# *Wolaita Sodo University,*
# *School of Informatics, Department of Computer Science*

# *Introduction to Artificial Intelligence Module*

*Prepared & Complied by:* ***Melaku Bayih*** *(MSc. in Computer Science)*

*Reviewers:*

1. Arba Asha (MSc.), HoD, Department of Computer Science
2. Dawit Uta (MTech.)
3. Mesay Wana (BSc.)

*6 February, 2023*
*Wolaita Sodo, Ethiopia*

# Contents

# Chapter 1: Introduction to Artificial Intelligence

Artificial Intelligence is the branch of computer science concerned with making computers behave like human.

## 1.1.Objectives /goals of AI

- To Create Expert Systems: The systems which exhibit intelligent behaviour, learn, demonstrate, explain, and advice its users.
- To Implement Human Intelligence in Machines: Creating systems that understand, think, learn, and behave like humans.
- Scientific goal: To determine which ideas about knowledge representation, learning, rule systems, search, and so on, explain various sorts of real intelligence. To understand the mechanism behind human intelligence.
- Engineering goal: design useful, intelligent artifacts. To solves real world problems using AI techniques such as knowledge representation, learning, rule systems, search, and so on. To develop concepts and tools for building intelligent agents capable of solving real world problems. .

**Examples:**

- Natural language understanding systems.
- Intelligent robots.
- Speech and vision recognition systems.
- Common sense reasoning systems

We call ourselves Homo sapiens—man the wise (Human being)—because our intelligence is so important to us. For thousands of years, we have tried to understand how we think; thatis, how a mere handful of matter can perceive, understand, predict, and manipulate a world far larger and more complicated than itself. AI is one of the newest fields in science and engineering. Work started in earnest soon after World War II, and the name itself was coined in 1956. Al currently encompasses a huge variety of subfields, ranging from the general (learning and perception) to the specific, such as playing chess, proving mathematical theorems, writing poetry, driving a car on a crowded street, and diagnosing diseases.

## 1.2.What Is AI

John McCarthy, who coined the term Artificial Intelligence in 1956, defines it as "the science and engineering of making intelligent machines, especially intelligent computer programs." It is the Intelligence of machine and the branch of computer science that aims to create it.
**Intelligence:** Intelligence is a property/ability attributed to people, such as to know, to think, to talk, to learn. Intelligence = Knowledge + ability to perceive, feel, comprehend, process, communicate, judge, learn.

It is the capability of observing, learning remembering & reasoning. AI attempts to develop intelligent agents.
Example: speech recognition, Image pattern understanding etc.

**Characteristics of Intelligent system**

- Use vast amount of knowledge
- Learn from experience and adopt to changing environment
- Interact with human using language and speech
- Respond in real time
- Tolerate error and ambiguity in communication

### 1.3.Types of Artificial Intelligence

Artificial Intelligence can be divided in various types, there are mainly two types of main categorization which are based on capabilities and based on functionally of AI. Following is flow diagram which explain the types of AI.

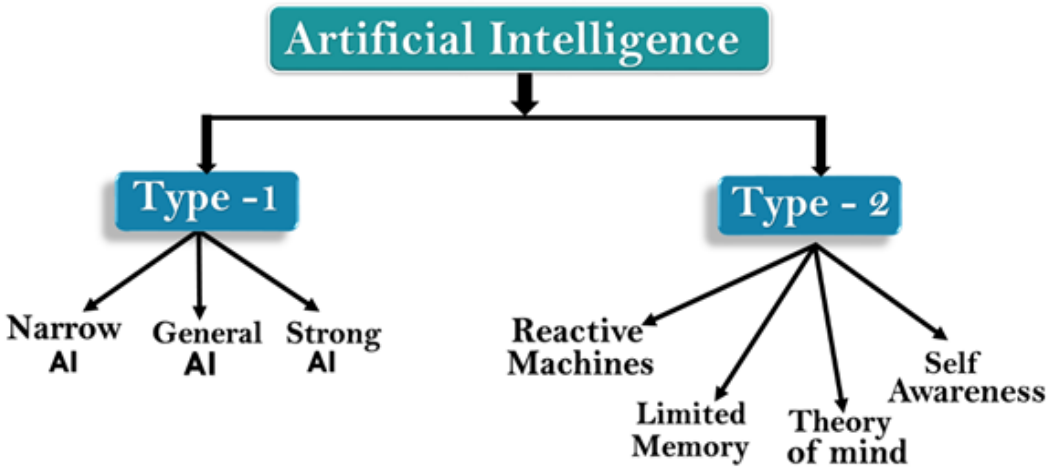



*Figure 1.1: Types of AI*

### 1.3.1. AI type-1: Based on Capabilities

**1.Weak AI or Narrow AI:**

- Narrow AI is a type of AI which is able to perform a dedicated task with intelligence. The most common and currently available AI is Narrow AI in the world of Artificial Intelligence.
- Narrow AI cannot perform beyond its field or limitations, as it is only trained for one specific task. Hence it is also termed as weak AI. Narrow AI can fail in unpredictable ways if it goes beyond its limits.
- Apple Siriis a good example of Narrow AI, but it operates with a limited pre-defined range of functions.
- IBM's Watson supercomputer also comes under Narrow AI, as it uses an Expert system approach combined with Machine learning and natural language processing.

- Some Examples of Narrow AI are playing chess, purchasing suggestions on e-commerce site, self-driving cars, speech recognition, and image recognition.

2.**General AI:**

- General AI is a type of intelligence which could perform any intellectual task with efficiency like a human.
- The idea behind the general AI to make such a system which could be smarter and think like a human by its own.
- Currently, there is no such system exist which could come under general AI and can perform any task as perfect as a human.
- The worldwide researchers are now focused on developing machines with General AI.
- As systems with general AI are still under research, and it will take lots of efforts and time to develop such systems.

**3.Super AI:**

- Super AI is a level of Intelligence of Systems at which machines could surpass human intelligence, and can perform any task better than human with cognitive properties. It is an outcome of general AI.
- Some key characteristics of strong AI include capability include the ability to think, to reason, solve the puzzle, make judgements, plan, learn, and communicate by its own.
- Super AI is still a hypothetical concept of Artificial Intelligence. Development of such systems in real is still world changing task.

### 1.3.2. Artificial Intelligence type-2: Based on functionality

1. **Reactive Machines**

- Purely reactive machines are the most basic types of Artificial Intelligence.
- Such AI systems do not store memories or past experiences for future actions.
- These machines only focus on current scenarios and react on it as per possible best action.
- IBM's Deep Blue system is an example of reactive machines.
- Google's AlphaGo is also an example of reactive machines.

2. **Limited Memory**

- Limited memory machines can store past experiences or some data for a short period of time.
- These machines can use stored data for a limited time period only.
- Self-driving cars are one of the best examples of Limited Memory systems. These cars can store recent speed of nearby cars, the distance of other cars, speed limit, and other information to navigate the road.

3. Theory of Mind

- Theory of Mind AI should understand the human emotions, people, beliefs, and be able to interact socially like humans.

- This type of AI machines are still not developed, but researchers are making lots of efforts and improvement for developing such AI machines.

4. Self-Awareness
   - Self-awareness AI is the future of Artificial Intelligence. These machines will be super intelligent, and will have their own consciousness, sentiments, and self-awareness.
   - These machines will be smarter than human mind.
   - Self-Awareness AI does not exist in reality still and it is a hypothetical concept.

### 1.4.Approaches to AI

AI is the study of – how to make computers do things which at the moment, people do better.
The definitions of AI according to some text books are categorized into four approaches and are summarized in the table below:

| | |
|---|---|
| Systems that think like humans<br>"The exciting new effort to make computers think … machines with minds, in the full and literal sense."(Haugeland,1985) | Systems that think rationally<br>"The study of mental faculties through the use of computer models."<br>(Charniak and McDermont,1985)<br>Use of Computer models |
| Systems that act like humans<br>The art of creating machines that perform functions that require intelligence when performed by people."(Kurzweil,1990)<br>Study of how to make computer to do things. | Systems that act rationally<br>"Computational intelligence is the study of the design of intelligent agents."(Poole et al.,1998)<br>Study of Automation of Intelligent behavior. |

### 1.4.1. Thinking Humanly: The Cognitive Modelling Approach

If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds.
Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program.

a. Through introspection – trying to capture our own thoughts as they go by;
b. Through psychological experiments

Allen Newell and Herbert Simon, who developed GPS, the "General Problem Solver" tried to trace the reasoning steps to traces of human subjects solving the same problems.
Example. A program that plays chess.
Instead of making the best possible chess-playing program, you would make one that play chess like people do.

### 1.4.2. Thinking Rationally: The Laws of Thought

A system is rational if it thinks/does the right thing through correct reasoning.
AI is the study of mental faculties through the use of computational models.
Develop formal models of knowledge representation, reasoning, learning, memory, problem solving that can be rendered in algorithms.

Greek Philosopher- Aristotle: provided the correct arguments/ thought structures that always gave correct conclusions given correct premises.

- Abebe is a man; all men are mortal; therefore Abebe is mortal
- These Laws of thought governed the operation of the mind and initiated the field of Logic.

By 1965, programs existed that could, in principle, solve any solvable problem described in logical notation. The so-called logicist tradition within artificial intelligence hopes to build on such programs to create intelligent systems. There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between being able to solve a problem "in principle" and doing so in practice. Although both of these obstacles apply to any attempt to build computational reasoning systems, they appeared first in the logicist tradition.

### 1.4.3. Acting humanly: The Turing Test

Can machines act like human do? Can machines behave intelligently?
The Turing Test, proposed by Alan Turing (195O), was designed to provide a satisfactory operational definition of intelligence.

**Turing Test approach**

- 3 rooms contain – a person, a computer and an Interrogator.
- The Interrogator can communicate with the other 2 by teletype.
- The Interrogator tries to find which one is the person & which one is machine.
- The machine tries to fool the Interrogator to believe that it is human and the person also tries to convince the Interrogator that it is human.
- It the machine succeed in fooling the Interrogator, then conclude that the machine is intelligent.
- Goal – to develop system that are behave like human.

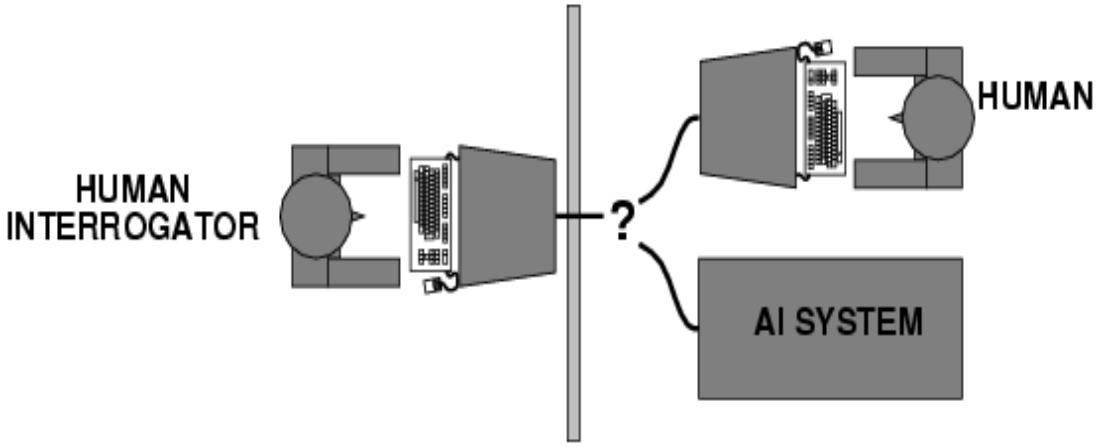



*Figure 1.2: Turing Test Approach*

We note that programming a computer to pass the test provides plenty to work on. The computer would need to possess the following capabilities:

- Natural language processing to enable it to communicate successfully in English.

- ♣ knowledge representation to store what it knows or hears;
- ♣ Automated reasoning to use the stored information to answer questions and to draw new conclusions;
- ♣ Machine learning to adapt to new circumstances and to detect and extrapolate patterns.

Turing's test deliberately avoided direct physical interaction between the interrogator and the computer, because physical simulation of a person is unnecessary for intelligence. To pass the total Turing Test, the computer will need,

- ♣ Computer vision to perceive objects, and
- ♣ Robotics to manipulate objects and move about.

### 1.4.4. Acting Rationally: The Rational Agent Approach

The approach is Doing the right things
An agent is just something that acts (agent comes from the Latin agere, to do). But computer agents are expected to have other attributes that distinguish them from mere "programs," such as

1. Operating under autonomous control,
2. Perceiving their environment,
3. Persisting over a prolonged time period,
4. Adapting to change, and
5. Being capable of taking on another's goals.

A rational agent is one that acts so as to achieve the best outcome.
In the "laws of thought" approach to AI, the emphasis was on correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals and then to act on that conclusion. On the other hand, correct inference is not all of rationality, because there are often situations where there is no provably correct thing to do, yet something must still be done. There are also ways of acting rationally that cannot be said to involve inference. For example, recoiling from a hot stove is a reflex action that is usually more successful than a slower action taken after careful deliberation.

All the skills needed for the Turing Test are there to allow rational actions. Thus, we need the ability to represent knowledge and reason with it because this enables us to reach good decisions in a wide variety of situations. We need to be able to generate comprehensible sentences in natural language because saying those sentences helps us get by in a complex society. We need learning not just for erudition, but because having a better idea of how the world works enables us to generate more effective strategies for dealing with it. We need visual perception not just because seeing is fun, but to get a better idea of what an action might achieve.

For these reasons, the study of AI as rational-agent design has at least two advantages. First, it is more general than the "laws of thought" approach, because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is more amenable to scientific development than are approaches based on human behavior or human thought because the standard of rationality is clearly defined and completely general. Human behavior, on the other

hand, is well-adapted for one specific environment and is the product, in part, of a complicated and largely unknown evolutionary process that still is far from producing perfection.

### 1.5.Foundations of AI

In this section, we will see a brief history of the disciplines that contributed ideas, viewpoints, and techniques to AI.

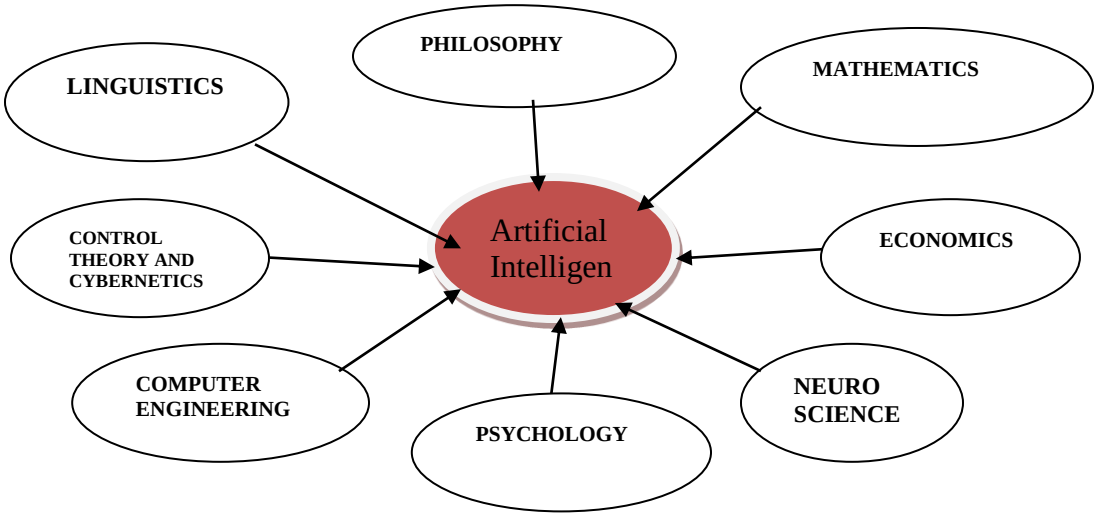



*Figure 1.3: foundation of AI*

PHILOSOPHY

- ☺ Can formal rules be used to draw valid conclusions?
- ☺ How does the mental mind arise from a physical brain?
- ☺ Where does knowledge come from?
- ☺ How does knowledge lead to action?

MATHEMATICS

- ☺ What are the formal rules to draw valid conclusions?
- ☺ What can be computed?
- ☺ How do we reason with uncertain information?

ECONOMICS

- ☺ How should we make decisions so as to maximize pay-off?
- ☺ How should we do this when others may not go along?
- ☺ How should we do this when the pay-off may be far in the future?

NEUROSCIENCE

- ☺ How do brains process information?

Brains and digital computers perform quite different tasks and have different properties. Figure 1.4 shows that there are 1000 times more neurons in the typical human brain than there are gates in the CPU of a typical high-end computer. Computer chips can execute an instruction in a nanosecond, whereas neurons are millions of times slower. Brains more than make up for this, however, because all the neurons and synapses are active simultaneously, whereas most current computers have only one or at most a few CPUs. Thus, even though a computer is a million times faster in raw switching speed, the brain ends up being 100,000 times faster at what it does.

| | Computer | Human Brain |
|---|---|---|
| Computational Units | 1 CPU, 108 gates | 1011 neurons |
| Storage Units | 1011 bits RAM<br>1012 bits disk | 1011 neurons<br>1014 synapses |
| Cycle time | 10-9 sec | 10-3 sec |
| Bandwidth | 1010 bits/sec | 1014 bits/sec |
| Memory updates/sec | 109 | 1014 |

*Figure 1.4 a crude comparison of the raw computational resources available to computers and brains.*

PSYCHOLOGY

- ☺ How do humans and animals think and act?

COMPUTER ENGINEERING

- ☺ How can we build an efficient computer?

CONTROL THEORY AND CYBERNETICS

- ☺ How can artifacts operate under their own control?

LINGUISTICS

- ☺ How does language relate to thought?

**1.6.The History of Artificial Intelligence**

**The gestation of artificial intelligence (1943-1955)**
There were a number of early examples of work that can be characterized as AI, but it was Alan Turing who first articulated a complete vision of AI in his 1950 article "Computing Machinery

and Intelligence." Therein, he introduced the Turing test, machine learning, genetic algorithms, and reinforcement learning.

**The birth of artificial intelligence (1956)**

McCarthy convinced Minsky, Claude Shannon, and Nathaniel Rochester to help him bring together U.S. researchers interested in automata theory, neural nets, and the study of intelligence. They organized a two-month workshop at Dartmouth in the summer of 1956. Perhaps the longest-lasting thing to come out of the workshop was an agreement to adopt McCarthy's new name for the field: artificial intelligence.

**Early enthusiasm, great expectations (1952-1969)**

The early years of AI were full of successes-in a limited way. General Problem Solver (GPS) was a computer program created in 1957 by Herbert Simon and Allen Newell to build a universal problem solver machine. The order in which the program considered sub goals and possible actions was similar to that in which humans approached the same problems. Thus, GPS was probably the first program to embody the "thinking humanly" approach. At IBM, Nathaniel Rochester and his colleagues produced some of the first AI programs. Herbert Gelernter (1959) constructed the Geometry Theorem Prover, which was able to prove theorems that many students of mathematics would find quite tricky. Lisp was invented by John McCarthy in 1958 while he was at the Massachusetts Institute of Technology (MIT). In 1963, McCarthy started the AI lab at Stanford.

**A dose of reality (1966-1973)**

From the beginning, AI researchers were not shy about making predictions of their coming successes. The following statement by Herbert Simon in 1957 is often quoted: “It is not my aim to surprise or shock you-but the simplest way I can summarize is to say that there are now in the world machines that think, that learn and that create. Moreover their ability to do these things is going to increase rapidly until-in a visible future-the range of problems they can handle will be coextensive with the range to which the human mind has been applied.

**Knowledge-based systems: The key to power? (1969-1979)**

Dendral was an influential pioneer project in artificial intelligence (AI) of the 1960s, and the computer software expert system that it produced. Its primary aim was to help organic chemists in identifying unknown organic molecules, by analyzing their mass spectra and using knowledge of chemistry. It was done at Stanford University by Edward Feigenbaum, Bruce Buchanan, Joshua Lederberg, and Carl Djerassi.

