide a unified view.
  - **Examples:**
    - **Combining Multiple Datasets:**
      - **Techniques**: Merging, joining, and concatenating datasets based on common keys.
      - **Example:** Integrating customer data from multiple departments (e.g., sales and support).
    - **Schema Integration:**
      - **Techniques:** Aligning different schemas to a common format.
      - **Example:** Standardizing different date formats into a single format.

- **Data Transformation**
  - **Scope:**Data transformation involves converting data into a suitable format or structure for analysis.
  - **Examples:**
    - **Scaling and Normalization:**
      - **Techniques:** Min-Max scaling, Z-score normalization, log transformation.
      - **Example:** Scaling features to a range of 0 to 1 for algorithms like k-NN that are sensitive to feature magnitudes.
    - **Encoding Categorical Variables:**
      - **Techniques:** One-hot encoding, label encoding, binary encoding.
      - **Example:** Converting categorical features like "color" (red, green, blue) into numerical format.
    - **Feature Engineering:**
      - **Techniques:** Creating new features based on existing ones.
      - **Example:** Extracting day, month, and year from a date feature.
- **Data Reduction**
  - **Scope:**Data reduction techniques are used to reduce the volume of data while retaining its essential characteristics.
  - **Examples:**
    - **Dimensionality Reduction:**
      - **Techniques:** Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), t-Distributed Stochastic Neighbor Embedding (t-SNE).
      - **Example:** Using PCA to reduce the number of features in an image dataset.
    - **Sampling:**
      - **Techniques:** Random sampling, stratified sampling.
      - **Example:** Selecting a representative subset of a large dataset for quicker analysis.
    - **Aggregation:**
      - **Techniques:** Summarizing data by aggregating individual data points.
      - **Example:** Aggregating hourly sales data to daily sales data.

- **Data Discretization**
  - **Scope:**Data discretization involves converting continuous data into discrete bins or intervals.
  - **Examples:**
    - **Binning:**

            - **Techniques:** Equal-width binning, equal-frequency binning, clustering-based binning.
            - **Example:** Discretizing age into bins like 0-18, 19-35, 36-60, 61+.
- **Data Splitting**
    - **Scope:**Data splitting involves dividing the dataset into training, validation, and test sets.
    - **Examples:**
        - **Train-Test Split:**
            - **Techniques:** Randomly splitting data into training and testing sets.
            - **Example:** Using 80% of the data for training and 20% for testing.
        - **Cross-Validation:**
            - **Techniques:** k-fold cross-validation, stratified cross-validation.
            - **Example:** Using 10-fold cross-validation to evaluate model performance.
- **Data Augmentation**
    - **Scope:** Data augmentation involves artificially increasing the size of a training dataset by creating modified versions of existing data.
    - **Examples:**
        - **Image Augmentation:**
            - **Techniques:** Applying transformations like rotation, scaling, flipping, and adding noise to images.
            - **Example:** Enhancing an image dataset for a convolutional neural network by rotating images to create more training samples.
        - **Text Augmentation:**
            - **Techniques:** Synonym replacement, random insertion, and back-translation.
            - **Example:** Expanding a text dataset by replacing words with their synonyms to create variations of the original sentences.
        - **Audio Augmentation:**
            - **Techniques:** Adding noise, changing pitch and speed, and shifting time.
            - **Example:** Increasing the diversity of an audio dataset by applying pitch changes and time shifts to original recordings.

6.7. Data Augmentation in Machine Learning

Data augmentation is a technique used to increase the diversity and quantity of training data without actually collecting new data. It is particularly useful in domains like computer vision, natural language processing, and audio processing. By creating modified versions of existing data, data augmentation helps prevent overfitting, improves model robustness, and enhances generalization.

Data augmentation is a powerful technique to artificially expand the size of training datasets, which can lead to better model performance and generalization. By applying various transformations to the original data, we create new, diverse examples that help the model learn more robustly. This is particularly useful in domains where data collection is expensive or time-consuming, such as computer vision, natural language processing, and audio processing. Through data augmentation, we can mitigate overfitting, improve model accuracy, and make models more resilient to variations in real-world data.

- **Image Augmentation:** Image augmentation involves applying various transformations to images to create new training samples.
    - **Techniques:**
        - **Rotation:** Rotating images by a certain angle.
        - **Scaling:** Resizing images.

        - **Flipping:** Flipping images horizontally or vertically.
        - **Translation:** Shifting images in different directions.
        - **Adding Noise:** Introducing random noise to images.
        - **Cropping:** Randomly cropping parts of images.
        - **Color Jittering:** Altering the brightness, contrast, saturation, and hue of images.
    - **Example:**

Input Image	**Rotation Image**	**Flipped Images**	**Total Images**
[[1, 2], [3, 4]]	[[3, 1], [4, 2]]	[[2, 1], [4, 3]]	3
[[5, 6], [7, 8]]	[[7, 5], [8, 6]]	[[6, 5], [8, 7]]	3
[[9, 10], [11, 12]]	[[11, 9], [12, 10]]	[[10, 9], [12, 11]]	3

Text Augmentation: Text augmentation creates variations in text data to increase the size of the dataset.

- **Techniques:**
    - **Synonym Replacement:** Replacing words with their synonyms.
    - **Random Insertion:** Adding random words into sentences.
    - **Back-Translation:** Translating text to another language and back to the original language.
    - **Random Swap:** Swapping the positions of words in a sentence.
    - **Random Deletion:** Removing random words from sentences.
- **Example:**
    - **Original Sentence:** "The quick brown fox jumps over the lazy dog."
    - **Synonym Replacement:** "The fast brown fox jumps over the lazy dog."

- **Audio Augmentation:** Audio augmentation involves modifying audio signals to create new training samples.

    - **Techniques:**
        - **Adding Noise:** Introducing background noise to audio signals.
        - **Time Shifting:** Shifting audio signals in time.
        - **Pitch Shifting:** Changing the pitch of audio signals.
        - **Speed Changing:** Adjusting the speed of audio playback.
        - **Volume Adjustment:** Increasing or decreasing the volume.
    - **Example:**
        - **Original Audio Signal:** [0.1, 0.3, 0.5, 0.4, 0.2]
        - **Added Noise:** [0.1, 0.3, 0.5, 0.4, 0.2] + [0.01, -0.02, 0.03, -0.01, 0.02] = [0.11, 0.28, 0.53, 0.39, 0.22]

6.8. Data Normalization in Machine Learning

Normalization is a key step in data preprocessing that transforms features to a similar scale, improving the performance and convergence of machine learning algorithms. Techniques like Min-Max Scaling, Z-Score Normalization, Max Abs Scaling, and Robust Scaling cater to different requirements and data characteristics. By applying normalization, we ensure that no single feature dominates the model, leading to better and more reliable results.

Why Normalize Data?

- **Improved Convergence Rate:**
  - Algorithms like gradient descent converge faster when the features are on a similar scale.
- **Better Performance:**
  - Distance-based algorithms like k-NN and SVM perform better when features are normalized, as it ensures that all features contribute equally to the distance calculations.
- **Reduced Sensitivity to Scale:**
  - Models become less sensitive to the scale of features, preventing features with larger ranges from dominating the learning process.

Techniques for Normalization

- **Min-Max Scaling:**
  - Scales the data to a fixed range, typically [0, 1] or [-1, 1].
  - Formula: $X' = \frac{X - X_{min}}{X_{max} - X_{min}}$
- **Normalization (Standardization):**
  - Scales the data based on the mean (μ) and standard deviation (σ) of the features.
  - Formula: $X' = \frac{X - \mu}{\sigma}$
- **Max Abs Scaling:**
  - Scales each feature by its maximum absolute value.
  - Formula: $X' = \frac{X}{|X_{max}|}$
- **Robust Scaling:**
  - Scales the data using statistics that are robust to outliers, such as the median and the interquartile range (IQR).
  - Formula: $X' = \frac{X - \text{median}}{IQR}$

Numerical Example: Min-Max Scaling

Consider a dataset with the following feature values:

- Feature 1: [50, 20, 30, 80, 40]
- Feature 2: [200, 300, 400, 500, 600]

We aim to normalize these features using Min-Max Scaling to a range of [0, 1].