**AI becomes an industry (1980-present)**

In 1981, the Japanese announced the "Fifth Generation" project, a 10-year plan to build intelligent computers running Prolog. Overall, the AI industry boomed from a few million dollars in 1980 to billions of dollars in 1988.

**The return of neural networks (1986-present)**

Psychologists including David Rumelhart and Geoff Hinton continued the study of neural-net models of memory.

**AI becomes a science (1987-present)**

In recent years, approaches based on hidden Markov models (HMMs) have come to dominate the area.

Speech technology and the related field of handwritten character recognition are already making the transition to widespread industrial and consumer applications.

The Bayesian network formalism was invented to allow efficient representation of, and rigorous reasoning with, uncertain knowledge.

**The emergence of intelligent agents (1995-present)**

One of the most important environments for intelligent agents is the Internet.

## 1.7.The state of art

**What can AI do today?**

Here are some example applications

- Computer vision: face recognition from a large set
- Robotics: autonomous (mostly) auto mobile
- Natural language processing: simple machine translation
- Expert systems: medical diagnosis in a narrow domain
- Spoken language systems: ~1000 word continuous speech
- Planning and scheduling: Hubble Telescope experiments
- Learning: text categorization into ~1000 topics
- User modelling: Bayesian reasoning in Windows help (the infamous paper clip…)
- Games: Grand Master level in chess (world champion), checkers, etc.

**What can't AI systems do yet?**

- Understand natural language robustly (e.g., read and understand articles in a newspaper)
- Surf the web
- Interpret an arbitrary visual scene
- Learn a natural language
- Construct plans in dynamic real-time domains
- Perform life-long learning

## 1.8.Proposing and evaluating Application of AI

Buying off-the-shelf AI (Artificial Intelligence) software is a good first step for those companies that are new to the technology. There should be little need to make investments in technical infrastructure or to hire expensive data sciences. There will also be the benefit of getting a solution that has been tested by other customers. For the most part, there should be confidence in the accuracy levels as the algorithms will probably be implemented properly. But there is a nagging issue: there are many AI applications on the market and it is extremely difficult to determine which the best option is. After all, it seems that most technology vendors are extolling their AI capabilities as a way to stand out from the crowd.

Then what are some factors to consider when evaluating a new solution? Let's take a look at the following:

**Data Connectors**: AI is useless without data. It is the fuel for the insights.

**Flexibility**: AI does not have general scope. Instead, it is focused on particular use cases. This is known a "weak AI."

**Ease-of-use**: This is absolutely critical. The user of AI is often a non-technical person. If the application is complex, there could easily be little adoption.

**Ethical AI**: Even if the application is accurate, there could be risks. The data may have inherent biases, which could skew the results. This is why you should get an explanation of the data and how it is used.
**Costs**: "If you're the first customer in a specific industry for an AI vendor, then you're a very valuable customer to have and you can use that as negotiating leverage for a beneficial contract.

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast.

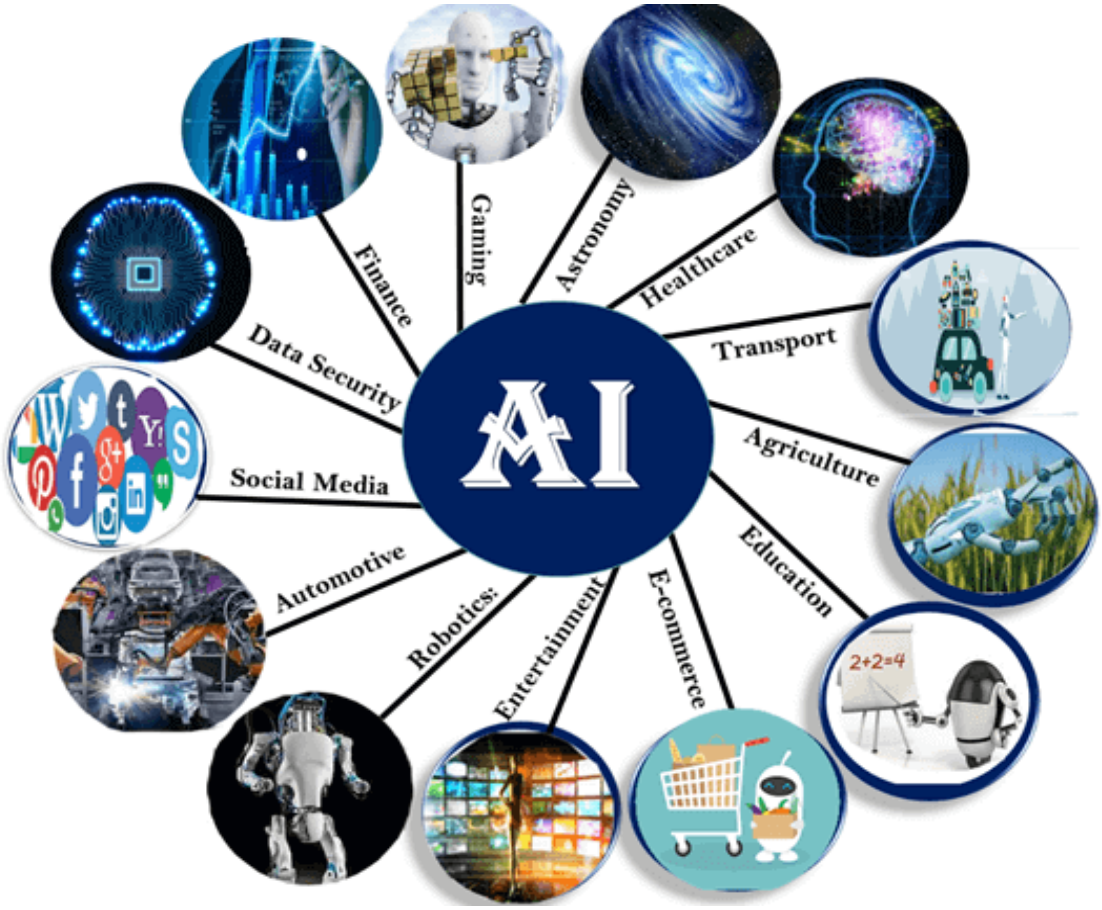



*Figure. 1.5: Those are some sectors which have the application of Artificial Intelligence:*

# Chapter 2: Intelligent Agents

## 2.1. Introduction

An agent is anything that can be viewed as perceiving its environment through sensors and sensor acting upon that environment through actuators. This simple idea is illustrated in Fig. 2.1.

- ✓ A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
- ✓ A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- ✓ A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

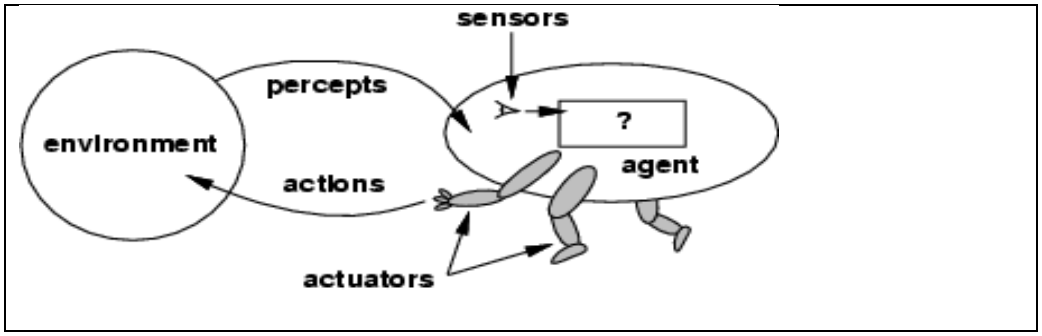



Figure 2.1: Agents interact with environments through sensors and actuators.

## 2.2.Agents and environments - Percept

We use the term percept to refer to the agent's perceptual inputs at any given instant. Percept Sequence An agent's percept sequence is the complete history of everything the agent has ever perceived.
Agent function mathematically speaking, we say that an agent's behaviour is described by the agent function that maps any given percept sequence to an action.

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

Agent programs Internally, The agent function for an artificial agent will be implemented by an agent program. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running on the agent architecture.

The agent program runs on the physical architecture to produce *f*

- agent = architecture + program

### 2.2.1. Inputs and outputs of an agent

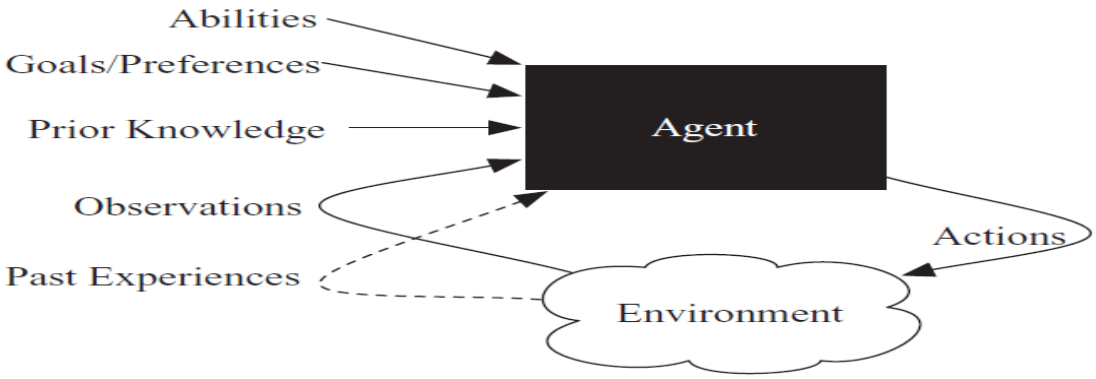



*Figure 2.2: Inputs and outputs of an agent*

- Prior knowledge about the agent and the environment;
- History of interaction with the environment, which is composed of
  - Observations of the current environment and
  - Past experiences of previous actions and observations, or other data, from which it can learn;
- Goals that it must try to achieve or preferences over states of the world; and
- Abilities, which are the primitive actions it is capable of carrying out.

To illustrate these ideas, we will use a very simple example-The vacuum-cleaner world
This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. A partial tabulation of this agent function is shown in Figure 2.3 :

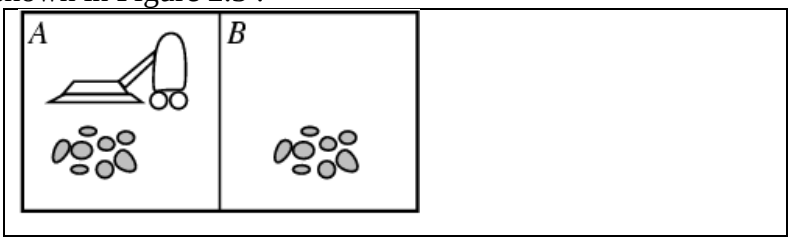



*Figure 2.3: A vacuum-cleaner world with just two locations.*

- Agent: robot vacuum cleaner
- Sensors:
  - dirt sensor: detects when floor in front of robot is dirty
  - bump sensor: detects when it has bumped into something
  - power sensor: measures amount of power in battery
  - bag sensor: amount of space remaining in dirt bag
- Actuators/Effectors:
  - motorized wheels
  - suction motor
- Environment: square A and B

- Percepts: [location and content] e.g. [A, Dirty]
- Actions: left, right, suck, and no-op

agent program

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

| Percept sequence | Action |
|---|---|
| [A, Clean] | Right |
| [A, Dirty] | Suck |
| [B, Clean] | Left |
| [B, Dirty] | Suck |
| [A, Clean], [A, Clean] | Right |
| [A, Clean], [A, Dirty] | Suck |
| [A, Clean], [A, Clean], [A, Clean] | Right |
| [A, Clean], [A, Clean], [A, Dirty] | Suck |

### 2.3.Good Behavior: The Concept of Rationality

A rational agent is one that does the right thing-conceptually speaking, every entry in the table for the agent function is filled out correctly. Obviously, doing the right thing is better than doing the wrong thing, but what does it mean to do the right thing? As a first approximation, we will say that the right action is the one that will cause the agent to be most successful. Therefore, we will need some way to measure success. Together with the description of the environment and the sensors and actuators of the agent, this will provide a complete specification of the task facing the agent.

#### 2.3.1. Rationality

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.

This leads to a definition of a rational agent:
For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

### 2.3.2. Performance measures

A performance measure embodies the criterion for success of an agent's behavior. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well.

- Some criticisms on the use of a performance measure are listed below:

✓ With a performance measure, we establish a standard of what it means to be successful in an environment and use it to measure the performance of agents.
✓ We should ask whether or not our performance measures are objective ones.
✓ Also, we should ask whether we impose quantitative or qualitative measures of performance.
✓ We should measure the performance over the long run.

Consider the vacuum-cleaner agent from the preceding section. We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift. With a rational agent, of course, what you ask for is what you get. A rational agent can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor, then cleaning it up again, and so on. A more suitable performance measure would reward the agent for having a clean floor. For example, one point could be awarded for each clean square at each time step (perhaps with a penalty for electricity consumed and noise generated). As a general rule, it is better to design performance measures according to what one actually wants in the environment, rather than according to how one thinks the agent should behave.

- E.g., Performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.

### 2.3.3. Omniscience, learning, and autonomy

An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.
Rationality maximizes expected performance, while perfection maximizes actual performance. Retreating from a requirement of perfection is not just a question of being fair to agents. The point is that if we expect an agent to do what turns out to be the best action after the fact, it will be impossible to design an agent to fulfill this specification-unless we improve the performance of crystal balls or time machines.
Our definition of rationality does not require omniscience, then, because the rational choice depends only on the percept sequence to date. We must also ensure that we haven't inadvertently allowed the agent to engage in decidedly under intelligent activities. For example, if an agent does not look both ways before crossing a busy road, then its percept sequence will not tell it that there is a large truck approaching at high speed. Does our definition of rationality say that it's now OK to cross the road? Far from it! First, it would not be rational to cross the road given this uninformative percept sequence: the risk of accident from crossing without looking is too great. Second, a rational agent should choose the "looking" action before stepping into the street, because looking helps maximize the expected performance. Doing actions in order to modify future percepts-sometimes called information gathering-is an important part of rationality. A second example of information gathering is provided by the exploration that must be undertaken by a vacuum-cleaning agent in an initially unknown environment.

Our definition requires a rational agent not only to gather information, but also to learn as much as possible from what it perceives. The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented. There are extreme cases in which the environment is completely known a priori. In such cases, the agent need not perceive or learn; it simply acts correctly. Of course, such agents are very fragile.
Successful agents split the task of computing the agent function into three different periods: when the agent is being designed, some of the computation is done by its designers; when it is deliberating on its next action, the agent does more computation; and as it learns from experience, it does even more computation to decide how to modify its behavior.
To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks autonomy. A rational agent should be autonomous-it should learn what it can to compensate for partial or incorrect prior knowledge. For example, a vacuum-cleaning agent that learns to foresee where and when additional dirt will appear will do better than one that does not. As a practical matter, one seldom requires complete autonomy from the start: when the agent has had little or no experience, it would have to act randomly unless the designer gave some assistance.

### 2.4.The Structure of Intelligent Agents - Task environments

We must think about task environments, which are essentially the "problems" to which rational agents are the "solutions." Before we design an intelligent agent, we must specify its “task environment”, define the problem.

SPECIFYING THE TASK ENVIRONMENT
In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent's actuators and sensors. We will group all these together under the heading of the task environment.
Environment is the surrounding areas that the agent perceives information from.

PEAS:

- Performance measure
- Environment
- Actuators
- Sensors

For the acronymically minded, we call this the
PEAS (Performance, Environment, Actuators, and Sensors) description.
In designing an agent, the first step must always be to specify the task environment as fully as possible. The vacuum world was a simple example;
let us consider a more complex problem: an automated taxi driver. The full driving task is extremely *open-ended.* There is no limit to the novel combinations of circumstances that can arise.
First, what is the **performance measure** to which we would like our automated driver to aspire? Desirable qualities include getting to the correct destination; minimizing fuel consumption and wear and tear; minimizing the trip time and/or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so there will be tradeoffs involved.

Next, what is the driving environment that the taxi will face? Any taxi driver must deal with a variety of roads. The roads contain other traffic, pedestrians, stray animals, road works, police cars, puddles, and potholes. The taxi must also interact with potential and actual passengers. The actuators available to an automated taxi will be more or less the same as those available to a human driver: control over the engine through the accelerator and control over steering and braking. In addition, it will need output to a display screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles.
To achieve its goals in the driving environment, the taxi will need to know where it is, what else is on the road, and how fast it is going. Its basic sensors should therefore include one or more controllable TV cameras, the speedometer, and the odometer. To control the vehicle properly, especially on curves, it should have an accelerometer; it will also need to know the mechanical state of the vehicle, so it will need the usual array of engine and electrical system sensors. It might have instruments that are not available to the average human driver: a satellite global positioning system (GPS) to give it accurate position information with respect to an electronic map, and infrared or sonar sensors to detect distances to other cars and obstacles. Finally, it will need a keyboard or microphone for the passenger to request a destination.

| **Agent Type** | **Performance Environment** | **Environment** | **Actuators** | **Sensors** |
|---|---|---|---|---|
| Taxi driver | Safe: fast, legal, comfortable trip, maximize profits | Roads, other traffic, pedestrians, customers | Steering, accelerator, brake, signal, horn, display | Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard |

***Figure 2.4:*** *PEAS description of the task environment for an automated taxi.*
Example 2: Agent = Part-picking robot

- **Performance measure:** Percentage of parts in correct bins
- **Environment:** Conveyor belt with parts, bins
- **Actuators:** Jointed arm and hand
- **Sensors:** Camera, joint angle sensors

Example 3: Agent = Medical diagnosis system

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

Example 4: Agent = Interactive English tutor

- **Performance measure**: Maximize student's score on test

- **Environment**: Set of students
- **Actuators**: Screen display (exercises, suggestions, corrections)
- **Sensors**: Keyboard

### 2.5.Properties of Task Environments

The range of task environments that might arise in A1 is obviously vast. We can, however, identify a fairly small number of dimensions along which task environments can be categorized. First, we list the dimensions, then we analyze several task environments to illustrate the ideas. Here are some kinds of environments.

- Fully Observable Vs. Partially Observable.
- Deterministic Vs. Nondeterministic.
- Episodic Vs. Non-Episodic.
- Static Vs. Dynamic.
- Single Agent Vs. Multi Agent.
- Discrete Vs. Continuous.

**1. FULLY OBSERVABLE vs. PARTIALLY OBSERVABLE.**

If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are relevant to the choice of action; relevance, in turn, depends on the performance measure. An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data-for example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and an automated taxi cannot see what other drivers are thinking.

**2. DETERMINISTIC vs. STOCHASTIC.**

If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise, it is stochastic. In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment. If the environment is partially observable, however, then it could appear to be stochastic. Taxi driving is clearly stochastic in this sense, because one can never predict the behavior of traffic exactly; moreover, one's tires blow out and one's engine seizes up without warning. If the environment is deterministic except for the actions of other agents, we say that the environment is strategic.

**3. EPISODIC vs. SEQUENTIAL.**

In an episodic task environment, the agent's experience is divided into atomic episodes. Each episode consists of the agent perceiving and then performing a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. In episodic environments, the choice of action in each episode depends only on the episode itself. Many classification tasks are episodic. In sequential environments, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

**4. STATIC vs. DYNAMIC.**

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. Taxi driving is clearly dynamic: the other cars and the taxi itself keep moving while the driving algorithm dithers about what to do next.

**5. SINGLE AGENT vs. MULTI AGENT.**

The distinction between single-agent and multi agent environments may seem simple enough. An agent operating by itself in an environment. Does the other agent interfere with the performance measure?.

There are, however, some subtle issues. First, we have described how an entity may be viewed as an agent, but we have not explained which entities must be viewed as agents. Does an agent A (the taxi driver for example) have to treat an object B (another vehicle) as an agent, or can it be treated merely as a stochastically behaving object, analogous to waves at the beach or leaves blowing in the wind? The key distinction is whether B's behavior is best described as maximizing a performance measure whose value depends on agent A's behavior. For example, in chess, the opponent entity B is trying to maximize its performance measure, which, by the rules of chess, minimizes agent A's performance measure. Thus, chess is a competitive multi agent environment. In the taxi-driving environment, on the other hand, avoiding collisions maximizes the performance measure of all agents, so it is a partially cooperative multi agent environment. It is also partially competitive because, for example, only one car can occupy a parking space.

**6. Discrete vs. Continuous**

Are the distinct percepts & actions limited or unlimited?

If there are a limited number of distinct, clearly defined percepts and actions, we say the environment is discrete.

- Taxi driving is continuous - speed location are in a range of continuous values.
- Chess is discrete - there are a fixed number of possible moves on each item

### 2.6.Agent Types

The agent programs we will see all have the same skeleton, they take the current percept as input from the sensors and return an action to the actuators. Notice the difference between the agent program, which takes the current percept as input, and the agent function, which takes the entire percept history. The agent program takes just the current percept as input because nothing more is available from the environment; if the agent's actions depend on the entire percept sequence, the agent will have to remember the percepts.

In the remainder of this section, we outline four basic kinds of agent program that embody the principles underlying almost all intelligent systems:

- ☺ Simple reflex agents;
- ☺ Model-based reflex agents;
- ☺ Goal-based agents; and
- ☺ Utility-based agents.
- ☺ Learning Agents

1. SIMPLE REFLEX AGENTS: The simplest kind of agent is the simple reflex agent. These agents select actions on the basis of the current percept, ignoring the rest of the percept history. It works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule.

Imagine yourself as the driver of the automated taxi.
E.g. If the car in front brakes, and its brake lights come on, then the driver should notice this and initiate braking,

- Some processing is done on the visual input to establish the condition. If "The car in front is braking"; then this triggers some established connection in the agent program to the action "initiate braking". We call such a connection a condition-action rule written as: **If car-in-front-is breaking then initiate-braking**.

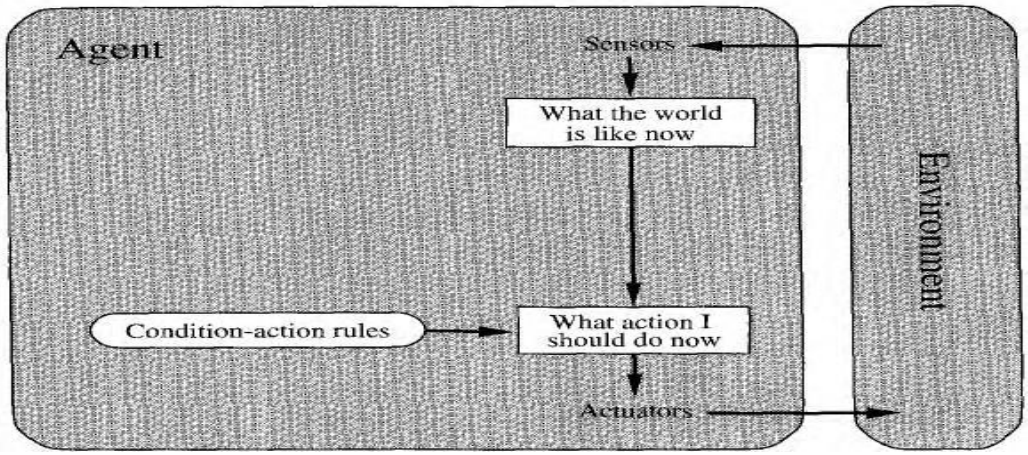



***Figure 2.5*** *Schematic diagram of a simple reflex agent.*

A more general and flexible approach for building an agent program is first to build a general-purpose interpreter for condition-action rules and then to create rule sets for specific task environments. Figure 2.4 gives the structure of this general program in schematic form, showing how the condition-action rules allow the agent to make the connection from percept to action.

We use rectangles to denote the current internal state of the agent's decision process and ovals to represent the background information used in the process.

**function** SIMPLE-REFLEX-AGENT(*percepts*) returns an action

```
static: rules, a set of condition-action rules
state ← INTERPRET-INPUT(percepts)
rule ← RULE-MATCH(state, rules)
action ←RULE-ACTION[rules]
return action
```

***Figure 2.6*** *A simple reflex agent.* It acts according to a rule whose condition matches the current state, as defined by the percept.

The INTERPRET-INPUT function generates an abstracted description of the current state from the percept, and the RULE-MATCH function returns the first rule in the set of rules that matches the given state description.

## 1. MODEL-BASED REFLEX AGENTS

- An agent that uses a description of how the next state depends on current state and action (model of the world) is called a **model-based agent**.
- This is a reflex agent with internal state.
    - It keeps track of the world that it can't see now.
- It works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state)
    - If the car is a recent model -- there is a centrally mounted brake light. With older models, there is no centrally mounted, so what if the agent gets confused?
    - Is it a parking light? Is it a brake light? Is it a turn signal light?
    - Some sort of internal state should be in order to choose an action.
    - The camera should detect two red lights at the edge of the vehicle go ON or OFF simultaneously.
- The driver should look in the rear-view mirror to check on the location of near by vehicles. In order to decide on lane-change the driver needs to know whether or not they are there.
    - The driver sees, and there is already stored information, and then does the action associated with that rule.

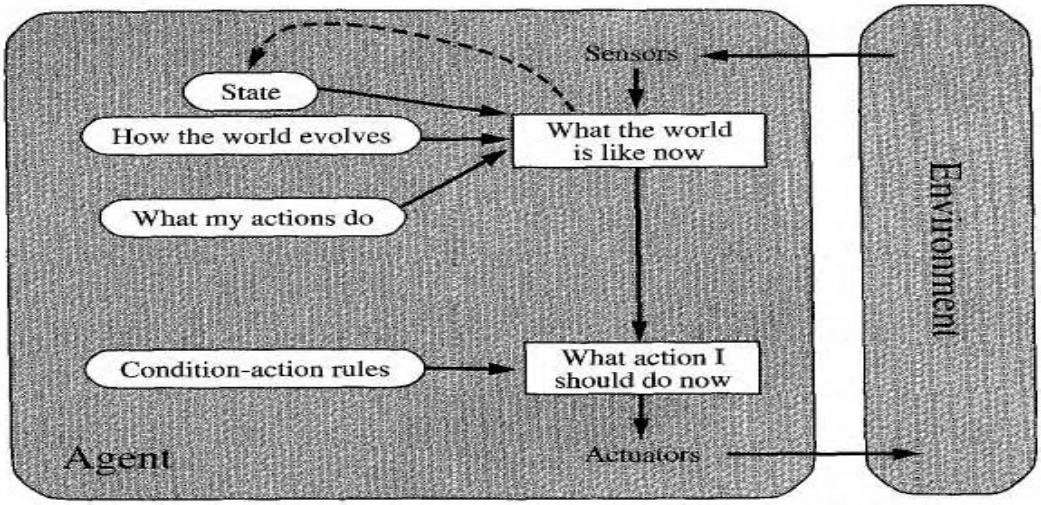



***Figure 2.7:*** *A model-based reflex agent.*

Figure 2.7: gives the structure of the reflex agent with internal state, showing how the current percept is combined with the old internal state to generate the updated description of the current state. The agent program is shown in Figure 2.8. The interesting part is the function UPDATE-STATE, which is responsible for creating the new internal state description.

```
function REFLEX-AGENT-WITH-STATE(percept)returns an action
        static: state, a description of the current world state
                rules, a set of condition-action rules
                action, the most recent action, initially none
        state ← UPDATE-STATE(state, action, percept)
        rule ← RULE-MATCH(state, rules)
        action ←RULE-ACTION[rule]
        return action
```

***Figure 2.8*** *A model-based reflex agent.* It keeps track of the current state of the world using an internal model. It then chooses an action in the same way as the reflex agent.

**2. GOAL-BASED AGENTS**

- Choose actions that achieve the goal (an agent with explicit goals)
- Involves consideration of the future:
  - ✓ **Knowing about the current state of the environment is not always enough to decide what to do.**
  - – For example, at a road junction, the taxi can turn left, right or go straight.
    - ✓ The right decision depends on where the taxi is trying to get to. As well as a current state description, the agent needs some sort of goal information, which describes situations that are desirable. E.g. being at the passenger's destination.
- The agent may need to consider long sequences, twists and turns to find a way to achieve a goal.

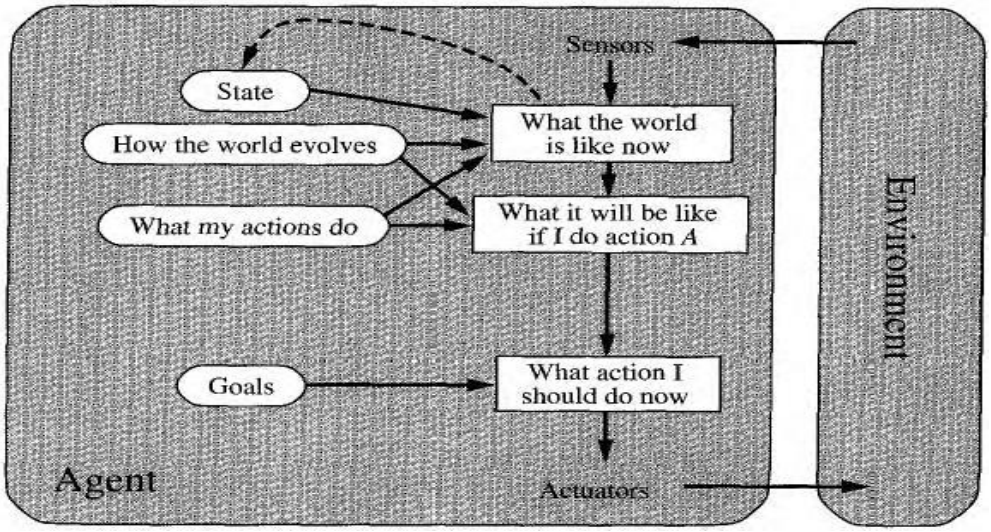



***Figure 2.9:*** *A goal-based agent.* It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

```
function GOAL_BASED_AGENT (percept) returns action
        state ← UPDATE-STATE (state, percept)
        action ← SELECT-ACTION [state, goal]
        state ← UPDATE-STATE (state, action)
 return action
```

***Figure 2.10*** *A Goal-based reflex agent program.*

**3. UTILITY-BASED AGENTS**

- Goals are not really enough to generate high quality behavior.

- For e.g., there are many action sequences that will get the taxi to its destination, thereby achieving the goal. Some are quicker, safer, more reliable, or cheaper than others. We need to consider Speed and safety

- When there are several goals that the agent can aim for, non of which can be achieved with certainty. Utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.
- An agent that possesses an explicit utility function can make rational decisions.

A **utility function** maps a state, which describes the associated degree of happiness.

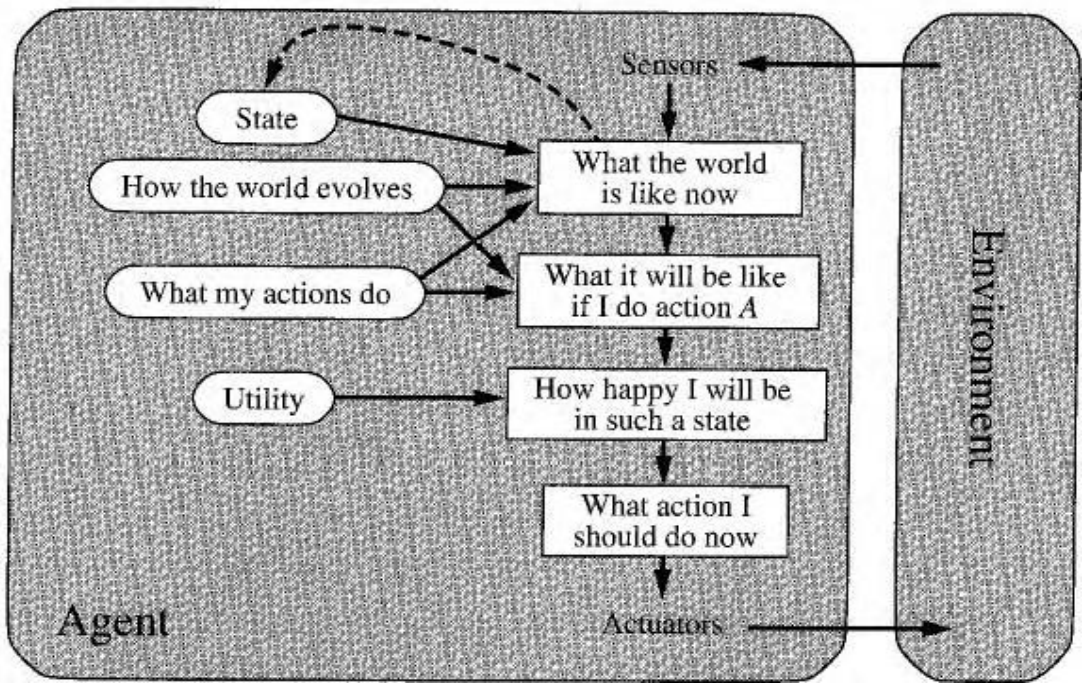



***Figure 2.11:*** *A utility-based agent. It uses a model of the world,* along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

```
function UTILITY_BASED_AGENT (percept) returns action
        state ← UPDATE-STATE (state, percept)
        action ← SELECT-OPTIMAL_ACTION [state, goal]
        state ← UPDATE-STATE (state, action)
   return action
```

***Figure 2.12:*** *A utility-based agent program*

**4. LEARNING AGENTS**

- How does an agent improve over time? By monitoring it's performance and suggesting better modelling, new action rules, etc. Learning has an advantage that it allows the agents to initially operate in unknown environments .
- A learning agent can be divided into four conceptual components,

- ✓ **learning element** which is responsible for making improvements.
- ✓ **performance element,** which is responsible for selecting external actions.
- ✓ **critic** gives feedback to learning element on how the agent is doing with respect to fixed performance standard and determines how the performance element should be modified to do better in the future.
- ✓ **problem generator** It is responsible for suggesting actions that will lead to new and informative experiences.

- E.g. automate taxi: using Performance element the taxi goes out on the road and drives. The critic observes the shocking language used by other drivers. From this experience, the learning element is able to formulate a rule saying this was a bad action, and the performance element is modified by installing new rule. The problem generator might identify certain areas in need of improvement, such as trying out the brakes on different roads under different conditions

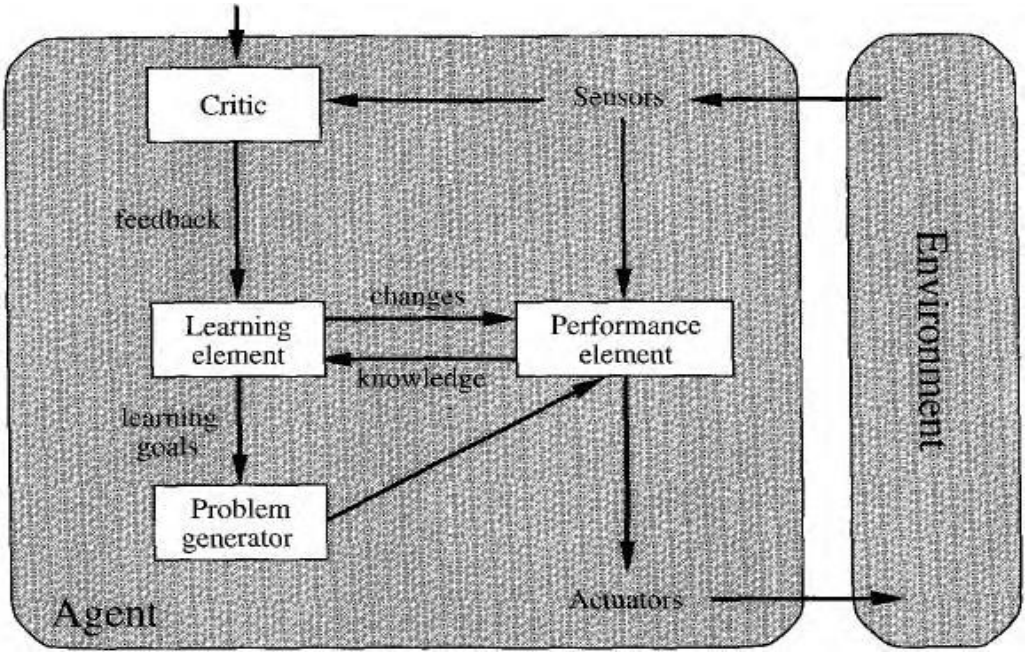
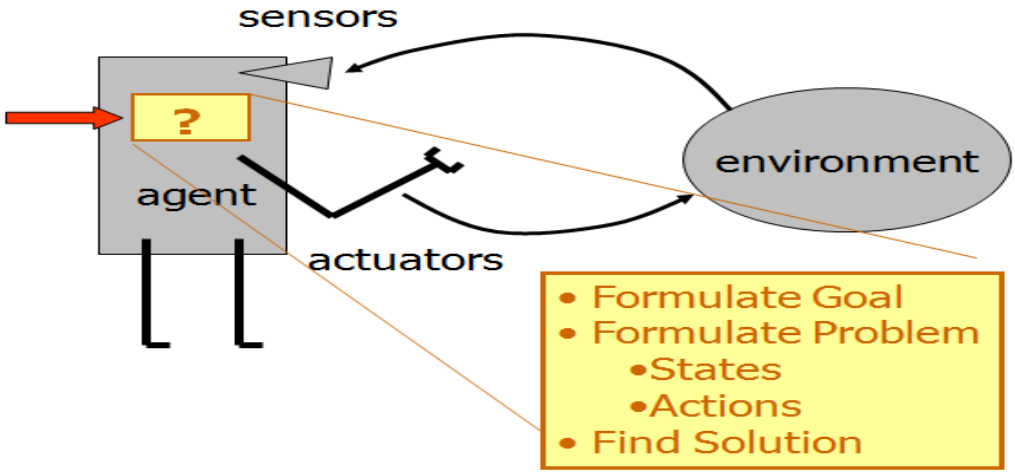



***Figure 2.13:****A general model of learning agents.*

# Chapter 3: Searching and planning

## 3.1. Solving problems by searching and planning

The simplest agents discussed in Chapter 2 were the reflex agents, which base their actions on a direct mapping from states to actions. Such agents cannot operate well in environments for which this mapping would be too large to store and would take too long to learn. Goal-based agents, on the other hand, can succeed by considering future actions and the desirability of their outcomes. One kind of goal-based agent called a problem-solving agent decides what to do by finding sequences of actions that lead to desirable states.




**Problem-Solving Agent**

## 3.2. Problem-Solving Agents

Intelligent agents are supposed to maximize their performance measure. Achieving this is sometimes simplified if the agent can adopt a goal and aim at satisfying it. Goals help organize behavior by limiting the objectives that the agent is trying to achieve.

Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.

We will consider a goal to be a set of world states-exactly those states in which the goal is satisfied.

The agent's task is to find out which sequence of actions will get it to a goal state. Before it can do this, it needs to decide what sorts of actions and states to consider.

Problem formulation is the process of deciding what actions and states to consider, given a goal.

In general, an agent with several immediate options of unknown value can decide what to do by just examining different possible sequences of actions that lead to states of known value, and then choosing the best sequence. This process of looking for such a sequence is called search.

A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out. This is called the execution phase.

Thus, we have a simple "formulate, search, execute" design for the agent.

After formulating a goal and a problem to solve, the agent calls a search procedure to solve it. It then uses the solution to guide its actions, doing whatever the solution recommends as the next thing to do-typically, the first action of the sequence-and then removing that step from the sequence. Once the solution has been executed, the agent will formulate a new goal.