Step 1: Calculate the Minimum and Maximum Values

- Feature 1:$X_{\text{min}} = 20, X_{max} = 80$
- Feature 2: $X_{min} = 200, X_{max} = 600$

Step 2: Apply the Min-Max Scaling Formula

For each value X in the feature, the normalized value X′ is calculated as: $X' = \frac{X - X_{min}}{X_{max} - X_{min}}$

Normalized Feature 1:

- $X_1' = \frac{50-20}{80-20} = \frac{30}{60} = 0.5$

- $X_2' = \frac{20-20}{80-20} = \frac{0}{60} = 0$
- $X_3' = \frac{30-20}{80-20} = \frac{10}{60} \approx 0.167$
- $X_4' = \frac{80-20}{80-20} = \frac{60}{60} = 1$
- $X_5' = \frac{40-20}{80-20} = \frac{20}{60} \approx 0.333$

Normalized Feature 2:

- $X_1' = \frac{200-200}{600-200} = \frac{0}{400} = 0$
- $X_2' = \frac{300-200}{600-200} = \frac{100}{400} = 0.25$
- $X_3' = \frac{400-200}{600-200} = \frac{200}{400} = 0.5$
- $X_4' = \frac{500-200}{600-200} = \frac{300}{400} = 0.75$
- $X_5' = \frac{600-200}{600-200} = \frac{400}{400} = 1$

Normalized Data:

- Feature 1: [0.5, 0, 0.167, 1, 0.333]
- Feature 2: [0, 0.25, 0.5, 0.75, 1]

6.9. Hypothesis Function and Hypothesis Testing

Hypothesis functions in machine learning represent the models used to make predictions from input data. Hypothesis testing, on the other hand, is a statistical method used to infer the validity of a hypothesis about a population based on sample data. By following the steps of hypothesis testing, we can make informed decisions about the effects or differences observed in data. This process is fundamental in validating assumptions and ensuring the reliability of conclusions drawn from data analysis.

6.9.1. Types of Hypothesis Functions:

- **Linear Hypothesis Function:**
  - Used in linear regression.
  - Formula: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
  - Example: Predicting house prices based on features like size, number of bedrooms, and location.
- **Logistic Hypothesis Function:**
  - Used in logistic regression for binary classification.
  - Formula: $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
  - Example: Predicting whether an email is spam or not based on features like word frequency.
- **Non-Linear Hypothesis Function:**
  - Used in polynomial regression or neural networks.
  - Example: Predicting complex patterns in data where a linear relationship is insufficient.

Hypothesis Testing:

Hypothesis testing is a statistical method used to make inferences or draw conclusions about a population based on a sample of data. It involves formulating a hypothesis, collecting data, and determining whether the evidence supports the hypothesis.

Key Steps in Hypothesis Testing:

- **Formulate the Hypotheses:**
  - **Null Hypothesis (H_0):** The default assumption that there is no effect or difference.
  - **Alternative Hypothesis (H_1):** The assumption that there is an effect or difference.
- **Choose a Significance Level (α):**
  - The probability of rejecting the null hypothesis when it is actually true.
  - Common significance levels are 0.05, 0.01, and 0.10.
- **Collect and Analyze Data:**
  - Compute a test statistic from the sample data.
  - Compare the test statistic to a critical value or use a p-value to determine the strength of evidence against the null hypothesis.
- **Make a Decision:**
  - Reject H_0 if the test statistic exceeds the critical value or if the p-value is less than α.
  - Fail to reject H_0 if the test statistic does not exceed the critical value or if the p-value is greater than α.

6.9.2.Numerical Example: Hypothesis Testing

Example Scenario: Suppose a company claims that its new training program increases the average test scores of employees. The current average test score is 70. The company conducts a pilot study with 10 employees who undergo the training, and their test scores are recorded.

Step 1: Formulate the Hypotheses

- Null Hypothesis (H_0): The average test score after training is 70. (μ=70)
- Alternative Hypothesis (H_1): The average test score after training is greater than 70. (μ>70)

Step 2: Choose a Significance Level

- Significance Level (α): 0.05

Step 3: Collect and Analyze Data

- Sample Data: [75, 78, 74, 70, 82, 73, 77, 76, 71, 79]
- Sample Mean ($\bar{x}$): $\frac{75+78+74+70+82+73+77+76+71+79}{10} = 75.5$
- Population Mean (μ_0): 70
- Sample Standard Deviation (s): Calculate using the formula: $s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$
  - $s \approx 3.56$
- Test Statistic (t): Use the t-test formula for a single sample: $t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} = \frac{75.5-70}{3.56/\sqrt{10}} \approx 5.00$

Step 4: Make a Decision

- Critical Value: For a one-tailed t-test with 9 degrees of freedom (n-1) at α=0.05, the critical value is approximately 1.833.
- Decision Rule: Reject H_0 if t > 1.833.
- Since 5.00 > 1.833, we reject the null hypothesis.

6.10. Data Distributions

Understanding data distributions and testing for them is essential in machine learning and statistics. Different distributions describe different types of data, and testing these distributions helps validate assumptions about the data. Techniques like the Kolmogorov-Smirnov test, Shapiro-Wilk test, Chi-Square goodness-of-fit test, and Anderson-Darling test provide robust methods for assessing whether data follows specific distributions. By properly understanding and testing data distributions, we can make better decisions regarding data preprocessing, model selection, and result interpretation.

- **Normal Distribution:**
  - **Description:** Symmetrical, bell-shaped distribution where most of the data points cluster around the mean.
  - **Properties:** Mean, median, and mode are equal; defined by mean (μ) and standard deviation (σ).
  - **Example:** Heights of people, IQ scores.
- **Uniform Distribution:**
  - **Description:** All outcomes are equally likely within a given range.
  - **Properties:** Defined by minimum (a) and maximum (b).
  - **Example:** Rolling a fair die.
- **Binomial Distribution:**
  - **Description:** Discrete distribution representing the number of successes in a fixed number of independent Bernoulli trials.
  - **Properties:** Defined by the number of trials (n) and probability of success (p).
  - **Example:** Number of heads in 10 coin flips.
- **Poisson Distribution:**
  - **Description:** Discrete distribution representing the number of events occurring within a fixed interval of time or space.
  - **Properties:** Defined by the rate (λ).
  - **Example:** Number of phone calls received by a call center in an hour.
- **Exponential Distribution:**
  - **Description:** Continuous distribution representing the time between events in a Poisson process.
  - **Properties:** Defined by the rate (λ).
  - **Example:** Time between arrivals of buses.
- **Log-Normal Distribution:**
  - **Description:** Distribution of a variable whose logarithm is normally distributed.
  - **Properties:** Skewed right; defined by mean and standard deviation of the logarithm.
  - **Example:** Income distribution.

6.10.1. Testing Data Distributions

Statistical tests can be employed to understand a dataset's distribution. These tests help determine if the data follows a specific distribution.

- **Kolmogorov-Smirnov (K-S) Test:**
  - **Purpose:** Test if a sample follows a specified distribution.
  - **Usage:** Applicable for continuous distributions like normal or exponential.
  - **Example:** Testing if a dataset follows a normal distribution
- **Chi-Square Goodness-of-Fit Test:**
  - **Purpose:** Test if a sample follows a specified categorical distribution.
  - **Usage:** Suitable for discrete distributions like binomial or Poisson.
  - **Example:** Testing if the observed frequency distribution of categorical data matches an expected distribution.
- **Shapiro-Wilk Test:**

- **Purpose:** Test the null hypothesis that a sample comes from a normally distributed population.
- **Usage:** Specific to normal distribution testing.
- **Example:** Testing normality of residuals in regression analysis.

6.11. Machine Learning Models: Supervised and Unsupervised Learning

Supervised Learning involves training models on labeled data to predict outcomes for new data. Key tasks include regression and classification, and examples include linear regression and logistic regression.

Unsupervised Learning involves finding hidden patterns in unlabeled data. Key tasks include clustering and dimensionality reduction, and examples include K-means clustering and PCA.

6.11.1. Supervised Learning

Definition: Supervised learning involves training a model on a labeled dataset, which means that each training example is paired with an output label. The model learns to map inputs to the correct outputs and is then able to make predictions on new, unseen data.

Types of Supervised Learning:

- **Regression:**
  - **Purpose:** Predict continuous values.
  - **Examples:** Linear regression, polynomial regression, support vector regression.
- **Classification:**
  - **Purpose:** Predict categorical values.
  - **Examples:** Logistic regression, decision trees, random forests, support vector machines (SVM), neural networks.

Numerical Example: Linear Regression

Example: We have a dataset of house prices based on their size (in square feet).

Size (sq ft)	Price ($)
1500	300,000
1700	340,000
2000	400,000
2100	420,000
2500	500,000

Objective: Predict the price of a house based on its size using linear regression.

Step 1: Formulate the Hypothesis Function

- Hypothesis function: $h_\theta(x) = \theta_0 + \theta_1 x$

Step 2: Define the Cost Function

- Cost function (Mean Squared Error): $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$

Step 3: Minimize the Cost Function using Gradient Descent

- Update rule: $\theta_j = \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$

Step 4: Compute the Partial Derivatives

- Partial derivative with respect to θ: $\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)$
- Partial derivative with respect to θ1: $\frac{\partial J(\theta)}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}\left((h_\theta(x_i) - y_i)x_i\right)$

In simple we can find:

The normal equation to find the optimal parameters θ is given by:

$$\theta = (X^T X)^{-1} X^T y$$

To solve this using linear regression, you first need to create a design matrix X and an output vector y. The design matrix includes a column of 1s to account for the intercept term (θ_0):

X = [[1, 1500],y = [300000, 340000, 400000, 420000, 500000]

[1, 1700],

[1, 2000],

[1, 2100],

[1, 2500]]

Compute $X^T X$:

First, compute the transpose of X (denoted as X^T) and then multiply it by X:

$$X^T X = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1500 & 1700 & 2000 & 2100 & 2500 \end{pmatrix} \times \begin{pmatrix} 1 & 1500 \\ 1 & 1700 \\ 1 & 2000 \\ 1 & 2100 \\ 1 & 2500 \end{pmatrix}$$

The result of this multiplication is:

$$X^T X = \begin{pmatrix} 5 & 9800 \\ 9800 & 19804000 \end{pmatrix}$$

2. **Compute $X^T y$:**

Next, compute the multiplication of X^T and y:

$$X^T y = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1500 & 1700 & 2000 & 2100 & 2500 \end{pmatrix} \times \begin{pmatrix} 300000 \\ 340000 \\ 400000 \\ 420000 \\ 500000 \end{pmatrix}$$

The result of this multiplication is:

$$X^T y = \begin{pmatrix} 1960000 \\ 3968000000 \end{pmatrix}$$

3. **Compute the Inverse of $X^T X$:**

   Next, compute the inverse of the matrix $X^T X$:

$$(X^T X)^{-1} = \begin{pmatrix} 5 & 9800 \\ 9800 & 19804000 \end{pmatrix}^{-1}$$

The determinant of $X^T X$ is calculated first:

$$\det = (5 \times 19804000) - (9800 \times 9800) = 99020000 - 96040000 = 2980000$$

Now, the inverse is:

$$(X^T X)^{-1} = \frac{1}{2980000} \begin{pmatrix} 19804000 & -9800 \\ -9800 & 5 \end{pmatrix}$$

Which simplifies to:

$$(X^T X)^{-1} = \begin{pmatrix} 6.6497 & -0.0033 \\ -0.0033 & 1.6785 \times 10^{-6} \end{pmatrix}$$

4. **Compute θ:**

   Finally, multiply$(X^T X)^{-1}$ by $X^T y$ to get θ:

$$\theta = \begin{pmatrix} 6.6497 & -0.0033 \\ -0.0033 & 1.6785 \times 10^{-6} \end{pmatrix} \times \begin{pmatrix} 1960000 \\ 3968000000 \end{pmatrix}$$

Calculate θ_0 and θ_1:

$$\theta_0 = (6.6497 \times 1960000) + (-0.0033 \times 3968000000) = 13031432 - 13034400 \\ = -2968$$

$$\theta_1 = (-0.0033 \times 1960000) + (1.6785 \times 10 - 6 \times 3968000000) = -6468 + 6664 = 196$$

Making Predictions

To predict the price of a house with a size of 1800 sq ft:

$$h_\theta(1800) = -2968 + 196 \times 1800 = -2968 + 352800 = 349832$$

6.11.2. Unsupervised Learning

Definition: Unsupervised learning involves training a model on a dataset without labeled responses. The model tries to find hidden patterns or intrinsic structures in the input data.

Types of Unsupervised Learning:

- **Clustering:**
  - **Purpose:** Group similar data points together.
  - **Examples:** K-means clustering, hierarchical clustering, DBSCAN.
- **Dimensionality Reduction:**
  - **Purpose:** Reduce the number of features while retaining the essential information.
  - **Examples:** Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE).

Numerical Example: K-means Clustering

Example: We have a dataset of customer spending in a store.

Customer ID	Annual Income ($)	Spending Score
1	15,000	39
2	16,000	81
3	17,000	6
4	18,000	77
5	19,000	40

Objective: Cluster customers into two groups based on their annual income and spending score.

Step 1: Initialize Centroids

- Choose k = 2 (number of clusters).
- Randomly initialize two centroids, $\mu 1 = (15000{,}39)$ and $\mu 2 = (19000{,}40)$.

Step 2: Assign Points to Nearest Centroid

- Compute the distance of each point to the centroids and assign each point to the nearest centroid.
- Distances to $\mu 1$ and $\mu 2$:

Customer ID	Distance to μ_1	Distance to μ_2	Cluster Assignment
1	0.0	4000.01	1

2	42.56	3000.01	1
3	33.29	2000.01	1
4	43.97	1000.01	2
5	4000.00	0.0	2

Step 3: Update Centroids

- Compute the new centroids by taking the mean of all points assigned to each cluster.

New centroids:

$$\mu_1 = \left(\frac{15000 + 16000 + 17000}{3}, \frac{39 + 81 + 6}{3}\right) = (16000{,}42)$$

$$\mu_2 = \left(\frac{18000 + 19000}{2}, \frac{77 + 40}{2}\right) = (18500{,}58.5)$$

Step 4: Repeat Steps 2 and 3

- Repeat until the centroids no longer change significantly.

Final clusters:

Customer ID	Annual Income ($)	Spending Score	Cluster No.
1	15,000	39	1
2	16,000	81	1
3	17,000	6	1
4	18,000	77	2
5	19,000	40	2

Thus, after several iterations, we have two clusters:

- Cluster 1: Customers 1, 2, 3
- Cluster 2: Customers 4, 5

7. Linearity vs non-linearity

Understanding the distinction between linear and non-linear models is crucial for selecting the appropriate machine learning approach. Linear models are simpler, easier to interpret, and less prone to overfitting but may not capture complex relationships in the data. Non-linear models, while more flexible and capable of modeling complex patterns, can be computationally intensive and prone to overfitting. Choosing the right model depends on the nature of the data and the specific problem at hand.

7.1. Linearity:

Definition: A linear relationship implies that there is a straight-line connection between the input variables (features) and the output variable (target). The relationship can be expressed as a linear equation, where each term is either a constant or the product of a constant and a single variable.

Mathematical Form:$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Example: Linear Regression (already explained)

Non-linearity:

Definition:A non-linear relationship implies that the relationship between the input variables and the output variable cannot be represented as a straight line. Instead, it may involve exponents, logarithms, trigonometric functions, or other non-linear transformations.