***WELL-DEFINED PROBLEMS AND SOLUTIONS***

A problem can be defined formally by four components:

- ☺ The **initial state** that the agent starts in.
- ☺ **Actions**: A description of the possible **actions** available to the agent. The most common formulation uses a successor function. Together, **the initial state and successor function** implicitly define the **state space** of the problem-the set of all states reachable from the initial state.

  The state space forms a graph in which **the nodes are states and the arcs between nodes are actions**.
- ☺ **The Goal test,** which determines whether a given state is a goal state. Sometimes there is an explicit set of possible goal states, and the test simply checks whether the given state is one of them.
- ☺ A **Path cost** function that assigns a numeric cost to each path. **The problem-solving agent** chooses a cost function that reflects its own performance measure.

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

Solution quality is measured by the **path cost function**, and an optimal solution has the lowest path cost among all solutions. Some examples of problem formulation are given below.

**8 – Puzzle**

The 8-puzzle, an instance of which is shown in Figure 3.1, consists of a 3 x 3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space.

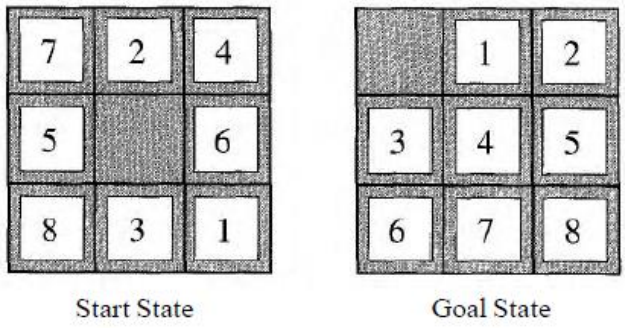



**Figure 3.1** A typical instance of the 8-puzzle.

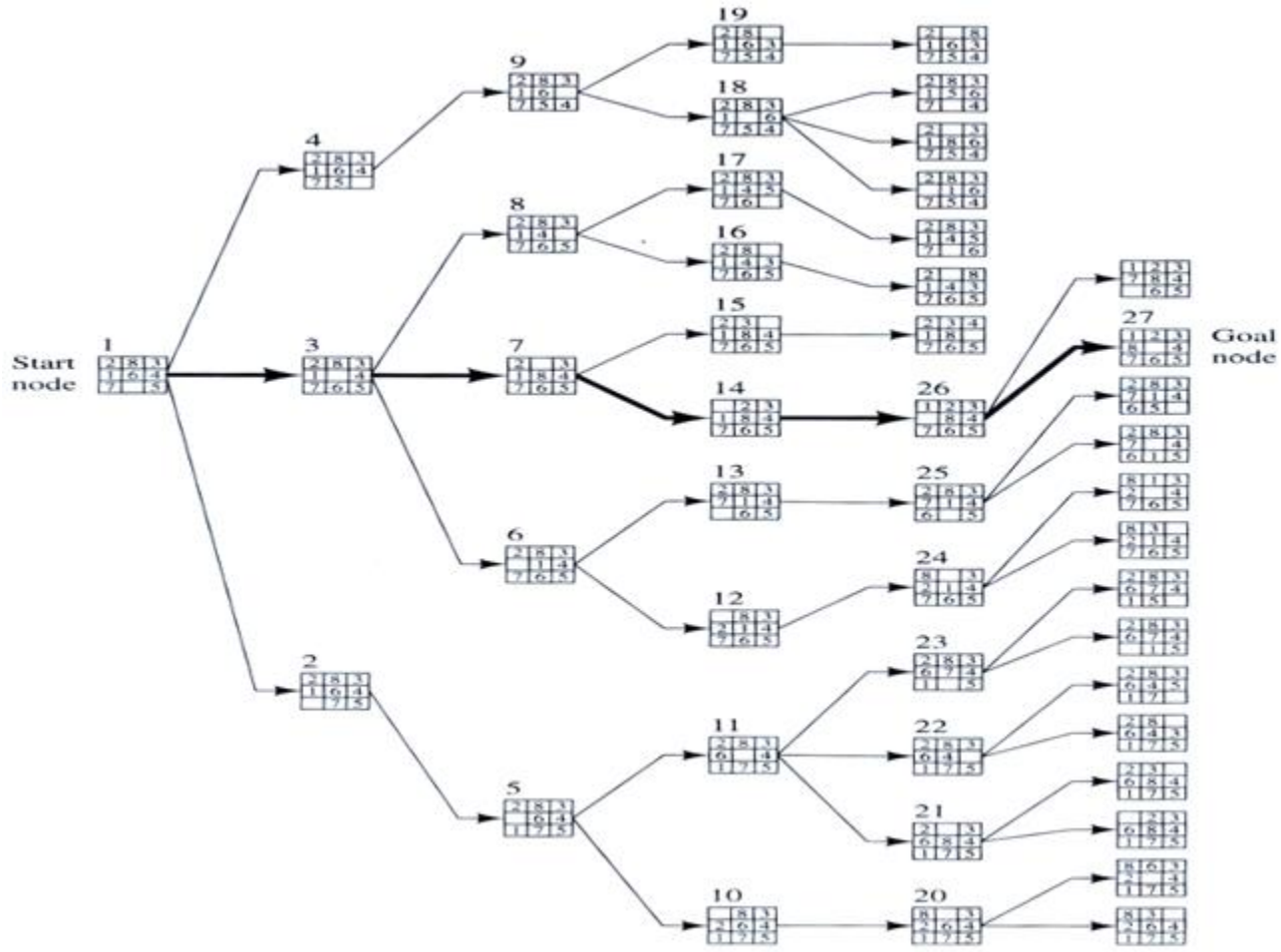
The object is to reach a specified goal state, such as the one shown on the right of the figure. The standard formulation is as follows:

- ☺ **States:** A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- ☺ **Initial state:** Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.

☺ **Successor function:** This generates the legal states that result from trying the four actions (blank moves Left, Right, Up, or Down).

☺ **Goal test:** This checks whether the state matches the goal configuration shown in Figure 3.1. (Other goal configurations are possible.)

☺ **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.




**8-queens problem**

The goal of the **8-queens problem** is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.) Figure 3.2 shows an attempted solution that fails: the queen in the rightmost column is attacked by the queen at the top left.

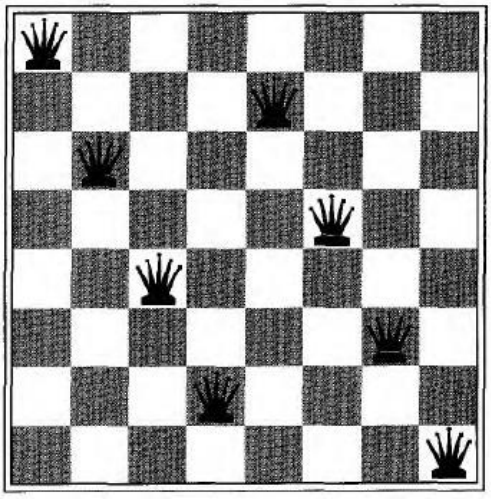


**Figure 3.2** Almost a solution to the 8-queens problem.

Although efficient special-purpose algorithms exist for this problem and the whole n-queens family, it remains an interesting test problem for search algorithms.

There are two main kinds of formulation.

An **incremental formulation** involves operators that *augment* the state description, starting with an empty state; for the 8-queens problem, this means that to the state.

A **complete-state formulation** starts with all 8 queens on the board and moves them around. In either case, the path cost is of no interest because only the final state counts.

The first incremental formulation one might try is the following:

- ☺ **States:** Any arrangement of 0 to 8 queens on the board is a state.
- ☺ **Initial state:** No queens on the board.
- ☺ **Successor function:** Add a queen to any empty square.
- ☺ **Goal test: 8** queens are on the board, none attacked.

### 3.2. Searching for Solutions

Having formulated some problems, we now need to solve them. This is done by a search through the state space. For now we will deal with search techniques that use an explicit **search tree** that is generated by the initial state and the successor function that together define the state space. In general, we may have a search *graph* rather than a search *tree*, when the same state can be reached from multiple paths.

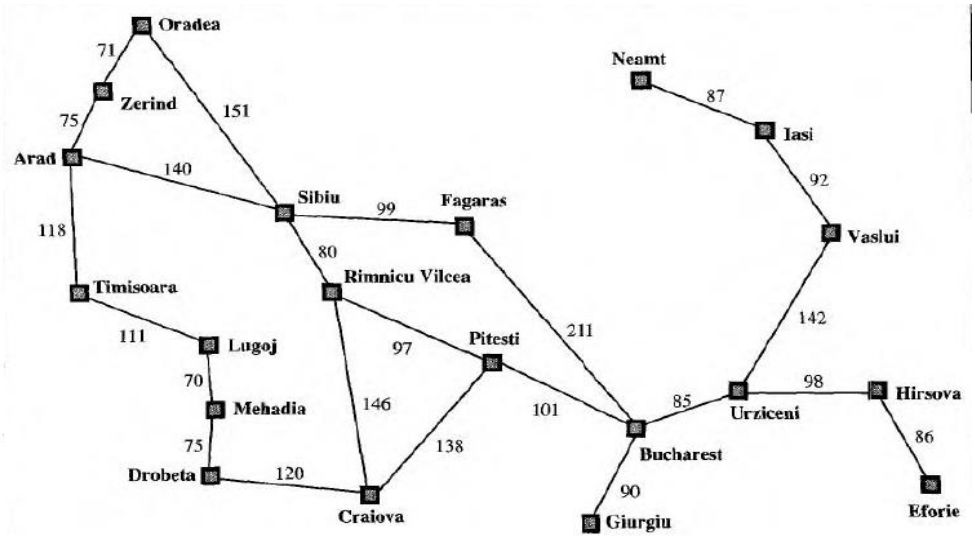



**Figure 3.3** A simplified road map of part of Romania.

For example let say an agent is in Arad and want to get to Bucharest (see Figure 3.3). Figure 3.4 shows some of the expansions in the search tree for finding a route from Arad to Bucharest.

The root of the search tree is a **search node** corresponding to the initial state, *In (Arad).* The first step is to test whether this is a goal state. Because this is not a goal state, we need to consider some other states. This is done by **expanding** the current state; that is, applying the successor function to the current state, thereby **generating** a new set of states. In this case, we get three new states: *In (Sibiu), In (Timisoara),* and *In (Zerind).* Now we must choose which of these three possibilities to consider further.

This is the essence of search-following up one option now and putting the others aside for later, in case the first choice does not lead to a solution. We continue choosing, testing, and expanding until either a solution is found or there are no more states to be expanded. The choice of which state to expand is determined by the **search strategy.**

It is important to distinguish between the state space and the search tree. For the route finding problem, there are only 20 states in the state space, one for each city. But there are an infinite number of paths in this state space, so the search tree has an infinite number of nodes. For example, the three paths Arad-Sibiu, Arad-Sibiu-Arad, Arad-Sibiu-Arad-Sibiu are the first three of an infinite sequence of paths. (Obviously, a good search algorithm avoids following such repeated paths.)

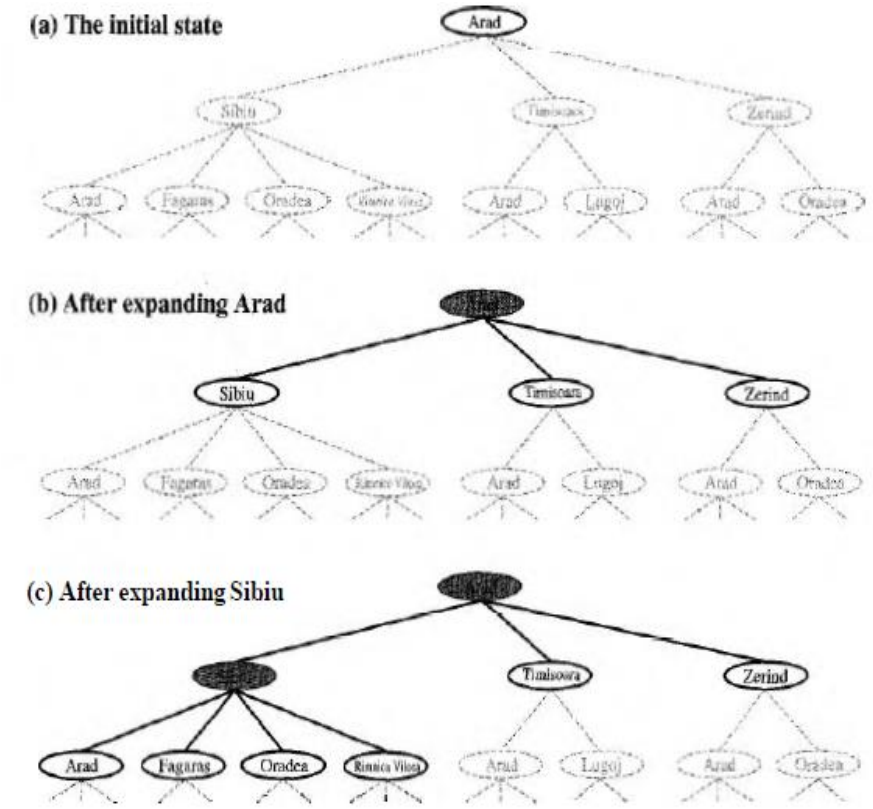



**Figure 3.4** Partial search trees for finding a route from Arad to Bucharest.

There are many ways to represent nodes, but we will assume that a node is a data structure with five components:

- **STATE:** the state in the state space to which the node corresponds;
- **PARENT-NODE:** the node in the search tree that generated this node;
- **ACTION:** the action that was applied to the parent to generate the node;
- **PATH-COST:** the cost, traditionally denoted by *g(n)*, of the path from the initial state to the node, as indicated by the parent pointers; and
- **DEPTH:** the number of steps along the path from the initial state.

It is important to remember the distinction between nodes and states.

**A node** is a bookkeeping 'data structure used to represent the search tree. A state corresponds to a configuration of the world.

Thus, nodes are on particular paths, as defined by PARENT-NODE pointers, whereas states are not. Furthermore, two different nodes can contain the same world state, if that state is generated via two different search paths.

**MEASURING PROBLEM-SOLVING PERFORMANCE**

The output of a problem-solving algorithm is either *failure* or a solution. (Some algorithms might get stuck in an infinite loop and never return an output.)

We will evaluate an algorithm's performance in four ways:

☺ **Completeness:** Is the algorithm guaranteed to find a solution when there is one?

- ☺ **Optimality:** Does the strategy find the optimal solution?
- ☺ **Time complexity:** How long does it take to find a solution?
- ☺ **Space complexity:** How much memory is needed to perform the search?

**Time and space** complexity are always considered with respect to some measure of the problem difficulty. In AI, graph is represented implicitly by **the initial state and successor function** and is frequently infinite, complexity is expressed in terms of **three quantities: b, the branching factor or maximum number of successors of any node; d, the depth of the shallowest goal node; and m, the maximum length of any path in the state space.** Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory.

## 3.3. Uninformed Search Strategies

This section covers five search strategies that come under the heading of **uninformed search** (also called **blind search**). The term means that they have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state.

Strategies that know whether one non- goal state is "more promising" than another are called **informed search** or **heuristic search** strategies.

The five strategies of uniformed search strategies are.

- ☺ Breadth-first search
- ☺ Uniform-cost search
- ☺ Depth-first search
- ☺ Iterative deepening depth-first search

### 3.3.1. Breadth-First Search

**Breadth-first search** is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

**Breadth-first search** can be implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out (FIFO) queue, assuring that the nodes that are visited first will be expanded first.

The **FIFO** queue puts all newly generated successors at the end of the queue, which means that shallow nodes are expanded before deeper nodes.

We will evaluate breadth-first search using the four criteria from the previous section. We can easily see that it is complete-if the shallowest goal node is at some finite depth d, breadth-first search will eventually find it after expanding all shallower nodes (provided the branching factor b is finite). The shallowest goal node is not necessarily the optimal one; technically, breadth-first

search is optimal if the path cost is a non-decreasing function of the depth of the node. (For example, when all actions have the same cost.)

- Complete? Yes it always reaches goal (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + (b^{d+1}-b)) = O(b^{d+1})$ (this is the number of nodes we generate)
- Space? $O(b^{d+1})$ (keeps every node in memory, either in fringe or on a path to fringe).
- Optimal? Yes (if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).

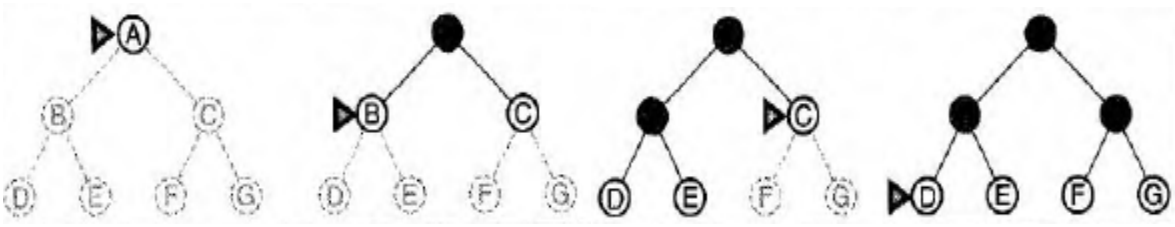



**Figure 3.5** Breadth-first search on a simple binary tree.

### 3.3.2. Uniform-Cost Search

Breadth-first search is optimal when all step costs are equal, because it always expands the *shallowest* unexpanded node. By a simple extension, we can find an algorithm that is optimal with any step cost function.

Instead of expanding the shallowest node, **uniform-cost search** expands the node n with the *lowest path cost.* Note that if all step costs are equal, this is identical to breadth-first search.

**Uniform-cost search** does not care about the *number* of steps a path has, but only about their total cost. Therefore, it will get stuck in an infinite loop if it ever expands a node that has a zero-cost action leading back to the same state (for example, a *NoOp* action).

**Uniform-cost search** is guided by path costs rather than depths, so its complexity cannot easily be characterized in terms of *b* and d. Instead, let C * be the cost of the optimal solution, and assume that every action costs at least ε. Then the algorithm's worst-case time and space complexity is $O\ (b^{\,1\,+\,[c^*/\,\varepsilon]})$, which can be much greater than $b^d$. This is because uniform-cost search can, and often does, explore large trees of small steps before exploring paths involving large and perhaps useful steps. When all step costs are equal, of course, $b^{\,1\,+\,[c^*/\,\varepsilon]}$, is just $b^d$.

Complete? Yes, if step cost ≥ ε (otherwise it can get stuck in infinite loops)

Time? No of nodes with *path cost* ≤ cost of optimal solution.

Space? No of nodes with path cost ≤ cost of optimal solution.

Optimal? Yes, for any step cost ≥ ε

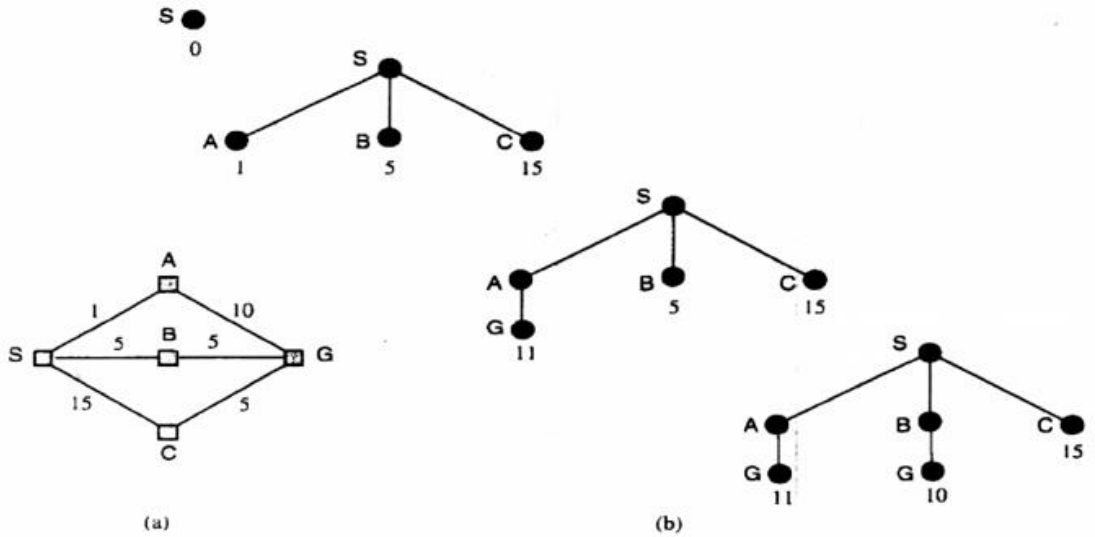



**Figure 3.6** A route-finding problem. (a) The state space, showing the cost of each operator. (b) Progression of the search.