Mathematical Form:$y = f(x_1, x_2, \dots, x_n)$ where f is a non-linear function of the input variables.

Example: Polynomial Regression

Consider extending the previous example to a polynomial regression model where the relationship between house size and price is quadratic:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

We want to fit a polynomial regression model for given example of the form:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

Size (sq ft)	Price ($)
1500	300,000
1700	340,000
2000	400,000
2100	420,000
2500	500,000

Step 1: Create the Design Matrix X

The design matrix for polynomial regression includes terms for x and x^2. Therefore:

$$X = \begin{pmatrix} 1 & 1500 & 2250000 \\ 1 & 1700 & 2890000 \\ 1 & 2000 & 4000000 \\ 1 & 2100 & 4410000 \\ 1 & 2500 & 6250000 \end{pmatrix}$$

Step 2: Create the Output Vector y

The output vector y is:

$$y = \begin{pmatrix} 300000 \\ 340000 \\ 400000 \\ 420000 \\ 500000 \end{pmatrix}$$

Step 3: Compute X^T

First, compute the transpose of X, denoted as X^T:

$$X^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1500 & 1700 & 2000 & 2100 & 2500 \\ 2250000 & 2890000 & 4000000 & 4410000 & 6250000 \end{pmatrix}$$

Next, multiply X^T by X:

$$X^TX = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1500 & 1700 & 2000 & 2100 & 2500 \\ 2250000 & 2890000 & 4000000 & 4410000 & 6250000 \end{pmatrix} \times \begin{pmatrix} 1 & 1500 & 2250000 \\ 1 & 1700 & 2890000 \\ 1 & 2000 & 4000000 \\ 1 & 2100 & 4410000 \\ 1 & 2500 & 6250000 \end{pmatrix}$$

The result is:

$$X^TX = \begin{pmatrix} 5 & 9800 & 19860000 \\ 9800 & 19860000 & 42420000000 \\ 19860000 & 42420000000 & 91476000000000 \end{pmatrix}$$

Step 4: Compute X^Ty

Next, compute the multiplication ofX^T and y:

$$X^Ty = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1500 & 1700 & 2000 & 2100 & 2500 \\ 2250000 & 2890000 & 4000000 & 4410000 & 6250000 \end{pmatrix} \times \begin{pmatrix} 300000 \\ 340000 \\ 400000 \\ 420000 \\ 500000 \end{pmatrix}$$

The result is:

$$X^T y = \begin{pmatrix} 1960000 \\ 4142000000 \\ 9220400000000 \end{pmatrix}$$

Step 5: Compute the Inverse of $X^T X$

The inverse of the matrix $X^T X$, we can use a computational tool or perform manual calculations.

$$(X^T X)^{-1} = \frac{1}{\det(X^T X)} \text{Adj}(X^T X)$$

Finally, multiply $(X^T X)^{-1}$ by $X^T y$to get θ:

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{pmatrix} = (X^T X)^{-1}$$

Resulting Coefficients:

After performing the calculations, we find:

$$\theta_0 \approx 9.24 \times 10^{-8} \quad \text{(Intercept term)}$$

$$\theta_1 = 200 \quad \text{(Coefficient for } x\text{)}$$

$$\theta_2 \approx 2.19 \times 10^{-14} \quad \text{(Coefficient for } x^2\text{)}$$

Final Model

The final polynomial regression model is:

$$y = 9.24 \times 10^{-8} + 200x + 2.19 \times 10^{-14} x^2$$

7.2. Activation Functions

Activation functions are crucial in neural networks, with linear functions being simple and used mainly for regression outputs, while non-linear functions introduce complexity, allowing the network to learn intricate patterns. Choosing the appropriate activation function is key to effectively training a neural network and achieving good performance on various tasks.

7.2.1.Linear Activation Functions

Definition: A linear activation function returns the weighted sum of the inputs. The output of the function is proportional to the input.

Formula:$f(x) = x$

Characteristics:

- No non-linearity: The output is a direct, unmodified value of the input.
- Simple: Easy to compute.
- Limited: Cannot model complex relationships because it doesn't introduce non-linearity.

Use Case:

- Primarily used in the output layer of regression models where the output is a continuous value.

7.2.2. Non-Linear Activation Functions

Definition: Non-linear activation functions apply a non-linear transformation to the input, allowing the model to learn complex patterns and relationships in the data. These functions can model more complex relationships by enabling the network to approximate non-linear mappings.

Examples of Non-Linear Activation Functions:

- **Sigmoid (Logistic) Function:**
  - **Formula:** $f(x) = \frac{1}{1+e^{-x}}$
  - **Output Range:** (0, 1)
  - **Characteristics:** Smooth gradient, useful for binary classification problems, but suffers from vanishing gradient problems for extreme values of input.
- **Hyperbolic Tangent (Tanh) Function:**
  - **Formula:** $f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
  - **Characteristics:** Similar to sigmoid but outputs values in a wider range, helping to mitigate the vanishing gradient problem.
- **Rectified Linear Unit (ReLU):**
  - **Formula:** $f(x) = \max(0, x)$
  - **Output Range:** [0, ∞)
  - **Characteristics:** Computationally efficient, introduces sparsity in the network, but can suffer from the "dying ReLU" problem where neurons output zero and do not learn.
- **Leaky ReLU:**
  - **Formula:** $f(x) = \max(\alpha x, x)\ (where\ \alpha\ is\ a\ small\ constant)$
  - **Output Range:** (-∞, ∞)
  - **Characteristics:** Similar to ReLU but allows a small gradient when the unit is inactive, which helps in mitigating the "dying ReLU" problem.
- **Softmax Function:**
  - **Formula:** $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$
  - **Output Range:** (0, 1) for each output, with the sum of all outputs equal to 1.
  - **Characteristics:** Used in the output layer of a classifier to represent a probability distribution over multiple classes.

7.3. Weights and bias

Weights and biases are fundamental components of neural networks and other machine learning models. They play a crucial role in determining the output of the model based on the given inputs. Let's delve into their definitions and roles:

7.3.1.Weights

Weights are the parameters within a neural network that are learned from the training data. They are used to control the strength or influence of the input features on the network's predictions. In the context of a simple linear model or neural network, weights are multiplied by the input values, and the result is passed on to the next layer or to the output.

Role in Neural Networks:

- Weights are associated with each connection between neurons (nodes) in the network. They determine how much influence the input feature has on the final prediction.
- During the training process, the learning algorithm adjusts the weights based on the error of the model's predictions compared to the actual output, with the goal of minimizing this error.

Example: In a simple neural network with one input neuron, one output neuron, and one connection between them, if the input value is xxx and the weight is www, the output before applying the activation function would be $w \times x$.

7.3.2.Bias

Bias is an additional parameter in a neural network that is added to the weighted sum of inputs before passing the result through an activation function. It helps the model fit the data more accurately by allowing it to shift the activation function to better match the training data.

Role in Neural Networks:

- Bias allows the activation function to be shifted left or right, which is essential for fitting the model to the training data, especially when the data does not pass through the origin (0,0).
- Without bias, the model's flexibility would be limited, as it could only model relationships that pass through the origin.

Example: Continuing from the previous example, if we include a bias term bbb, the output before applying the activation function would be $w \times x + b$. The bias allows the model to output a non-zero value even when the input is zero, providing additional flexibility in the model.

Mathematical Representation in a Single Layer

For a single-layer perceptron (a simple neural network), the output yyy can be represented as:

$$y = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b)$$

Where:

- $x_1, x_2, \dots, x_n$ are the input features.
- $w_1, w_2, \dots, w_n$are the corresponding weights.
- b is the bias term.
- f is the activation function.

7.4. Loss Function

The loss function, also known as the cost function or objective function, is a critical component of a neural network. It quantifies how well or poorly a model's predictions align with the actual target values in the training data. The primary goal of training a neural network is to minimize the loss function, thereby improving the model's accuracy.A loss function measures the difference between the predicted outputs of the neural network and the actual target values. The choice of loss function depends on the type of problem (regression, classification, etc.) and the specific characteristics of the data.