### 3.3.3. Depth-First Search

***Depth-first search*** always expands the *deepest* node in the current fringe of the search tree. The progress of the search is illustrated in Figure 3.7. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the next shallowest node that still has unexplored successors.

This strategy can be implemented by TREE-SEARCH with a last-in-first-out (LIFO) queue, also known as a stack. As an alternative to the TREE-SEARCH implementation, it is common to implement depth-first search with a recursive function that calls itself on each of its children in turn.

The problem of unbounded trees can be alleviated by supplying depth-first search with a predetermined depth limit *l*.

That is, nodes at depth *l* are treated as if they have no successors. This approach is called **depth-limited search.** The depth limit solves the infinite-path problem.

**Drawback of Depth-first-search**

The drawback of depth-first-search is that it can make a wrong choice and get stuck going down very long(or even infinite) path when a different choice would lead to solution near the root of the search tree. For example, depth-first-search will explore the entire left subtree even if node C is a goal node.

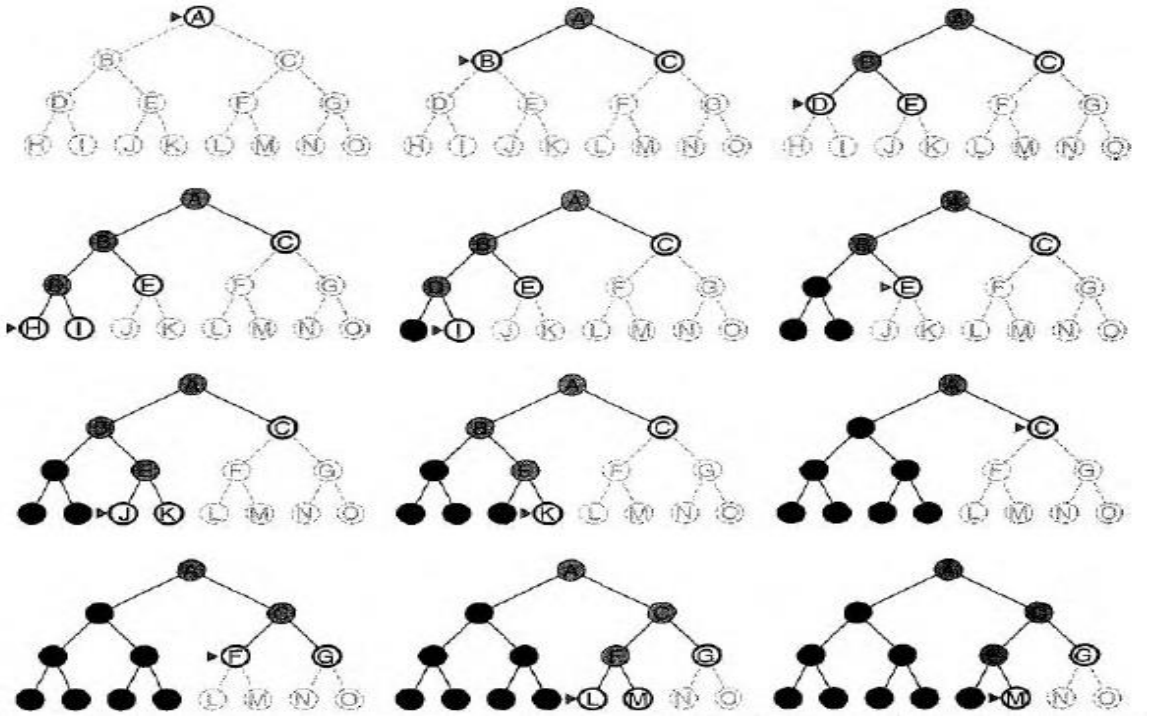


***Figure 3.7*** *Depth-first search on a binary tree.*

Properties of depth-first search.

- Complete? No: fails in infinite-depth spaces. Can modify to avoid repeated states along path
- Time? $O(b^m)$ with m=maximum depth terrible if *m* is much larger than *d.*
  - but if solutions are dense, may be much faster than breadth-first.
- Space? *O (bm),* i.e., linear space! (we only need to remember a single path + expanded unexplored nodes)
- Optimal? No (It may find a non-optimal goal first)

### 3.3.4. Iterative Deepening Depth-First Search

**Iterative deepening search** (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first search, which finds the best depth limit. It does this by gradually increasing the limit-first 0, then 1, then 2, and so on-until a goal is found. This will occur when the depth limit reaches d, the depth of the shallowest goal node.
**Iterative deepening** combines the benefits of depth-first and breadth-first search. **Like depth-first search**, its memory requirements are very modest: O(bd) to be precise. Like breadth-first search, it is complete when the branching factor is finite and optimal when the path cost is a non-decreasing function of the depth of the node. Figure 3.8 shows four iterations of ITERATIVE-DEEPENING-SEARCH on a binary search tree, where the solution is found on the third iteration.

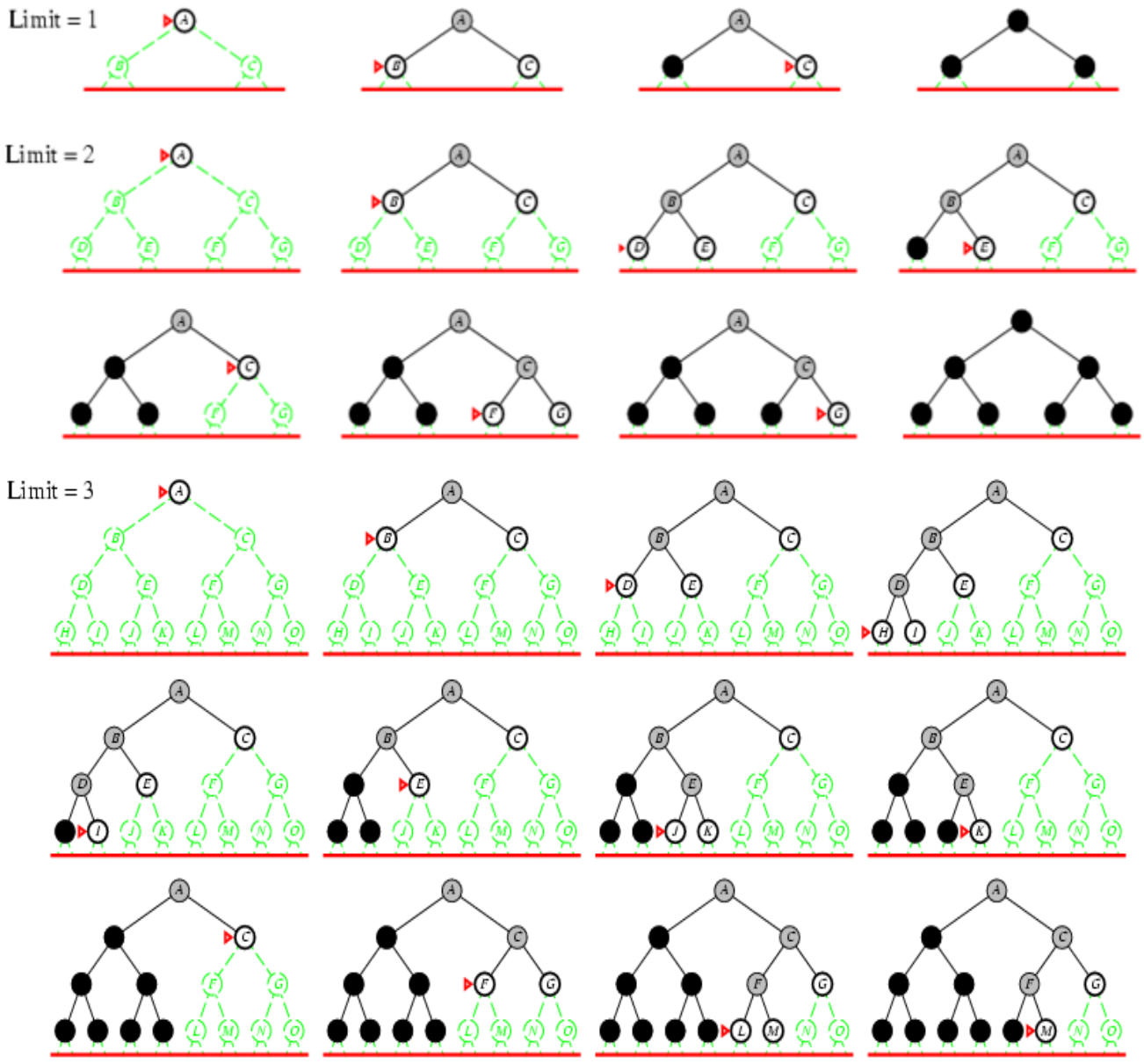



**Figure 3.8** Three iterations of iterative deepening search on a binary tree.

Properties of iterative deepening search.

- Complete? Yes
- Time? $O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 or increasing function of depth.

### 3.4. Informed (Heuristic) Search Strategies

Uninformed search strategies can find solutions to problems by systematically generating new states and testing them against the goal.
Unfortunately, these strategies are incredibly inefficient in most cases.

**But informed search** strategy--one that uses problem-specific knowledge beyond the definition of the problem itself-can find solutions more efficiently than an uninformed strategy.
The general approach for informed search strategy is called **best-first search.**
**Best-first search** is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an e**valuation function,** f (n) . Traditionally, the node with the *lowest* evaluation is selected for expansion, because the evaluation measures distance to the goal. Best-first search can be implemented via a priority queue, a data structure that will maintain the fringe in ascending order of f -values.

**Tree Search**

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

Figure 1.24 An informal description of the general tree-search algorithm

There is a whole family of BEST-FIRST-SEARCH algorithms with different evaluation functions. A key component of these algorithms is a heuristic function, denoted *h (n):* ***h (n) = estimated cost of the cheapest path from node n to a goal node.***
For example, in Romania, one might estimate the cost of the cheapest path from Arad to Bucharest via the straight-line distance from Arad to Bucharest. Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.
Let see the two ways to use heuristic information to guide search.

- ☺ Greedy best-first search
- ☺ A* search

### 3.4.1. Greedy Best-First Search

**Greedy best-first search** tries to expand the node that is closest to the goal, on the: grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function: f *(n) = h (n).*
Let us see how this works for route-finding problems in Romania (see Figure 3.9), using the straight line distance heuristic, which we will call $h_{SLD}$. If the goal is Bucharest, we will need to know the straight-line distances to Bucharest, which are given. Notice that the values of $h_{SLD}$ cannot be computed from the problem description itself. Moreover, it takes a certain amount of

experience to know that $h_{SLD}$ is correlated with actual road distances and is, therefore, a useful heuristic.

Greedy best-first search resembles depth-first search in the way it prefers to follow a single path all the way to the goal, but will back up when it hits a dead end. It suffers from the same defects as depth-first search-it is not optimal, and it is incomplete (because it can start down an infinite path and never return to try other possibilities). The worst-case time and space complexity is O ($b^m$), where m is the maximum depth of the search space.

**With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic.**

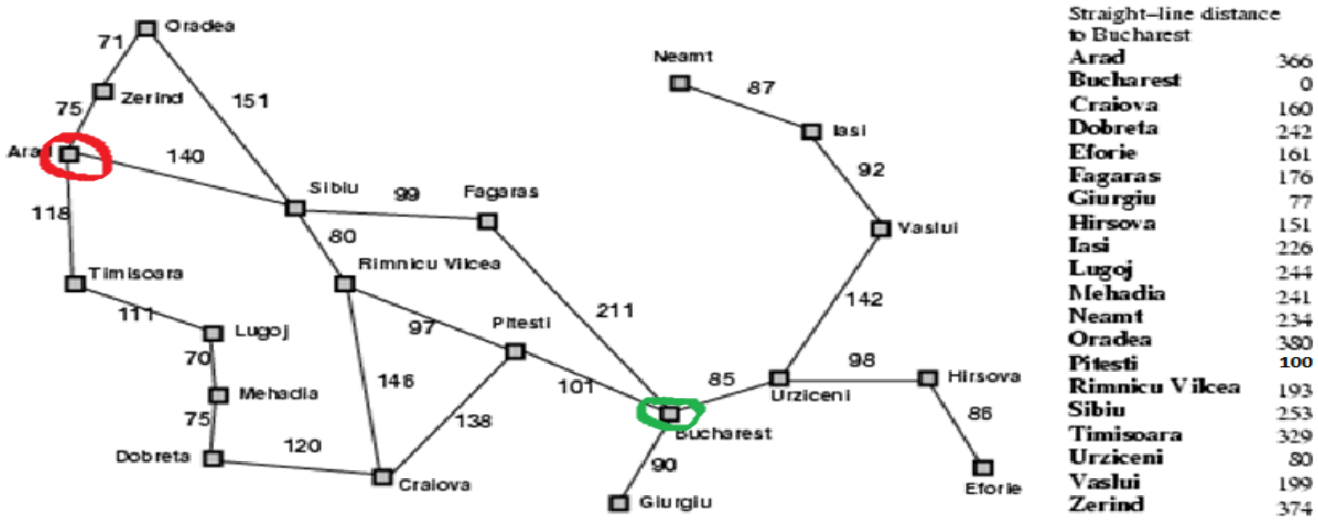



**Figure 3.9** Romania road map and $h_{SLD}$ *values* of each cities

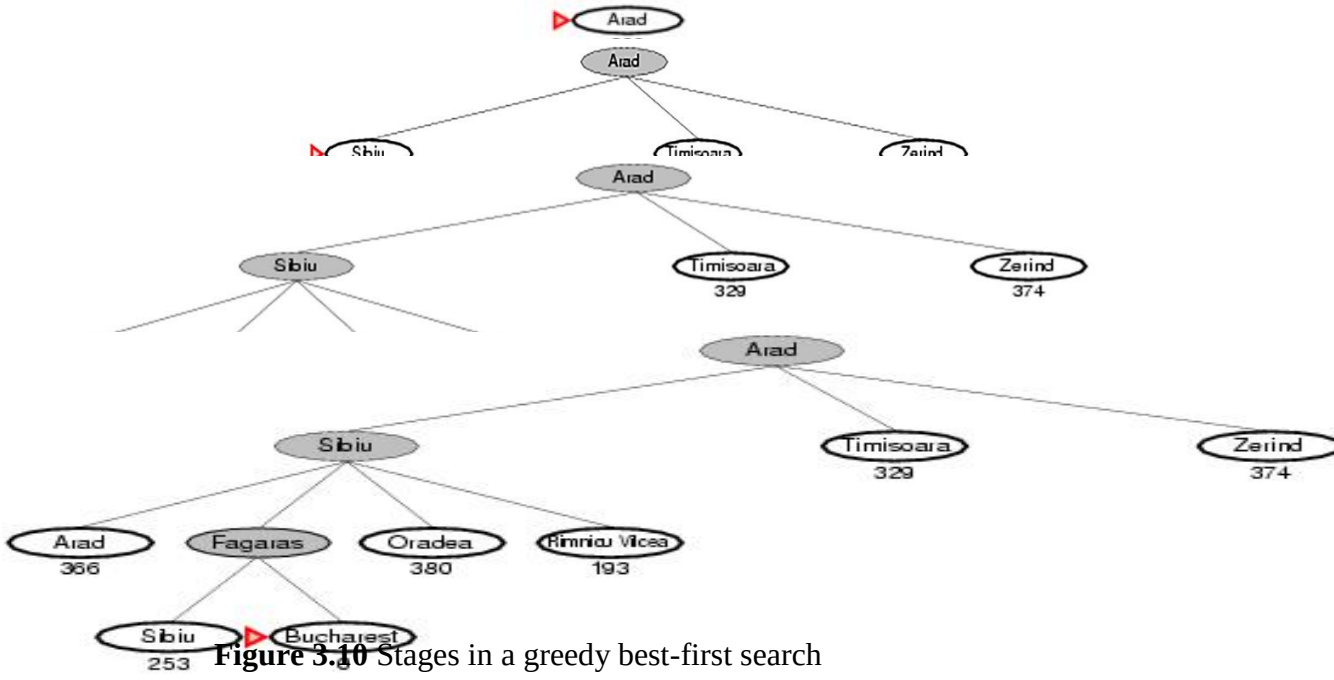



**Figure 3.10** Stages in a greedy best-first search

***Properties of GBFS***

- **Complete?** No – can get stuck in loops.
- **Time?** $O(b^m)$, but a good heuristic can give dramatic improvement
- **Space?** $O(b^m)$ - keeps all nodes in memory
- **Optimal?** No e.g. Arad→Sibiu→Rimnicu Virea→Pitesti→Bucharest is shorter!

### 3.4.2. A* SEARCH

The most widely-known form of best-first search is called **A*** search (pronounced "A-star search"). It evaluates nodes by combining *g (n),* the cost to reach the node, and *h (n.),* the cost to get from the node to the goal:

***f (n) = g (n) + h (n)*** Since *g(n)* gives the path cost from the start node to node *n*, and *h(n)* is the estimated cost of the cheapest path from n to the goal, we have ***f (n) = estimated cost of the cheapest solution through n***

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of *g(n) + h(n).* It turns out that this strategy is more than just reasonable: provided that the heuristic function *h(n)* satisfies certain conditions, *A** search is both complete and optimal.

The optimality of A* is straightforward to analyze if it is used with Tree-Search. Admissible Heuristic In this case, A* is optimal if *h(n)* is an admissible heuristic-that is, provided that *h(n)* ***never overestimates*** the cost to reach the goal. Admissible heuristics are by nature optimistic, because they think the cost of solving the problem is less than it actually is. Since *g(n)* is the exact cost to reach *n*, we have as immediate consequence that *f (n)* never overestimates the true cost of a solution through n.

Figure 3.11, shows the progress of an A* tree search for Bucharest. The values of g are computed from the step costs in Figure 3.9, and the values of $h_{SLD}$ are also given in Figure 3.9. Notice in particular that Bucharest first appears on the fringe at step (e), but it is not selected for expansion because its *f* -cost (450) is higher than that of Pitesti (417). Another way to say this is that there ***might*** be a solution through Pitesti whose cost is as low as 417, so the algorithm will not settle for a solution that costs 450. From this example, we can extract a general proof that A* ***using*** Tree-Search ***is optimal*** *if h(n)* ***is admissible.***

**A* search** is complete, optimal, and optimally efficient among all such algorithms is rather satisfying. Unfortunately, it does not mean that A* is the answer to all our searching needs. The catch is that, for most problems, the number of nodes within the goal contour search space is still exponential in the length of the solution.