Types of Loss Functions

- **Mean Squared Error (MSE) for Regression:**
  - **Formula:**$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2$
  - **Description:** The MSE loss function measures the average of the squares of the differences between the actual target values (y_i) and the predicted values $(\hat{y_i})$. It is commonly used in regression tasks, where the output is a continuous value.
- **Binary Cross-Entropy Loss for Binary Classification:**
  - **Formula:**$\text{Loss} = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})]$
  - **Description:** This loss function measures the difference between the actual labels and the predicted probabilities in binary classification problems. It is used when the output is a probability value between 0 and 1.
- **Categorical Cross-Entropy Loss for Multi-class Classification:**
  - **Formula:**$\text{Loss} = -\sum_{i=1}^{m}\sum_{c=1}^{C} y_{i,c}\log\left(\widehat{y_{i,c}}\right)$
  - **Description:** Used in multi-class classification tasks, this loss function compares the one-hot encoded true labels with the predicted probabilities for each class.
- **Hinge Loss for Support Vector Machines:**
  - **Formula:**$\text{Loss} = \sum_{i=1}^{m} \max\left(\text{M}(0,1 - y_i\hat{y_i})\right)$
  - **Description:** Hinge loss is used in support vector machines for binary classification. It penalizes incorrect or insufficiently confident predictions.

*7.4.1.*Role of the Loss Function in Training

- **Guiding the Learning Process:** The loss function provides a measure of how well the model is performing. During training, the model's weights and biases are adjusted to minimize the loss function, thus improving the model's predictions.
- **Optimization:** The optimization algorithm, such as gradient descent, uses the gradient of the loss function with respect to the model's parameters (weights and biases) to determine the direction and magnitude of updates needed to reduce the loss.
- **Evaluating Performance:** The value of the loss function gives an indication of the model's performance on the training and validation datasets. A lower loss value generally indicates better model performance.

Consider a simple linear regression model with one input feature, x, and one output, y . The model predicts $\hat{y} = w \times x + b$.

Given Data:

Feature (x)	Actual Output (y)
1	2
2	2.5
3	3.5

Model Parameters (initial guess):

- Weight (w) = 1
- Bias (b) = 0.5

Predicted Outputs:

- $For\ x = 1: \hat{y} = 1 \times 1 + 0.5 = 1.5$
- $For\ x = 2: \hat{y} = 1 \times 2 + 0.5 = 2.5$
- $For\ x = 3: \hat{y} = 1 \times 3 + 0.5 = 3.5$

Calculate MSE:

- For the first data point: $(2 - 1.5)^2 = 0.25$
- For the second data point: $(2.5 - 2.5)^2 = 0$
- For the third data point: $(3.5 - 3.5)^2 = 0$

The MSE is:

$$\text{MSE} = \frac{1}{3}(0.25 + 0 + 0) = 0.0833$$

The model's parameters will be updated during training to minimize this MSE, improving the model's accuracy.

7.5. Gradient Descent

Gradient Descent is an optimization algorithm used to minimize the loss function in machine learning models, particularly neural networks. It iteratively adjusts the model's parameters (weights and biases) to find the values that result in the lowest possible loss. This process improves the model's accuracy by aligning its predictions more closely with the actual target values.

7.5.1. Types of Gradient Descent

- **Batch Gradient Descent:**
  - Uses the entire training dataset to compute the gradient of the loss function.
  - Can be computationally expensive and slow for large datasets.
- **Stochastic Gradient Descent (SGD):**
  - Uses only one training example at a time to compute the gradient and update the parameters.
  - Faster and more memory-efficient but has higher variance in the updates, which can lead to a noisier convergence path.
- **Mini-batch Gradient Descent:**
  - Uses a small, randomly selected subset of the training data (mini-batch) to compute the gradient.
  - Offers a balance between the computational efficiency of SGD and the smooth convergence of batch gradient descent.

Example: Simple Linear Regression

Consider a simple linear regression model:

$$y = wx + b$$

Where w is the weight and b is the bias. Given a training dataset, the goal is to find the optimal w and b that minimize the Mean Squared Error (MSE) loss function:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Gradient Descent Steps:

1. **Initialize**w and b with random values.
2. **Compute the loss** using the current values of w and b.
3. **Compute the gradients**:
   - Gradient with respect to w:$\frac{\partial \text{MSE}}{\partial w} = \frac{2}{m}\sum_{i=1}^{m} x_i\big(y_i - (wx_i + b)\big)$
   - Gradient with respect to b: $\frac{\partial \text{MSE}}{\partial b} = \frac{2}{m}\sum_{i=1}^{m}\big(y_i - (wx_i + b)\big)$
4. **Update the parameters** using the learning rate α:
   - $w := w - \alpha\frac{\partial \text{MSE}}{\partial w}$
   - $b := b - \alpha\frac{\partial \text{MSE}}{\partial b}$
5. **Repeat** the process for a set number of iterations or until the loss converges.

7.6. Unstable gradient problem

The unstable gradient problem refers to issues that can arise during the training of deep neural networks due to the exponential growth or decay of gradients. It manifests in two primary forms: the **exploding gradient problem** and the **vanishing gradient problem**. Both of these issues can significantly affect the learning process, making it difficult for the network to converge to an optimal solution.

- *Exploding Gradient Problem*

The exploding gradient problem occurs when the gradients of the loss function with respect to the weights grow exponentially as they are propagated backward through the network during training. This often happens in deep networks where many layers exist between the input and output.

Reasons:

- Deep architectures with many layers.
- Large initial weights.
- Certain activation functions that can amplify the gradients.

Consequences:

- The model parameters can become extremely large.
- The learning process becomes unstable, with the loss function diverging rather than converging.

- The optimization algorithm can fail to converge, leading to poor performance.

Mitigation Techniques:

- **Gradient Clipping:** Limit the gradient values during backpropagation to a specified maximum threshold, preventing them from growing too large.
- **Weight Regularization:** Techniques like L2 regularization can help prevent weights from becoming excessively large.
- **Normalized Initialization:** Using initialization schemes such as Xavier or He initialization, which are designed to keep the scale of the gradients roughly the same in all layers.

- *Vanishing Gradient Problem*

The vanishing gradient problem occurs when the gradients of the loss function become extremely small, approaching zero, as they are propagated backward through the network. This problem is particularly prevalent in deep networks, especially those with sigmoidal or tanh activation functions.

Reasons:

- Deep architectures with many layers.
- Certain activation functions that squash the input into a small range, causing gradients to diminish as they are propagated backward.
- Poor initialization of weights.

Consequences:

- Slow or stalled learning, as the weights update very slowly or not at all.
- Difficulty in training the lower layers of the network, which may effectively stop learning.

Mitigation Techniques:

- **ReLU Activation Function:** The ReLU (Rectified Linear Unit) activation function does not suffer from the vanishing gradient problem as much because it does not saturate for positive values.
- **Batch Normalization:** This technique normalizes the inputs to each layer, reducing internal covariate shift and helping to maintain the gradient flow.
- **Residual Connections:** In architectures like ResNets, residual connections allow gradients to flow more easily through the network, bypassing some layers and mitigating the vanishing gradient problem.

Example

Consider a simple neural network with three layers where each neuron uses the sigmoid activation function. The sigmoid function squashes the input to a range between 0 and 1, with derivatives also constrained between 0 and 0.25. If a gradient update in the later layers is small, it becomes even smaller as it backpropagates through each successive layer, eventually becoming negligible. This is the vanishing gradient problem.

On the other hand, if the gradients are large, they can become exponentially larger as they are multiplied by the weights in each layer during backpropagation, leading to the exploding gradient problem.

7.7. Multilayer network

A multilayer network, often referred to as a Multilayer Perceptron (MLP), is a type of feedforward artificial neural network. It consists of multiple layers of neurons: an input layer, one or more hidden layers, and an output layer. MLPs are capable of learning and representing complex patterns in data by using non-linear activation functions.