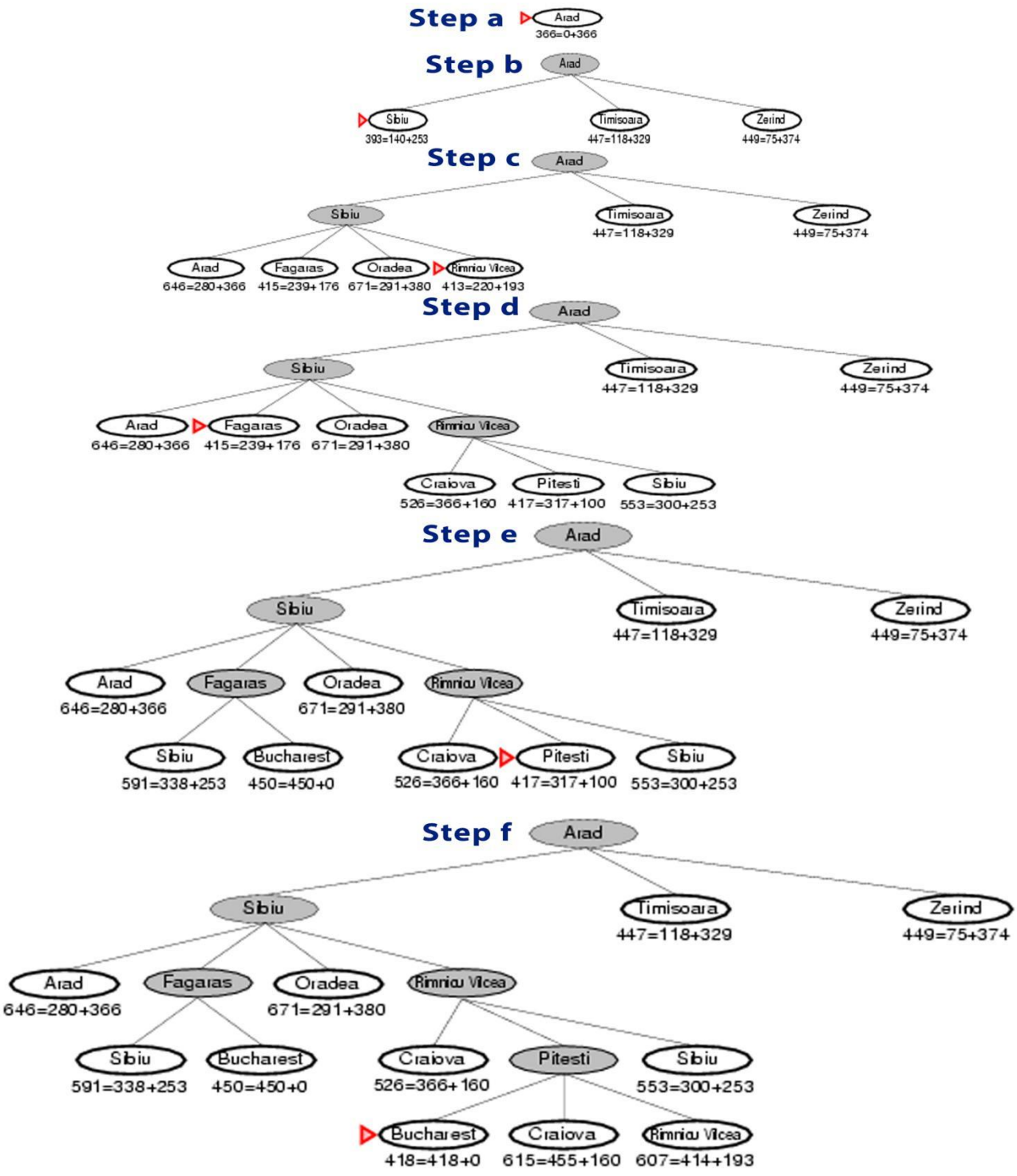



Figure 3.11 Stages in an A* search for Bucharest.

***Properties of A****

- ☆ **Complete?** Yes (unless there are infinitely many nodes)
- ☆ **Time/Space?** The better the heuristic, the better the time
  - Best case h is perfect, O(d)
- ☆ **Optimal?** Yes
- ☆ ***Optimally Efficient*:** Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)
- ☆ ***Space***
  - Keeps all nodes in memory and save in case of repetition
  - This is $O(b^d)$ or worse
  - A* usually runs out of space before it runs out of time

# Chapter 4: Knowledge Representation and Reasoning

## 4.1. Introduction

Knowledge includes facts about the real world entities and the relationship between them

- Characteristics of Knowledge:
  - It is large in nature and requires proper structuring.
  - It may be incomplete and vague.
  - It may keep on changing (dynamic).

**Knowledge: What and Why?**

- We are living in complex environment where there are:
  - Many actors, presume, strong competitors, and high turnover
- It enables to:
  - Automate reasoning , Discover new facts, Deduce new facts that follow from the KB, and Answer users queries
  - Make quality decisions - select courses of actions, etc.
  - Hence, there is a need to represent knowledge to ease the development of an intelligent system.

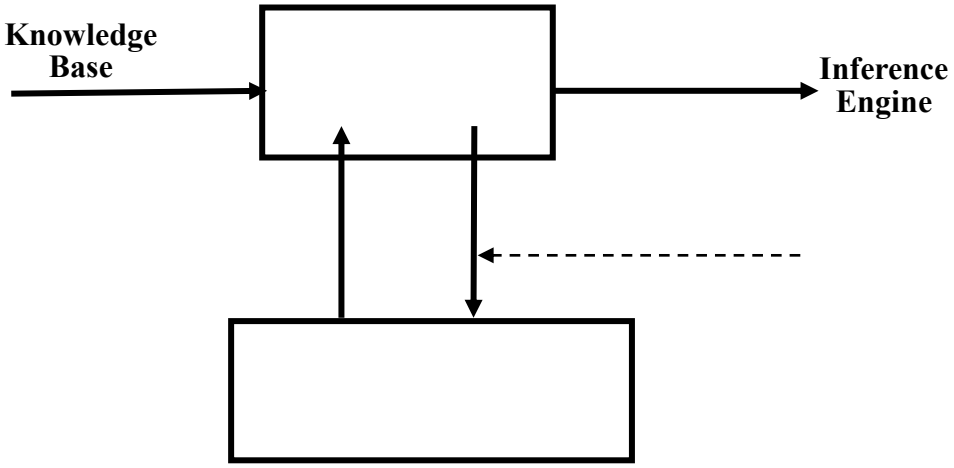
**Motivation of Knowledge-Based (KB)**

- Knowledge-Based Systems (KBSs) are useless without the ability to represent knowledge. Agents can be seen as knowing about their world, and reasoning about their possible courses of action. One can also design an **autonomous agent** that learns from experience and construct knowledge without human interventions
- Intelligent system require:
  - **Knowledge** (current state) of the world
    - formally represented using knowledge representation language
  - **Inference mechanisms**: Inference is deriving new sentences from old.
  - **Goal**: what it wants to achieve
  - **Effect of its action**: what its own action do in various circumstances

### 4.2. Components of a Knowledge-based Agent (KBA)

The agent is composed of:

- ➔ Knowledge base
  - It operates by storing a set of sentences (facts and relationship between facts) about the world in its KB.
- ➔ Inference mechanism :
  - With the help of inference mechanism deduce new sentences and use them to decide what action to take
- ➔ Learning mechanism:
  - Adapt to changes in the environment by updating the relevant knowledge




**Knowledge Base (KB)**

- Contains set of **facts** about the domain expressed in a suitable representation language
  - Each individual representation are called sentences
  - Sentences are expressed in a (formal) **knowledge representation (KR) language**
- A KBA is designed such that there is a way to:-
  - **TELL** it (i.e. to add new sentences to the KB) and **ASK** it (i.e. to query the KBA)
  - When one ASKs a question, the answer should follow from what has been TELLed to the KB previously
  - **Inference mechanism** determines what follows from what has been TELLed to the KB

**Inference Engine (IE)**

- The Inference engine derives new sentences from the input and KB
- The agent operates as follows:
  1. It receives percepts from environment
  2. It computes what action it should perform (by IE and KB)
  3. It performs the chosen action.

**KB can be viewed at different levels**

- **Knowledge Level.**

- The most abstract level -- describe agent by saying what it knows.
- Example: An automated taxi might be said to know that the ***Abay*** Bridge links ***Kebele10*** and ***Kebele11***

- **Logical Level.**
  - The level at which the knowledge is encoded into sentences.
  - For e.g. The taxi might be described as having the logical sentences.

  Links(AB,K10,K11) in our knowledge base.
- **Implementation Level.**
  - The physical representation of the sentences in the logical level - how will it be represented in the computer?

### 4.3. Knowledge Representation, Reasoning and Logic

- **Knowledge Representation (KR)**: express knowledge clearly in a computer-tractable way such that the agent can reason out.
- **Reasoning**: is the process of constructing new sentences from existing facts in the KB.
  - Proper reasoning ensures that the new configuration represent facts that actually follow from the facts in the KB.

**Parts of KR language:**

Syntax of a language: describes the possible configuration to form sentences.

E.g.: if x & y denote numbers, then x > y is a sentence about numbers

Semantics: determines the facts in the world to which the sentences refer.

E.g.: Y > X is false when X is greater than Y

Proof theory (inference rules and proof procedures)

**Why KR is important?**

- It enables to:
  - Automate reasoning
  - Discover new facts.
  - Deduce new facts that follow from the KB
  - Answer users queries
  - Make decisions - select courses of actions
  - etc.

**How is Knowledge represented?**

- Knowledge is basically represented as "**symbol structures**" (essentially, complex data structures) representing bits of knowledge (objects, concepts, facts, rules, strategies...)

Example: "**red**" represents color red.

"**mycar**" represents my car.

**red(mycar)** represents the fact that my car is red.

- Intelligent behavior can be achieved through manipulation of symbol structures
- Build KB following **declarative approach**

- Rather than procedural language (like C++/Java data structures), use declarative languages (like prolog).

**Logic as KR**

- A Logic is a **formal language** in which knowledge can be represented such that conclusions can be drawn.
  - It is a declarative language to assert sentences and deduce from sentences.
- **Components of a formal logic** include: syntax, semantics, reasoning and inference mechanism.
  - **Syntax:** what expressions/structures are allowed in the language. Describes how to make sentences

  E.g. red(mycar) is ok, but mycar(grey or green) is not.
  - **Semantics:** express what sentences mean, in terms of a mapping to real world.
    - E.g. red(mycar) means that my car is red.
  - **Proof Theory:** how we can draw new conclusions from existing statements in the logic.

**Why *formal* languages** (Logic) ?

- An obvious way of expressing or representing facts and thoughts is by writing them in a natural language such as English. However,
  - The meaning of a sentence depends on the sentence itself and on the context on which the sentence was spoken
  - Natural languages exhibit **ambiguity.** E.g. small dogs and cats.
- Ambiguity makes reasoning difficult and incomplete.
  - Hence we need formal languages to express facts and concepts in an unambiguous and well-defined way.

Natural languages:

- Natural languages are certainly expressive,
- they help us for communication, not for representation.
- A good way for a speaker to get a listener to come to know something
- Natural language also suffers from ambiguity.
  - Small dogs and cats
  - Dangerous men and locations

**Logics**

- Logic is the science or art of reasoning
  - Concerned with what is true and how we know whether something is true.
- Two kinds of Logic
  - Propositional logic or Boolean Logic
  - First-order Logic

### 4.4. Propositional logic

A simple language useful for showing key ideas and definitions

**Syntax:** PL allows facts about the world to be represented as sentences formed from:

i. **Logical constants**: True, False
ii. **Proposition symbols (*P, Q, R, …)*** are used to represent facts about the world: e.g.: P = "It is hot“, Q = "It is humid“, R = "It is raining“

iii. **Logical connectives**: not **(¬),** and **(∧),** or **(∨),** implies **(→),** is equivalent, if and only if (↔).

**1.** Precedence order from highest to lowest is:

**¬, ∧,∨,→, ↔**

**e.g.** The sentence **¬P v Q ∧ *R* → *S*** *is equivalent to*

***[(¬P) v (Q ∧ R)] → S***

**2. Parenthesis** ( ): Used for grouping sentences and to specify order of precedence

### 4.4.1. Propositional logic (PL) sentences

- A sentence is made by linking prepositional symbols together using logical connectives.
  - There are atomic and complex sentences.
  - Atomic sentences consist of propositional symbol (e.g. P, Q, TRUE, FALSE)
  - Complex sentences are combined by using **connectives** or **parenthesis:**
  - while S and T are atomic sentences, S ∨ T, (S ∨ T), (S ∧ T), (S → T), and (S ↔ T) are complex sentences.

**Examples:** Given the following sentences about the **"**weather problem" convert them into **PL sentences:**

- "It is humid.": Q
- "If it is humid, then it is hot" : Q → P
- "If it is hot and humid, then it is raining": (P ∧ Q) → R

### 4.4.2. A BNF (Backus-Naur form) grammar of sentences in propositional logic:

S := <Sentence> ;
<Sentence> := <AtomicSentence> | <ComplexSentence> ;
<AtomicSentence> := "TRUE" | "FALSE" | "P" | "Q" | "S" ;
<ComplexSentence> := "(" <Sentence> ")" | <Sentence> <Connective> <Sentence> |"NOT" <Sentence> ;
<Connective> := "NOT" | "AND" | "OR" | "IMPLIES" | "EQUIVALENT" ;

**Exercise: 1**

Convert the following English sentences to Propositional logic. Let:

A = Sara is absolutely beautiful
R = her personality is perfect
P = dawit will in love with Sara

- Sara is **not** absolutely beautiful: **¬A**
- Sara is absolutely beautiful **and** her personality is perfect: **A ∧ R**
- either Sara is absolutely beautiful **or** her personality is perfect : **A V R**
- **if** Sara is absolutely beautiful, **then** her personality is **not** perfect : **A → ¬ R**
- Sara is absolutely beautiful**if and only if** her personality is perfect: **A ↔ R**
- if Sara is absolutely beautiful, then if her personality is not perfect, Dawit will not in love with Sara : **A → (¬R → ¬P )**

**Exercise: 2**

**Examples:** Convert from **English to Propositional sentence**

Let **A** = Lectures are active, **R** = Text is readable, **P** = Kebede will pass the exam, then represent the following:

- the lectures are ***not*** active: **$\neg A$**
- the lectures are active **and** the text is readable: **$A \wedge R$**
- either the lectures are active **or** the text is readable: **$A \vee R$**
- **if** the lectures are active, **then** the text is not readable: **$A \rightarrow \neg R$**
- the lectures are active **if and only if** the text is readable: **$A \leftrightarrow R$**
- if the lectures are active, then if the text is not readable, Kebede will not pass the exam: **$A \rightarrow (\neg R \rightarrow \neg P)$**

### 4.4.3. Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas
  The proposition symbols $P_1$, $P_2$ etc are sentences
  - If S is a sentence, $\neg S$ is a sentence (negation)
  - If $S_1$ and $S_2$ are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

**Terminology**

- **Valid sentence:** A sentence **is valid sentence or tautology** if and only if it is True under all possible interpretations in all possible worlds.
  **Example:** "It's raining or it's not raining." ($R \vee \neg R$).
- **Satisfiable**: A sentence is satisfiable if and only if there **is some interpretations** in some world for which the sentence is True.
  **Example**: "It is raining or it is humid". R v Q
- **Unsatisfiable:** A sentence is unsatisfiable (inconsistent sentence or self- contradiction) if and only if it is not satisfiable, i.e. a sentence that is False under all interpretations.
  **The world is never like what it describes.**
  **Example**: "It's raining and it's not raining." $R \wedge \neg R$

**Logical equivalence**

| | |
|---|---|
| $p \vee q \equiv q \vee p$ | Commutativity of disjunction |
| $p \wedge q \equiv q \wedge p$ | Commutativity of conjunction |
| $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ | Associativity of conjunction |
| $(p \vee q) \vee r \equiv p \vee (q \vee r)$ | Associativity of disjunction |
| $(\neg (\neg p) \equiv p$ | Double Negation elimination |

| | |
|---|---|
| $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$ | Contra position |
| $p \Rightarrow q \equiv \neg p \vee q$ | implication elimination |
| $\neg (p \vee q) \equiv (\neg p \wedge \neg q)$ | De-Morgan |
| $\neg (p \wedge q) \equiv (\neg p \vee \neg q)$ | De-Morgan |
| $(p \Leftrightarrow q) \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$ | Bi-conditional elimination |
| $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ | Distributive of $\wedge$ over $\vee$ |
| $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ | Distributive of $\vee$ over $\wedge$ |

**Inference Rules**

| RULE | PREMISE | CONCLUSION |
|---|---|---|
| **Modus Ponens** | $\mathbf{A, A \rightarrow B}$ | $\mathbf{B}$ |
| **Modus Tolens** | $\mathbf{\neg B, A \rightarrow B}$ | $\mathbf{\neg A}$ |
| **And Elimination** | $\mathbf{A \wedge B}$ | $\mathbf{A}$ |
| **And Introduction** | **A, B** | $\mathbf{A \wedge B}$ |
| **Or Introduction** | **A** | $\mathbf{A_1 \vee A_2 \vee \ldots \vee A_n}$ |
| **Double Negation Elimination** | $\mathbf{\neg \neg A}$ | **A** |
| **Unit Resolution** | $\mathbf{A \vee B, \neg B}$ | **A** |
| **Resolution** | $\mathbf{A \vee B, \neg B \vee C}$ | $\mathbf{A \vee C}$ |
| **Hypothetical Syllogism** | $\boldsymbol{P \rightarrow Q, Q \rightarrow R}$ | $\boldsymbol{P \rightarrow R}$ |

**Validity and satisfiability**

A sentence is valid if it is true in all models,

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

A sentence is satisfiable if it is true in some model

e.g., A∨ B, C

A sentence is unsatisfiable if it is true in no models

e.g., A∧¬A

**Example**

- Example 1: Given the following facts and rules that relates facts; What can we say about the weather condition?
- It is humid:
- If it is humid, then it is hot :
- If it is hot and humid, then it is raining:

  - Q Premise
  - $Q \rightarrow P$ Premise
  - $(P \wedge Q) \rightarrow R$ Premise
  - P using Modes Ponens (1 & 2)
  - $P \wedge Q$ using AND introduction (1 & 4)
  - R using Modes Ponens (3 & 5)

**Propositional logic is a weak language**

- Example: Prove that "my dog Fido is Nice, given that "all dogs are Nice."
  - This requires to get at the structure and meanings of statements (where FOL is useful).

**First Order Logic**

- First-Order Logic (FOL) is expressive enough to concisely represent any kind of situation that are expressed in natural language.
  - FOL represents objects and relations between objects, variables, and quantifiers in addition to propositions

*Every elephant is gray:*

$\forall$ x (elephant(x) $\rightarrow$ gray(x))

*There is a white alligator:*

$\exists$ x (alligator(X) ^ white(X))

**Syntax of FOL**

- **Constants symbol**
  - names (like Jonas, Kebede, …), numbers (like 1, 2, … n), ...
- **Predicates**:
  - Predicates used to relate one object with another. E.g. brother, >,...
- **Functions**: Returns value (mother-of,...)
- **Variables**: x, y, a, b,...
  - Important to increase generalization capability of KB
- **Connectives**:
  - retains connectives used in PL ($\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$)
- **Quantifiers**:
  - Quantifiers specify whether all or some objects satisfy properties or relations between objects
  - Two standard quantifiers: Universal (" for all, for every) and Existential ($ there exists, some)

1. **Universal quantification**

- Universal Quantifiers: makes statements about every object

  ∀<*variables*> <*sentence*>

  - Everyone at WKU is smart:

    $\forall_x$ At(x,WKU) ⇒ Smart(x)
  - All cats are mammals:

    $\forall_x$ cat(x) ⇒ mammal(x)
    - $\forall_x$ sentence *P* is true iff *P* is true with *x* being each possible object in the given universe
  - The above statement is equivalent to the conjunction

  At(Jonas, WKU) ⇒ Smart(Jonas)

  At(Rawad, WKU) ⇒ Smart(Rawad)

  ...
- A common mistake to avoid
  - Typically, ⇒ is the main connective with ∀
  - Common mistake: the use of ∧ as the main connective with ∀:

  $\forall_x$ At(x,WKU) ∧ Smart(x) is true if "Everyone is at Wku & everyone is smart"

2. **<u>Existential quantification</u>**

- Makes statements about some objects in the universe

  ∃<*variables*> <*sentence*>
  - Someone at WKU is smart:

    $\exists_x$ At(x,WKU) ∧ Smart(x)
  - Spot has a sister who is a cat:

    $\exists_x$ sister(spot,x) ∧ cat(x)
    - $\exists_x$ sentence *P* is true iff *P* is true with *x* being some possible objects
  - The above statement is equivalent to the disjunction

  At(Jonas, WKU) ∧ Smart(Jonas)

  At(Alemu, WKU) ∧ Smart(Alemu)

  …..
- Common mistake to avoid
  - Typically, ∧ is the main connective with ∃
  - Common mistake: using ⇒ as the main connective with ∃:

  $\exists_x$ At(x,WKU) ⇒ Smart(x) is true if there is anyone who is not at WKU

**3. Nested quantifiers**

- $\forall_{x,y}$ parent(x,y) ⇒ child(y,x)
  - for all x and y, if x is the parent of y then y is the child of x.
- $\exists_x \forall_y$ Loves(x,y)
  - There is a person who loves everyone in the given world
- $\forall_y \exists_x$ Loves(x,y)
  - Everyone in the given universe is loved by at least one person

**Properties of quantifiers**

- $\forall_x \forall_y$ is the same as $\forall_y \forall_x$
- $\exists_x \exists_y$ is the same as $\exists_y \exists_x$

- $\exists_x \forall_y$ is not the same as $\forall_y \exists_x$

➔ Quantifier duality: each can be expressed using the other, using negation ($\neg$)

$\forall_x$ Likes(x,icecream) $\neg\exists_x \neg$Likes(x,icecream)

- Everyone likes ice cream means that there is nobody who dislikes ice cream

$\exists_x$ Likes(x,cake) $\neg\forall_x \neg$Likes(x,cake)

- There is someone who likes cake means that there is no one who dislikes cake

**Sentence structure**

In FOL the basic unit is a predicate (argument/terms) structure called sentence to represent facts.

- A predicate is the one that says something about the subject. E.g., There is a red book
  - Subject: color of the book, represented as: $\exists_x$ book(x)→red(x)
- Predicate also refers to a particular relation between objects
  - Example: likes(X, richard)
    friends(motherof(jonas), motherof(semu))
- A predicate statement takes the value true or false

**Sentences**

**Atomic sentences:** formed from a predicate symbol followed by a parenthesized list of terms

Atomic sentence = *predicate* (*term*$_1$,...,*term*$_n$)

**Example:** *Brother(John, Richard)*

- *Atomic sentences can have arguments that are complex terms (e.g. t*erm = *function* (*term*$_1$,...,*term*$_n$) *)*

**Example:** *married (fatherof(Richard),motherof(John))*

**Complex sentences: c**omplex sentences are made by combining atomic sentences using connectives:

$\neg S$, $S_1 \wedge S_2$, $S_1 \vee S_2$, $S_1 \Rightarrow S_2$, $S_1 \Leftrightarrow S_2$,

Ex. Likes (john, mary) $\wedge$ tall (mary)

Tall (john) $\Rightarrow$ handsome (john)

*Sibling (John, Richard)* $\Rightarrow$ *Sibling (Richard, John)*

- Sentences can also be formed using quantifiers to indicate how to treat variables:
  - Universal quantifier: $\forall$x lovely(x) - Everything is lovely.
  - Existential quantifier: $\exists$x lovely(x) - Something is lovely.

**Exercise - 1**

- Can have several quantifiers, e.g.,

$\forall_x \exists_y$ loves(x, y)

$\forall_x$ handsome(x) $\Rightarrow \exists_y$ loves(y, x)

Represent the following in FOL:

- Everything in the garden is lovely → $\forall$x in(x, garden) $\Rightarrow$ lovely(x)
- Everyone likes ice cream → $\forall$x likes(x, icecream)
- Peter has some friends →$\exists$y friends(y, Peter)
- John plays the piano or the violin → plays(john, piano) v plays(john, violin)
- Some people like snakes → $\exists$x(person(x) $\Lambda$ likes(x, snakes))
- Winston did not write Hamlet → $\neg$write( Winston, hamlet)
- Nobody wrote Hamlet → $\neg\exists$x write(x, hamlet)

**Exercise - 2**

Bob is a buffalo. Pat is a pig. Buffaloes outrun pigs

Conclude: Bob outruns Pat
1. Buffalo (Bob)
2. Pig (Pat)
3. ∀x, y Buffalo(x) ^ Pig(y)→ outrun(x, y)
4. Buffalo (Bob) ^ Pig (Pat) → and Introduction (1, 2)
5. Buffalo (Bob) ^ Pig (Pat) → outrun (Bob, Pat)
6. Outrun (Bob, Pat)

**Inference Mechanisms**

- Inference is a means of interpretation of knowledge in the KB to reason and give advise to users query.
- There are two inference strategies to control and organize the steps taken to solve problems:
  - **Forward chaining**: also called data-driven chaining
    - It starts with facts and rules in the KB and try to draw conclusions from the data
  - **Backward chaining**: also called goal-driven chaining
    - It starts with possible solutions/goals and tries to gather information that verifies the solution

# Chapter 5: Machine Learning Basics

## 5.1. Knowledge in Learning

Knowledge is like glue that sticks information as well as learning together. Additionally, it is a way of describing how machines can represent learning in artificial intelligence. Knowledge representation is not just storing data in some database. Still, it also enables an intelligent device to learn from that knowledge and experience to behave intelligently like a human.

## 5.2. Learning Probabilistic Models

Probabilistic Models are one of the most important segments in Machine Learning, which is based on the application of statistical codes to data analysis. Unobserved variables are seen as stochastic in probabilistic models, and interdependence between variables is recorded in a joint probability distribution. It provides a foundation for embracing learning for what it is. The probabilistic framework outlines the approach for representing and deploying model reservations. In scientific data analysis, predictions play a dominating role. Their contribution is also critical in machine learning, cognitive computing, automation, and artificial intelligence. **Probabilistic** modeling is a statistical approach that uses the effect of random occurrences or actions to forecast the possibility of future results. It is a quantitative modeling method that projects several possible outcomes that might even go beyond what has happened recently.

## 5.3. Supervised learning

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labelled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.
Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

### 5.3.1. Linear classification models

A linear classifier is a **model that makes a decision to categories a set of data points to a discrete class based on a linear combination of its explanatory variables**. As an example, combining details about a dog such as weight, height, color and other features would be used by a model to decide its species. The effectiveness of these models lie in their ability to find this mathematical combination of features that groups data points together when they have the same class and separate them when they have different classes, providing us with clear boundaries for how to classify.
If each instance belongs to one and only one class, then our input data can be divided into decision regions separated by decision boundaries.
Although each instance represents a unique attribute, we can consider each instance as being a point in a d-dimensional space, where each individual feature is a dimension on its own and d therefore represents the number of features in our dataset. An instance with 4 features will therefore have its own unique point in a 4 dimensional space for example.
We can create linear decision boundaries to split our classes and use these to then predict the class of new data points. Linear decision boundaries are linear functions of x. They are D-1 dimensional hyperplanes in a D dimensional input space. For example, a set of 3 dimensional features will have 2D decision boundaries (planes) and a set of 2 dimensional features will have 1D decision boundaries (lines).

Linear Decision Surfaces

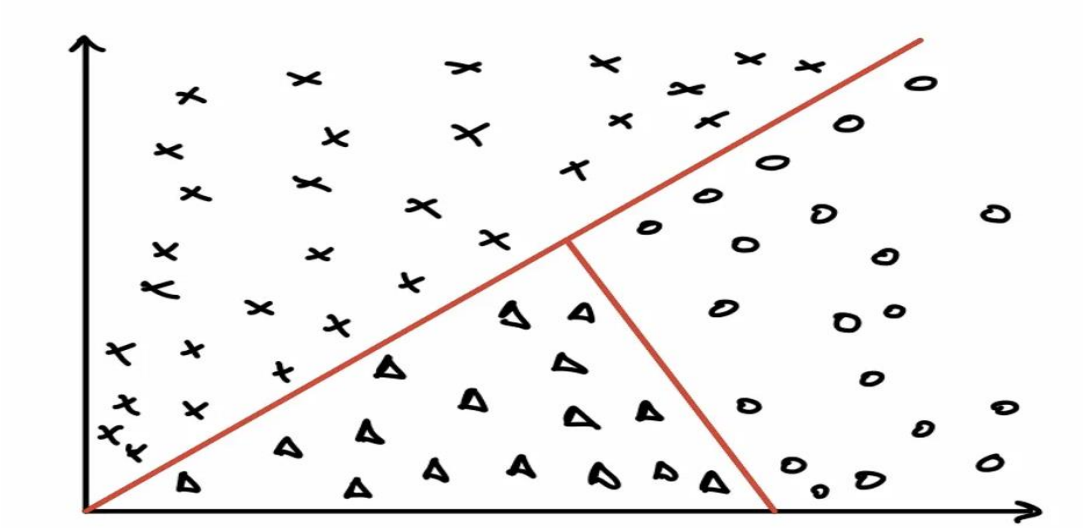


*Figure 5.1: liner decision surface*

Linear separability is how we define the decision surfaces that our model creates on our data. If our data is linearly separable, then individual classes can be separated by linear decision boundaries, like in the example we have above for a 2D input space.

Linear Separability

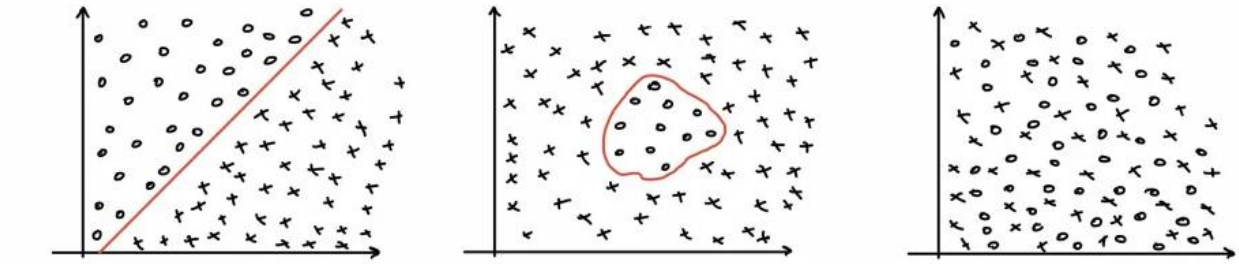
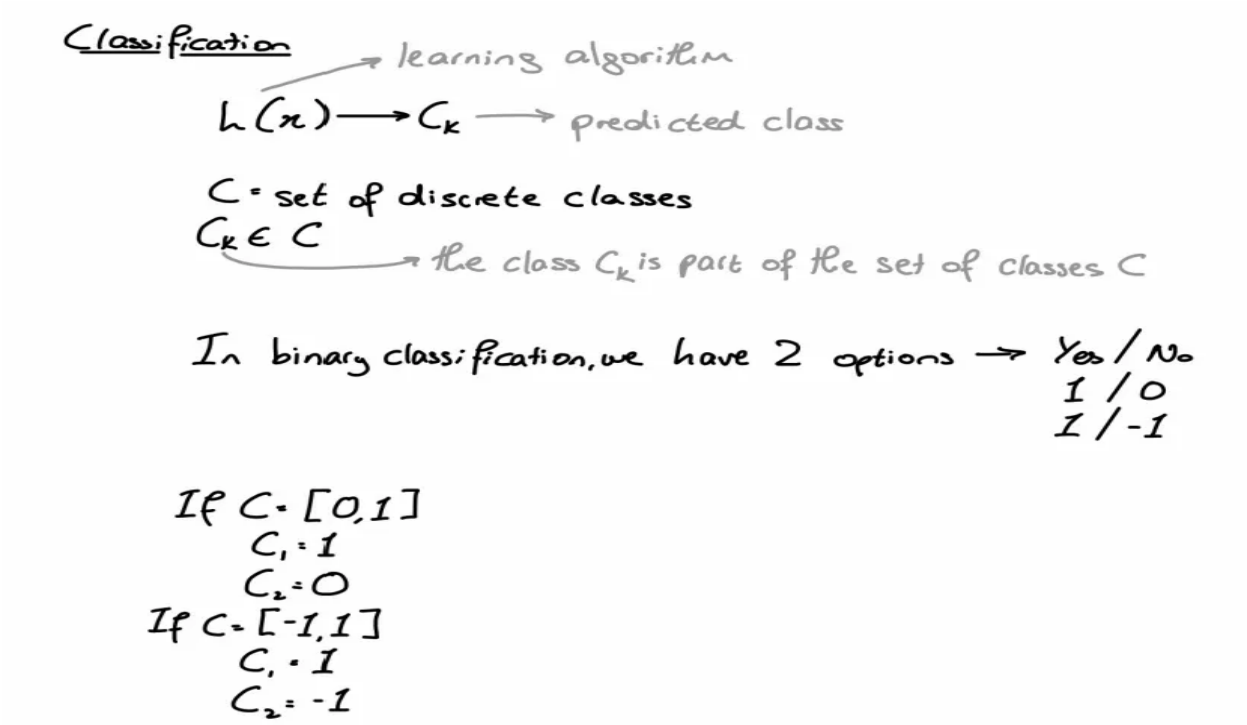


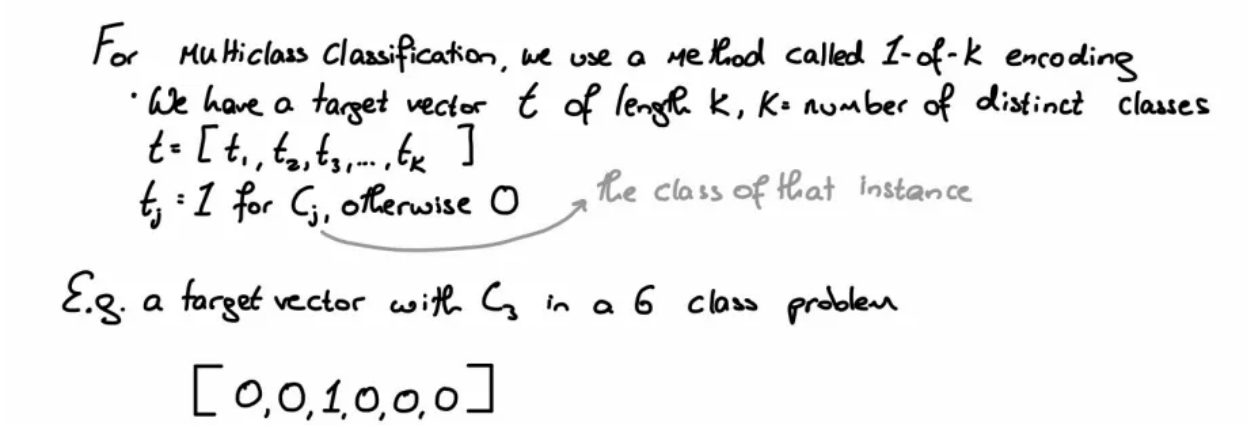
Figure 5.2: Linear Separability

If decision surfaces are not linearly separable, the decision surfaces may take the form of more complex functions. The third situation is where our data is not separable at all, through either linear or nonlinear decision surfaces. If the input data we have is not separable at all, this will generally lead to poor performance from simple machine learning methods, because this concept is what they need to generate accurate predictions. Models need to be able to find a feature or combination of features in your dataset that, when combined in a specific way, allow the model to understand how the classes can be separated based on their attributes.
Mathematically, we can define our classes as follows:

## Notation




## Multiclass Classification




Linear Classification is initially an extension of our Linear Regression model. We are aiming to find a set of coefficients for our features that when summed together, will provide us with an accurate measure of our target variable. It is however at this point that a standard linear regression might break down for the purposes of classification. The result of the output is unbounded and therefore makes it very difficult for us to establish any thresholds that we can use to differentiate classes. Linear Classification solves this by introducing the concept of a nonlinear activation function, which we will pass our regression output into. This function will constrict

any value to a certain range, generally (0, 1) or (-1, 1). This makes it far easier for us to generate decision boundaries to determine classes

5.3.2. Probabilistic models

A probabilistic method or model is based on the theory of probability or the fact that randomness plays a role in predicting future events. The opposite is deterministic, which is the opposite of random — it tells us something can be predicted exactly, without the added complication of randomness. Probabilistic Models are one of the most important segments in Machine Learning, which is based on the application of statistical codes to data analysis. This dates back to one of the first approaches of machine learning and continues to be widely used today. Unobserved variables are seen as stochastic in probabilistic models, and interdependence between variables is recorded in a joint probability distribution. It provides a foundation for embracing learning for what it is. The probabilistic framework outlines the approach for representing and deploying model reservations. In scientific data analysis, predictions play a dominating role. Their contribution is also critical in machine learning, cognitive computing, automation, and artificial intelligence.

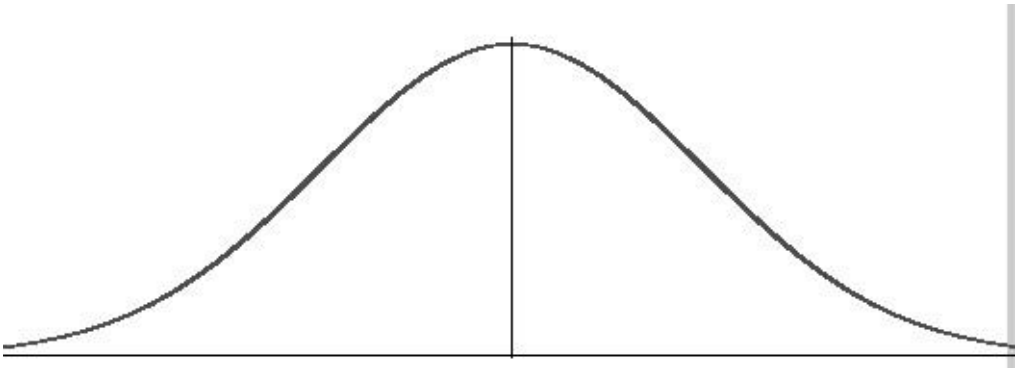


*Figure 5.3: A normal distribution curve, sometimes called a bell curve, is one of the building blocks of a probabilistic model.*

## 5.4. Unsupervised machine learning

Unsupervised machine learning and supervised machine learning are frequently discussed together. Unlike supervised learning, unsupervised learning uses unlabeled data. From that data, it discovers patterns that help solve for clustering or association problems. This is particularly useful when subject matter experts are unsure of common properties within a data set. Common clustering algorithms are hierarchical, k-means, and Gaussian mixture models.
As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and

insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:
Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.
Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**.
**Example:** Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

### 5.5. Reinforcement learning

Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty. In Reinforcement Learning, the agent learns automatically using feedbacks without any labelled data, unlike learning. Since there is no labelled data, so the agent is bound to learn by its experience only. RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.

The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards. The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that **"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."** How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.

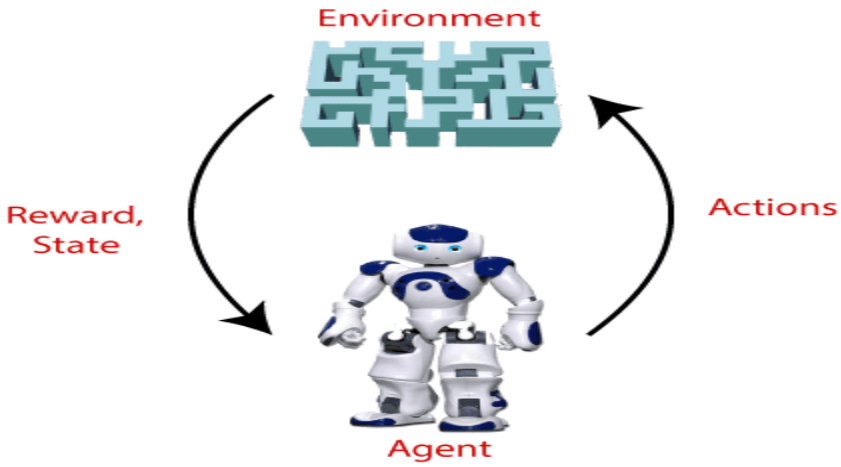



*Figure 5.4: re-enforcement learning*

### 5.5.1. Terms used in Reinforcement Learning

- **Agent ():** An entity that can perceive/explore the environment and act upon it.
- **Environment ():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action ():** Actions are the moves taken by an agent within the environment.
- **State ():** State is a situation returned by the environment after each action taken by the agent.
- **Reward ():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy ():** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value ():** It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- **Q-value ():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

### 5.5.2. Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- ✓ **Positive Reinforcement**
- ✓ **Negative Reinforcement**

**Positive Reinforcement:**

The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

**Negative Reinforcement:**

The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

### 5.6. Deep Learning

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now.