7.7.1. Components of a Multilayer Network

- **Input Layer:**
  - The first layer in the network that receives the input data. Each neuron in this layer represents a feature from the input dataset.
- **Hidden Layers:**
  - Layers of neurons between the input and output layers. These layers are called "hidden" because their values are not observed in the training dataset.
  - The purpose of the hidden layers is to capture complex patterns and relationships in the data. Each neuron in a hidden layer receives input from the previous layer and passes its output to the next layer.
- **Output Layer:**
  - The final layer in the network, producing the output of the model. The number of neurons in the output layer depends on the task:
    - **Regression:** Typically, there is a single neuron outputting a continuous value.
    - **Binary Classification:** Typically, there is one neuron outputting a probability (using a sigmoid activation function) or two neurons for one-hot encoding (using softmax).
    - **Multiclass Classification:** Multiple neurons representing each class, with a softmax activation function to output class probabilities.

7.7.2. How a Multilayer Network Works

- **Forward Propagation:**
  - Input data is fed into the network, and each layer performs a series of operations, applying weights, biases, and activation functions. The output from one layer becomes the input to the next.
  - The process continues through all the layers until the final output is generated in the output layer.
- **Activation Functions:**
  - Activation functions introduce non-linearity into the network, enabling it to learn and represent more complex patterns. Common activation functions include ReLU, sigmoid, tanh, and softmax.
- **Backpropagation:**
  - The backpropagation algorithm is used to train the network by minimizing the loss function. It involves computing the gradient of the loss function with respect to each weight and bias using the chain rule of calculus.
  - The gradients are then used to update the weights and biases through an optimization algorithm like gradient descent.

7.7.3. Training a Multilayer Network

The training process for an MLP involves the following steps:

1. **Initialization:**
    - Initialize the weights and biases, typically with small random values.
2. **Forward Propagation:**
    - Pass the input data through the network to obtain predictions.
3. **Compute Loss:**
    - Calculate the loss (error) between the network's predictions and the actual target values using a loss function appropriate for the task.
4. **Backward Propagation:**
    - Calculate the gradients of the loss with respect to the weights and biases by propagating the error backward through the network.
5. **Update Parameters:**
    - Update the weights and biases using the gradients and an optimization algorithm.
6. **Repeat:**
    - Repeat the forward and backward propagation steps for many iterations (epochs) until the loss converges to a minimum value or until a satisfactory level of performance is achieved.

Example:

A numerical example of training a simple Multilayer Perceptron (MLP) for a binary classification task. We use a small dataset with two features and a target variable.

7.7.4.Dataset

Feature 1	Feature 2	Target
0.1	0.2	0
0.2	0.3	0
0.3	0.4	1
0.4	0.5	1

7.7.5.MLP Structure

- **Input Layer:** 2 neurons (for the two features)
- **Hidden Layer 1:** 2 neurons with ReLU activation
- **Output Layer:** 1 neuron with sigmoid activation (for binary classification)

7.7.6.Initial Weights and Biases

Assume the following initial weights and biases:

- **Weights between Input Layer and Hidden Layer 1:**
    - $W_1 = \begin{pmatrix} 0.1 & -0.2 \\ 0.4 & 0.3 \end{pmatrix}$
- **Biases for Hidden Layer 1:**$b_1 = (0.1 - 0.1)$
- **Weights between Hidden Layer 1 and Output Layer:**
    - $W_2 = \begin{pmatrix} 0.3 \\ -0.4 \end{pmatrix}$
- **Bias for Output Layer:**$b_2 = 0.2$

7.7.7.Activation Functions

- **ReLU for Hidden Layer 1:**$\text{ReLU}(x) = \max(0, x)$
- **Sigmoid for Output Layer:**$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$

7.7.8.Forward Propagation

For the first training example (0.1, 0.2):

1. **Input Layer to Hidden Layer 1:**

   Compute the weighted sum z_1 and apply the ReLU activation function:

$$z_1 = W_1 \cdot \begin{pmatrix} 0.1 \\ 0.2 \end{pmatrix} + b_1$$

$$z_1 = \begin{pmatrix} 0.1 \cdot 0.1 + (-0.2) \cdot 0.2 \\ 0.4 \cdot 0.1 + 0.3 \cdot 0.2 \end{pmatrix} + \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix}$$

$$z_1 = \begin{pmatrix} 0.01 - 0.04 + 0.1 \\ 0.04 + 0.06 - 0.1 \end{pmatrix}$$

$$z_1 = \begin{pmatrix} 0.07 \\ 0.0 \end{pmatrix}$$

   Apply the ReLU activation:

$$a_1 = \begin{pmatrix} \text{ReLU}(0.07) \\ \text{ReLU}(0.0) \end{pmatrix} = \begin{pmatrix} 0.07 \\ 0.0 \end{pmatrix}$$

2. **Hidden Layer 1 to Output Layer:**

   Compute the weighted sum z_2 and apply the Sigmoid activation function:

$$z_2 = W_2 \cdot a_1 + b_2$$

$$z_2 = (0.3 \cdot 0.07 + (-0.4) \cdot 0.0) + 0.2$$

$$z_2 = 0.021 + 0.2$$

$$z_2 = 0.221$$

Apply the Sigmoid activation:

$$\hat{y} = \frac{1}{1 + e^{-0.221}} \approx 0.555$$

The predicted output for the first training example is approximately 0.555.

Loss Function: For simplicity, let's use the binary cross-entropy loss function. The loss for the first example is:

$$\text{Loss} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

$$Where\ y = 0\ (actual\ \ target)\ and\ \hat{y} = 0.555(predicted\ output).$$

$$\text{Loss} = -(0\log(0.555) + (1-0)\log(1-0.555))$$

$$\text{Loss} = -(0 + \log(0.445))$$

$$\text{Loss} \approx -(-0.809) = 0.809$$

7.8. Back Propagation

Backpropagation is a fundamental algorithm for training neural networks. It involves calculating the gradient of the loss function with respect to each weight and bias by the chain rule, propagating the error backward through the network. This gradient is then used to update the weights and biases, thereby minimizing the loss function.

Key Concepts

1. **Forward Pass:**
    - Compute the output of the network for a given input by propagating the input through the network.
2. **Loss Calculation:**
    - Calculate the difference between the predicted output and the actual target output using a loss function.
3. **Backward Pass (Backpropagation):**
    - Compute the gradient of the loss function with respect to each parameter (weight and bias) by propagating the error backward through the network.
4. **Parameter Update:**
    - Update the weights and biases using the computed gradients to reduce the loss. This update is typically done using gradient descent.

Example:

Forward pass as forward propagation (Do it as earlier)

Backward Pass (Backpropagation)

Output Layer:

- **Calculate the gradient of the loss with respect to z_2:**

$$\frac{\partial \text{Loss}}{\partial z_2} = \hat{y} - y = 0.555 - 0 = 0.555$$

- **Gradient with respect to W_2 and b_2:**

- **For W_2:**

$$\frac{\partial \text{Loss}}{\partial W_2} = a_1 \cdot \frac{\partial \text{Loss}}{\partial z_2}$$

$$\frac{\partial \text{Loss}}{\partial W_2} = \begin{pmatrix} 0.07 \\ 0.0 \end{pmatrix} \cdot 0.555 = \begin{pmatrix} 0.03885 \\ 0.0 \end{pmatrix}$$

- **For b_2:**

$$\frac{\partial \text{Loss}}{\partial b_2} = \frac{\partial \text{Loss}}{\partial z_2} = 0.555$$

Hidden Layer:

- **Calculate the gradient with respect to a_1:**

$$\frac{\partial \text{Loss}}{\partial a_1} = W_2 \cdot \frac{\partial \text{Loss}}{\partial z_2}$$

$$\frac{\partial \text{Loss}}{\partial a_1} = \begin{pmatrix} 0.3 \\ -0.4 \end{pmatrix} \cdot 0.555 = \begin{pmatrix} 0.1665 \\ -0.222 \end{pmatrix}$$

- **Calculate the gradient with respect toz_1:**

$$\frac{\partial \text{Loss}}{\partial z_1} = \frac{\partial \text{Loss}}{\partial a_1} \cdot \text{ReLU}'(z_1)$$

Since ReLU's derivative is 1 for positive inputs and 0 for zero or negative inputs:

$$\frac{\partial \text{Loss}}{\partial z_1} = \begin{pmatrix} 0.1665 \cdot 1 \\ -0.222 \cdot 0 \end{pmatrix} = \begin{pmatrix} 0.1665 \\ 0 \end{pmatrix}$$

- **Gradient with respect toW_1 andb_1:**
- **For W_1:**

$$\frac{\partial \text{Loss}}{\partial W_1} = x \cdot \frac{\partial \text{Loss}}{\partial z_1}$$

$$\frac{\partial \text{Loss}}{\partial W_1} = \begin{pmatrix} 0.1 & 0.2 \end{pmatrix} \cdot \begin{pmatrix} 0.1665 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.01665 & 0.0333 \end{pmatrix}$$

- **For b_1:**

$$\frac{\partial \text{Loss}}{\partial b_1} = \frac{\partial \text{Loss}}{\partial z_1} = \begin{pmatrix} 0.1665 \\ 0 \end{pmatrix}$$

- **Parameter Update**

Using a learning rate $\alpha = 0.01$ we update the weights and biases:

Update W_2:

$$W_2 := W_2 - \alpha \frac{\partial \text{Loss}}{\partial W_2}$$

$$W_2 := \begin{pmatrix} 0.3 \\ -0.4 \end{pmatrix} - 0.01 \cdot \begin{pmatrix} 0.03885 \\ 0.0 \end{pmatrix} = \begin{pmatrix} 0.2996115 \\ -0.4 \end{pmatrix}$$

Update b_2:

$$b_2 := b_2 - \alpha \frac{\partial \text{Loss}}{\partial b_2}$$

$$b_2 := 0.2 - 0.01 \cdot 0.555 = 0.19445$$

Update W_1:

$$W_1 := W_1 - \alpha \frac{\partial \text{Loss}}{\partial W_1}$$

$$W_1 := \begin{pmatrix} 0.1 & -0.2 \\ 0.4 & 0.3 \end{pmatrix} - 0.01 \cdot (0.01665 \quad 0.0333)$$

$$W_1 := \begin{pmatrix} 0.0998335 & -0.200333 \\ 0.4 & 0.3 \end{pmatrix}$$

Update b_1:

$$b_1 := b_1 - \alpha \frac{\partial \text{Loss}}{\partial b_1}$$

$$b_1 := \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix} - 0.01 \cdot \begin{pmatrix} 0.1665 \\ 0 \end{pmatrix}$$

$$\mathrm{b}_1 := \begin{pmatrix} 0.098335 \\ -0.1 \end{pmatrix}$$

7.9. Training andTesting

In machine learning, the concepts of training and testing are fundamental to building and evaluating models. Understanding these concepts is crucial for developing models that generalize well to unseen data.

Training Phase

The training phase involves teaching a machine learning model to recognize patterns in a dataset. This is done by feeding the model a large set of labeled data, known as the training set, and adjusting the model's parameters (weights and biases) to minimize a loss function.The primary goal of training is to minimize the loss function, thereby improving the model's ability to make accurate predictions on the training data. However, it is crucial to ensure that the model does not overfit the training data, meaning it learns the training data too well, including noise, and performs poorly on new, unseen data.

Testing Phase

Definition: The testing phase involves evaluating the trained model's performance on a separate set of data known as the testing set. This data is not used during the training process and serves as a proxy for how the model will perform on unseen, real-world data. The primary goal of testing is to assess the model's generalization ability. A good model should perform well not only on the training data but also on the testing data. This indicates that the model has learned the underlying patterns in the data rather than memorizing the training examples.

Example Scenario

Consider a model being developed to classify emails as spam or not spam:

1. **Training Phase:**
   - The model is trained on a dataset of emails labeled as spam or not spam. It learns to identify patterns and features that differentiate spam emails from legitimate ones.
2. **Testing Phase:**
   - After training, the model is tested on a new set of emails it hasn't seen before. The model's predictions are compared to the actual labels, and metrics like accuracy and precision are calculated to evaluate its performance.

7.10. Autoencoders in Machine Learning

Autoencoders are a type of artificial neural network used for unsupervised learning tasks, particularly for dimensionality reduction, feature learning, and data compression. They work by attempting to copy their input to their output, thereby learning a compressed, lower-dimensional representation of the data. The primary objective of an autoencoder is to minimize the difference between the input and the output, which is often done using a reconstruction loss function like Mean Squared Error (MSE). The network is trained to produce outputs that are as close as possible to the inputs, effectively learning the most salient features of the data.

Structure of Autoencoders

An autoencoder consists of two main components:

1. **Encoder:**
   - The encoder part of the network compresses the input data into a lower-dimensional representation called the **latent space** or **bottleneck**. This is achieved by applying a series of linear and non-linear transformations through multiple layers, gradually reducing the dimensionality of the data.
2. **Decoder:**
   - The decoder takes the encoded representation from the latent space and attempts to reconstruct the original input. It essentially reverses the transformations applied by the encoder, expanding the compressed data back to the original dimension.

The structure of an autoencoder can be summarized as:

- **Input Layer:** The original data.
- **Hidden Layers (Encoder):** Layers that compress the data.
- **Latent Space:** The compressed, lower-dimensional representation.
- **Hidden Layers (Decoder):** Layers that reconstruct the data from the latent space.
- **Output Layer:** The reconstructed data.

7.10.1. Types of Autoencoders

1. **Vanilla Autoencoder:**
   - The simplest form of autoencoder, which consists of one or more fully connected layers in both the encoder and decoder.
2. **Convolutional Autoencoder:**
   - Uses convolutional layers in the encoder and decoder, making it particularly useful for image data. Convolutional autoencoders can learn spatial hierarchies of features, which are beneficial for tasks like image denoising and super-resolution.
3. **Variational Autoencoder (VAE):**
   - A probabilistic model that imposes a probability distribution on the latent space. VAEs can generate new data samples by sampling from the learned distribution, making them useful for generative tasks.
4. **Denoising Autoencoder:**
   - Aims to reconstruct the original input from a corrupted version of it. It learns to denoise the data, making it useful for applications like noise reduction in images.
5. **Sparse Autoencoder:**
   - Introduces a sparsity constraint on the hidden units, encouraging the network to learn only a few active features at a time. This is achieved using a sparsity penalty in the loss function.

7.10.2. Applications of Autoencoders

1. **Dimensionality Reduction:**
   - Autoencoders can be used to reduce the dimensionality of data while preserving important features, similar to Principal Component Analysis (PCA).
2. **Data Denoising:**
   - By training on noisy data, autoencoders can learn to reconstruct the original, noise-free data, making them useful for tasks like image and signal denoising.
3. **Anomaly Detection:**
   - Autoencoders can identify anomalies by reconstructing normal data patterns well but failing to reconstruct anomalous data, as they haven't seen such patterns during training.
4. **Image Compression:**
   - Autoencoders can learn efficient representations of images, enabling compression and decompression of image data.
5. **Generative Models:**
   - Variational Autoencoders (VAEs) can generate new data samples by sampling from the latent space distribution, useful in applications like image generation and data augmentation.

7.11. Batch Normalization

Batch normalization is a technique used to improve the training of deep neural networks by normalizing the inputs of each layer. This process helps stabilize and accelerate the training process by addressing issues like internal covariate shift and gradient vanishing or exploding.

•**Internal Covariate Shift:**

- The term refers to the change in the distribution of network activations due to the changing parameters during training. This shift can slow down training and make it more challenging for the network to converge.

•**Normalization:**

- Batch normalization normalizes the activations of the previous layer for each mini-batch, bringing them to a standard normal distribution. This normalization involves shifting and scaling the values to have a mean of zero and a variance of one.

7.11.1. How Batch Normalization Works

1. **Mini-Batch Calculation:**
   - For each mini-batch during training, calculate the mean (μ_B) and variance (σ_B^2) of the activations.
2. **Normalization:**
   - Normalize each activation x_i in the mini-batch using the computed mean and variance:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Where:

   - $\hat{x_i}$ is the normalized activation.
   - ϵ is a small constant added for numerical stability to prevent division by zero.
3. **Scaling and Shifting:**
   - After normalization, scale and shift the normalized activations using learnable parameters $\gamma\ and\ \beta$:

$$y_i = \gamma\hat{x_i} + \beta$$

Where:

   - $y_i$is the output of the batch normalization layer.
   - $\gamma\ and\ \beta$ are learnable parameters that allow the network to restore the original activation distribution if that is optimal.
4. **Learnable Parameters:**
   - $\gamma\ and\ \beta$ are updated during the training process like other model parameters, enabling the network to find the best scaling and shifting for the normalized activations.

7.11.2. Benefits of Batch Normalization

1. **Accelerated Training:**
   - Batch normalization allows the use of higher learning rates because it mitigates issues like exploding and vanishing gradients. This can lead to faster convergence.
2. **Regularization Effect:**

    - By adding noise through normalization based on mini-batch statistics, batch normalization has a regularization effect, which can reduce the need for other regularization techniques like dropout.
3. **Reduced Sensitivity to Initialization:**
    - Batch normalization reduces the sensitivity of the network to the initial choice of parameters, making the network more robust and easier to train.
4. **Improved Gradient Flow:**
    - By maintaining stable activations, batch normalization improves the gradient flow through the network, making it easier to train deep networks.

7.12. Dropout

Dropout is a regularization technique used to prevent overfitting in neural networks. It works by randomly "dropping out" a subset of neurons during the training phase, meaning they are temporarily removed from the network. This forces the network to learn more robust features that are not reliant on any single neuron, thus improving the model's generalization ability.

7.12.1. How Dropout Works

1. **Random Neuron Deactivation:**
    - During each training iteration, dropout randomly selects a subset of neurons to be set to zero (i.e., deactivated) in the forward pass. This is done independently for each mini-batch and for each neuron.
    - The probability of a neuron being dropped, called the **dropout rate** or **dropout probability**, is a hyperparameter denoted as p. Common choices for p are 0.5 for hidden layers and a smaller value like 0.2 or no dropout for input and output layers.
2. **Scaling During Training:**
    - To maintain the expected output value, the activations of the remaining neurons are scaled by $\frac{1}{1-p}$ during training. This scaling ensures that the sum of the activations remains approximately the same, even though some neurons are dropped.
3. **Full Network During Testing:**
    - During testing or inference, dropout is not applied. Instead, the full network is used, and the weights are scaled down by the dropout rate p. Alternatively, this can be seen as multiplying the weights by $1 - p$ during inference.

7.12.2. Benefits of Dropout:

1. **Prevents Overfitting:**
    - By randomly dropping neurons, dropout forces the network to learn redundant representations of the data, which helps prevent overfitting to the training data.
2. **Improves Generalization:**
    - Dropout encourages the network to learn more general features that are useful for all input data, not just the training examples.
3. **Efficient Training:**
    - Despite the additional computational cost during training, dropout can lead to more efficient training as it often converges faster and leads to better-performing models.

Implementation:

During training for each layer in the network:

1. **Create a binary mask** with the same shape as the layer's activations. The mask contains 0s and 1s, where a 1 indicates the neuron is kept and a 0 indicates it is dropped.

2. **Multiply** the mask with the layer's activations.
3. **Scale** the activations by $\frac{1}{1-p}$ to maintain the expected sum of the outputs.

7.13. L1 and L2 regularization

L1 and L2 regularization are techniques used to prevent overfitting in machine learning models by adding a penalty term to the loss function. This penalty discourages the model from fitting too closely to the training data, thus helping to generalize better to unseen data. Both regularization methods achieve this by imposing different constraints on the model parameters.

7.13.1. L1 Regularization (Lasso Regularization)

1. **Definition:**
   - L1 regularization adds a penalty equal to the absolute value of the magnitude of coefficients to the loss function.
2. **Mathematical Formulation:**
   - For a model with weights θ, the L1 regularization term added to the loss function $J(\theta)$ is:

$$J(\theta) + \lambda \sum_{i=1}^{n} |\theta_i|$$

   - Here, λ is the regularization parameter that controls the strength of the penalty.
3. **Effect on Weights:**
   - L1 regularization tends to produce sparse weight vectors, meaning it can drive some of the weights to exactly zero. This property makes L1 regularization useful for feature selection, as it effectively removes less important features from the model.
4. **Applications:**
   - L1 regularization is often used when the model needs to be interpretable, and there is a desire to have some features with exactly zero coefficients.

7.13.2. L2 Regularization (Ridge Regularization)

1. **Definition:**
   - L2 regularization adds a penalty equal to the square of the magnitude of coefficients to the loss function.
2. **Mathematical Formulation:**
   - For a model with weights θ, the L2 regularization term added to the loss function J(θ) is:

$$J(\theta) + \lambda \sum_{i=1}^{n} \theta_i^2$$

   - Here, λ is the regularization parameter that controls the strength of the penalty.
3. **Effect on Weights:**
   - L2 regularization tends to distribute the weights more evenly, reducing the overall size of the weights but rarely driving them to exactly zero. It encourages the model to keep all weights small, which can help in reducing model complexity and avoiding overfitting.
4. **Applications:**

- L2 regularization is commonly used when it is important to retain all features but to limit their influence, ensuring that the model does not become overly sensitive to any particular feature.

7.13.3. Choosing Regularization:

The choice between L1, L2, depends on the specific problem:

- **L1 Regularization:** Use when feature selection is important.
- **L2 Regularization:** Use when retaining all features with small weights is preferable.

L1 and L2 regularization are crucial techniques in machine learning for preventing overfitting. They add penalties to the loss function based on the magnitude of the model coefficients. L1 regularization encourages sparsity in the model, potentially setting some coefficients to zero, while L2 regularization encourages small, distributed coefficients. Both techniques help improve the generalization of models and are widely used in various machine learning applications.

7.14. Tuning Hyper Parameters

Hyperparameter tuning is the process of selecting the optimal set of hyperparameters for a machinelearning model. Unlike model parameters (which are learned during training), hyperparameters are set before the training process begins and govern the model's learning process. The choice of hyperparameters can significantly impact the model's performance and generalization ability.

Key Hyperparameters:

Hyperparameters vary depending on the type of model. Here are some common hyperparameters for different models:

1. **Linear Models (e.g., Linear Regression, Logistic Regression):**
   - **Regularization parameter (λ)**: Controls the strength of the regularization (L1, L2).
   - **Learning rate (α)**: Determines the step size during optimization.
2. **Decision Trees and Ensembles (e.g., Random Forest, Gradient Boosting):**
   - **Max depth**: The maximum depth of the tree.
   - **Min samples split**: The minimum number of samples required to split a node.
   - **Number of trees**: For ensemble methods, the number of trees in the forest.
3. **Neural Networks:**
   - **Learning rate**: Step size for weight updates.
   - **Number of layers and neurons**: The architecture of the network.
   - **Activation functions**: Functions that determine the output of each neuron.
   - **Batch size**: Number of samples per gradient update.
   - **Dropout rate**: Proportion of neurons to drop during training for regularization.

7.15. Support Vector Machines (SVM):

- **C**: Regularization parameter.
- **Kernel**: The type of kernel function (linear, polynomial, RBF).

Hyperparameter tuning is a critical step in optimizing machine learning models. It involves selecting the best set of hyperparameters that maximize model performance while minimizing overfitting.