Deep learning is based on the branch of machine learning, which is a subset of artificial intelligence. Since neural networks imitate the human brain and so deep learning will do. In deep learning, nothing is programmed explicitly. Basically, it is a machine teaching class that makes use of numerous non-linear processing units so as to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.

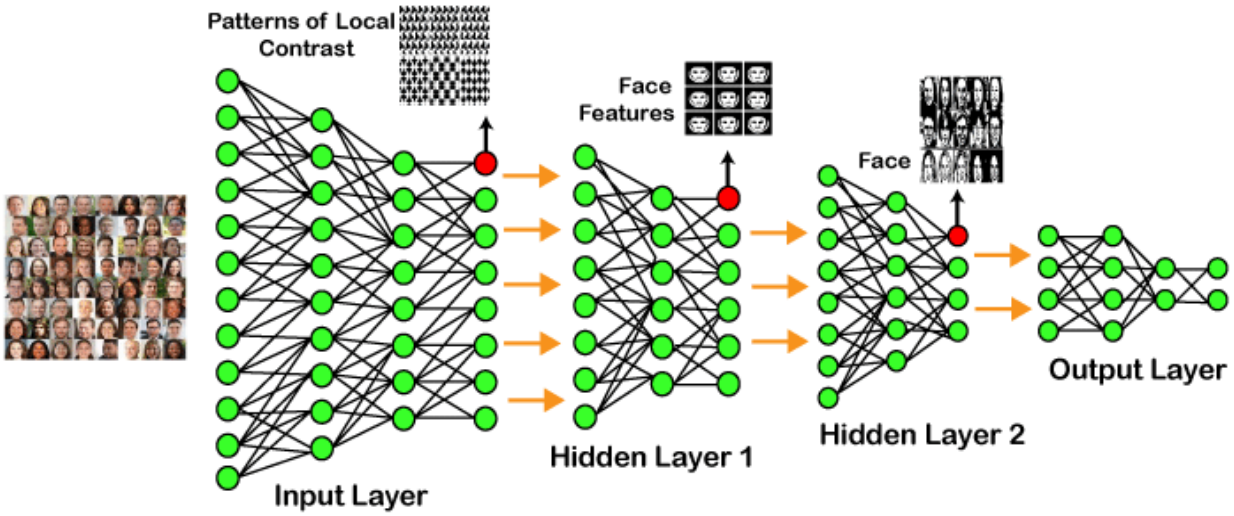


*Figure 5.5: Deep learning Examples*

### 5.6.1 Neural networks and back-propagation

The concept of the backpropagation neural network was introduced in the 1960s and later it was published by David Rumelhart, Ronald Williams, and Geoffrey Hinton in the famous 1986 paper. They explained various neural networks and concluded that network training is done through backpropagation.
Backpropagation is widely used in neural network training and calculates the loss function with respect to the weights of the network. It functions with a multi-layer neural network and observes the internal representations of input-output mapping.
A supervised neural network, at the highest and simplest representation, can be presented as a black box with 2 methods learn and predict as following:

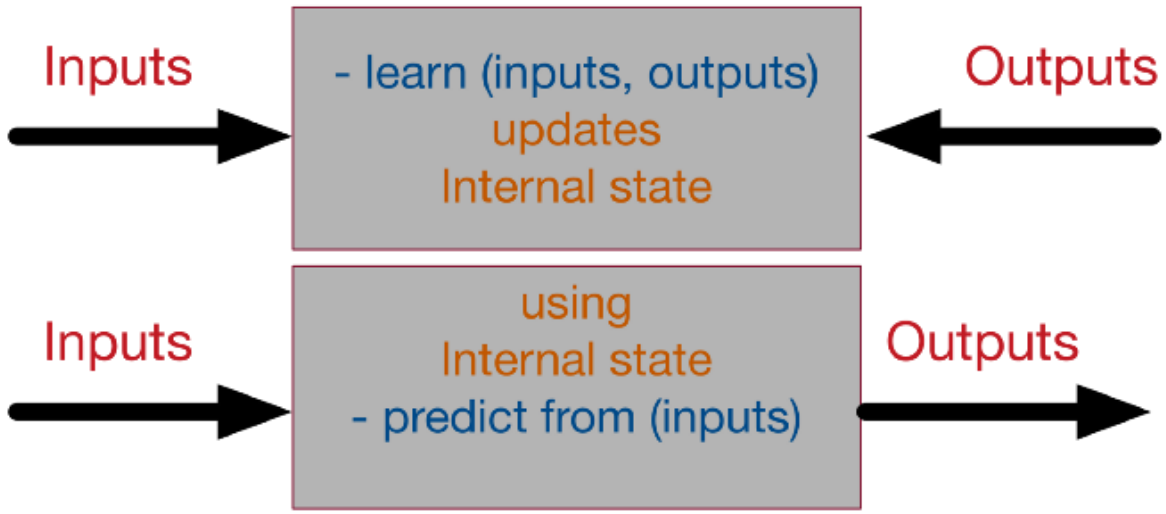



*Figure 5.6: Neural network as a black box*

The learning process takes the inputs and the desired outputs and updates its internal state accordingly, so the calculated output get as close as possible to the desired output. The predict process takes an input and generate, using the internal state, the most likely output according to its past "training experience".

### 5.6.2. Convolutional Neural Network

Convolutional Neural Networks are a special type of feed-forward artificial neural network in which the connectivity pattern between its neuron is inspired by the visual cortex.

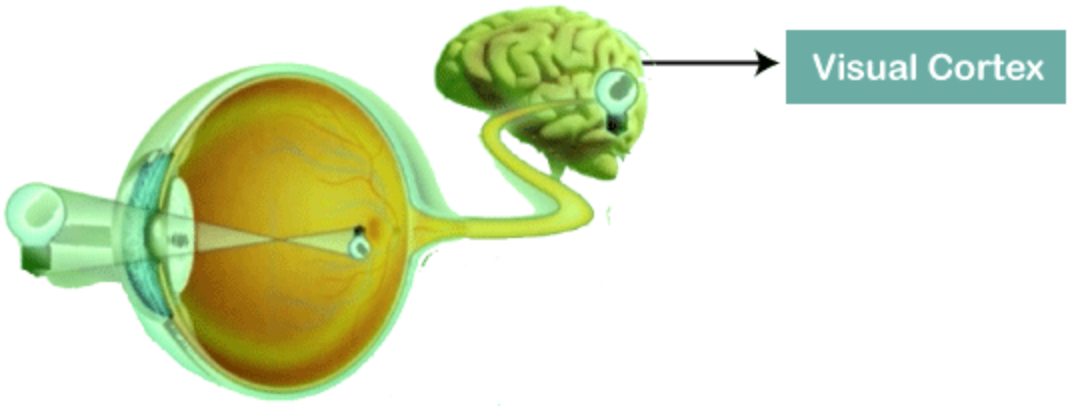



*Figure 5.7: visual Cortex*

The visual cortex encompasses a small region of cells that are region sensitive to visual fields. In case some certain orientation edges are present then only some individual neuronal cells get fired inside the brain such as some neurons responds as and when they get exposed to the vertical edges, however some responds when they are shown to horizontal or diagonal edges, which is nothing but the motivation behind Convolutional Neural Networks.

The Convolutional Neural Networks, which are also called as covnets, are nothing but neural networks, sharing their parameters. Suppose that there is an image, which is embodied as a cuboid, such that it encompasses length, width, and height. Here the dimensions of the image are represented by the Red, Green, and Blue channels, as shown in the image given below.

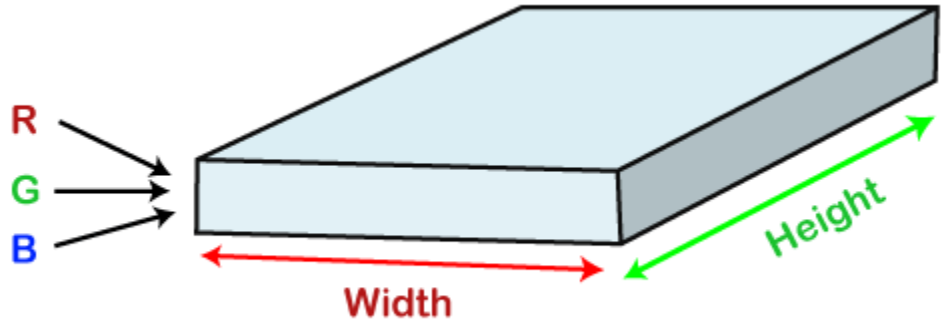



*Figure 5.8: RGB dimension image*

### 5.6.3. Recurrent neural networks and LSTMs

Recurrent neural networks, of which LSTMs ("long short-term memory" units) are the most powerful and well known subset, are a type of artificial neural network designed to recognize patterns in sequences of data, such as numerical times series data emanating from sensors, stock markets and government agencies (but also including text, genomes, handwriting and the spoken word). What differentiates RNNs and LSTMs from other neural networks is that they take time and sequence into account, they have a temporal dimension.

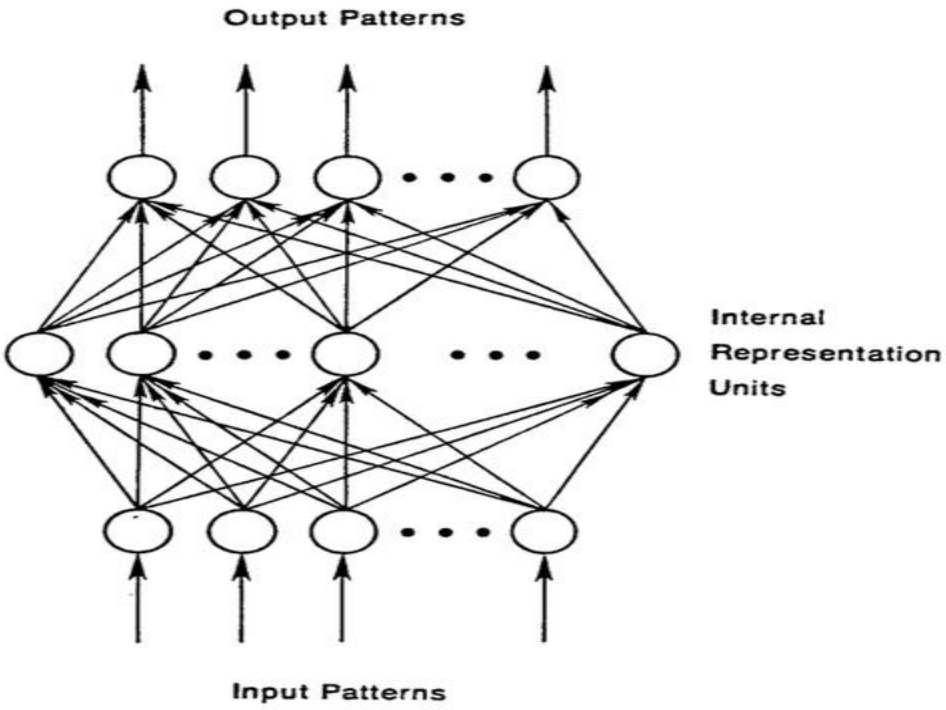


Figure 5.9: RNN temporal dimension

# Chapter 6: Natural Language Processing (NLP) Basics

## 6.1. Intro to Natural Language Processing

Natural language processing (NLP) is an interdisciplinary subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.
NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment. NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly—even in real time. There's a good chance you've interacted with NLP in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.
Machine Learning and Natural Language Processing are important subfields of Artificial Intelligence that have gained prominence in recent times. Machine Learning and Natural Language Processing play a very important part in making an artificial agent into an artificial 'intelligent' agent. An Artificially Intelligent system can accept better information from the environment and can act on the environment in a user-friendly manner because of the advancement in Natural Language Processing. Similarly, an Artificially Intelligent System can process the received information and perform better predictions for its actions because of the adoption of Machine Learning techniques.

Processing of natural language so that the machine can understand the natural language involves many steps. These steps include Morphological Analysis, Syntactic Analysis, Semantic Analysis, Discourse Analysis, and Pragmatic Analysis, generally, these analysis tasks are applied serially.

Machine Learning acts as important value addition in almost all these processes in some form or the other. Let us try to understand how.

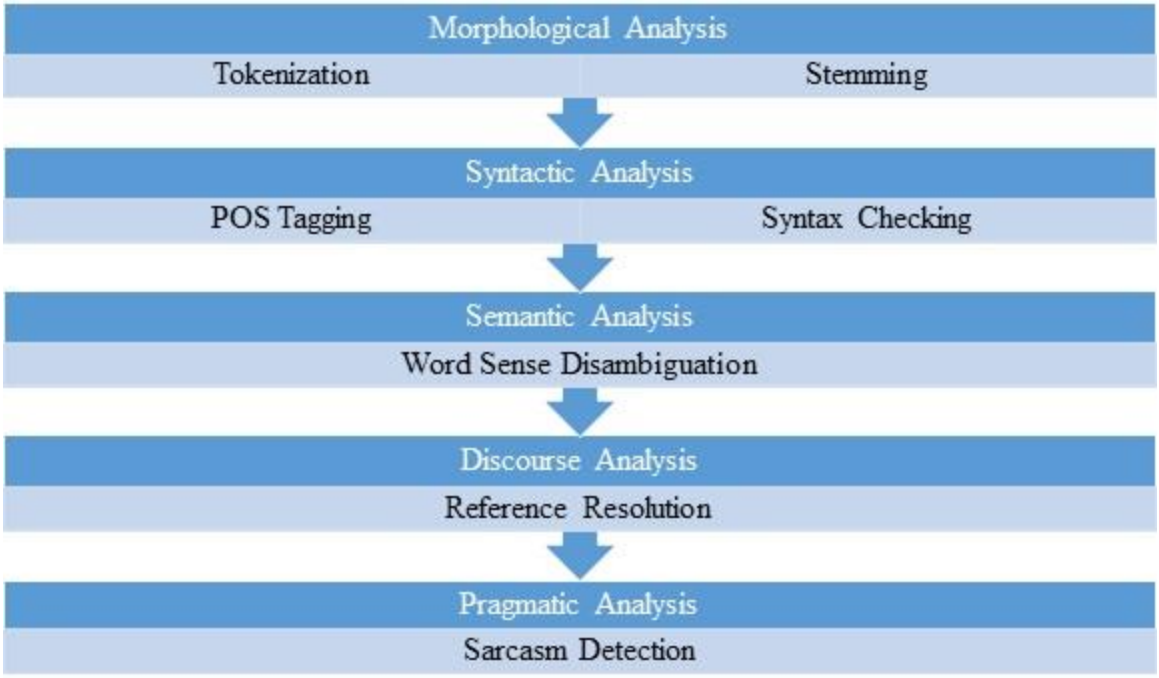



*Figure 6.1: Morphological Analysis*

## 6.2 Machine learning Application in NLP

Most importantly, "machine learning" really means "machine teaching." We know **what** the machine needs to learn, so our task is to create a **learning framework** and provide properly-formatted, relevant, clean **data** for the machine to learn from.

When we talk about a "model," we're talking about a mathematical representation. Input is key. A machine learning model is the sum of the learning that has been acquired from its training data. The model changes as more learning is acquired.

Unlike algorithmic programming, a machine learning model is able to generalize and deal with novel cases. If a case resembles something the model has seen before, the model can use this prior "learning" to evaluate the case. The goal is to create a system where the model continuously improves at the task you've set it.

Machine learning for NLP and text analytics involves a set of statistical techniques for identifying parts of speech, entities, sentiment, and other aspects of text. The techniques can be expressed as a model that is then applied to other text, also known as supervised machine learning. It also could be a set of algorithms that work across large sets of data to extract meaning, which is known as unsupervised machine learning. It's important to understand the difference between supervised and unsupervised learning, and how you can get the best of both in one system.

## 6.3 Natural language interaction

Natural Language Processing is used **by NLI to split the input text into sentences and words and to normalize and pre-process it.** For example, NLP might convert all the words to lowercase or correct spelling errors before determining if the word is an adjective or verb etc.

**Natural language interaction (NLI)** is an interaction style that allows users to interact with computers in a humanlike conversational way. Through NLI, users can converse with computers just like they do with other humans.

Natural language is the way people communicate with each other. The opposite is a computer language, such as programming languages used to communicate with machines. Computers generally cannot understand and interpret natural languages. However, NLI technologies have made it possible to interact with devices using natural languages.

Natural Language Interaction (NLI) is the convergence of a diverse set of natural language principles that enables people to interact with any connected device or service in a humanlike way. Increasingly known as conversational AI, NLI allows technology to understand complex sentences containing multiple pieces of information and more than one request. It can then react accordingly, creating value and enhancing the user experience.

How does natural language interaction work? Natural language interaction uses **natural language processing (NLP)**, a branch of artificial intelligence**,** to make computers understand the user's commands. The commands received by the computer are parsed into sentences and phrases. These are then identified, and the individual actions are performed.

### Natural Language Processing

Natural Language Processing is used by NLI to split the input text into sentences and words and to normalize and pre-process it. For example, NLP might convert all the words to lowercase or correct spelling errors before determining if the word is an adjective or verb etc. and tagging this for future reference.

### Natural Language Understanding

Natural Language Understanding (NLU) encompasses the building blocks to interpret human language. They are the base upon which both general and domain/client/project-specific Language Objects such as lexicon, synonyms and themes, NLU rules and dialogue flows can be constructed in the context of each NLI solution.

### Natural Language Generation

Responding to a query using anything more that pre-scripted responses requires at a minimum, natural-language generation (NLG). This enables NLI to interrogate data, including integrated back-end systems and third-party databases, and to use that information in creating a response, combined with incorporating additional parameters which may be known, for instance user name, gender, location, time of day, appropriate tense, etc.

### Why Syntax, Spelling, and Semantics Matter

Natural Language Interaction technology takes natural language processing (NLP) and natural language understanding (NLU) to the next level. It allows enterprises to create advanced

dialogue systems that utilize memory, personal preferences, and contextual understanding to deliver a proactive natural language interface.

### 6.4 Computer vision and Image processing

What is “computer vision”?
an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do
Includes methods for:

- acquiring
- processing
- analyzing and
- understanding digital images

Deals with the extraction of high-dimensional data from the real world in order to produce numerical or symbolic information

What is “image processing”? any form of signal processing for which the input is an image, such as photographs or frames of video; the output of image processing can be either an image or a set of characteristics or parameters related to the image
Includes:
Image display and printing
Image editing and manipulation
Image enhancement
Feature detection
Image compression

### 6.5 Case study: Sentiment Analysis, speech recognition, Chabot

**Sentiment analysis** (also known as opinion mining or emotion AI) is the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information
Sentiment analysis is a technique through which you can analyze a piece of text to determine the sentiment behind it. It combines machine learning and natural language processing (NLP) to achieve this. Using basic Sentiment analysis, a program can understand whether the sentiment behind a piece of text is positive, negative, or neutral. It is a powerful technique in Artificial intelligence that has important business applications. Sentiment analysis is a powerful tool that you can use to solve problems from brand influence to market monitoring. New tools are built around sentiment analysis to help businesses become more efficient.

**Speech recognition:** Speech recognition most important an interdisciplinary subfield of computer science and computational linguistics that develops methodologies and technologies that enable the recognition and translation of spoken language into text by computers with the

main benefit of search ability. It is also known as automatic speech recognition (ASR), computer speech recognition or speech to text (STT).

# Chapter 7: Robotic Sensing and Manipulation

## 7.1 Introduction to robotics

Robotics is a branch of engineering and science that includes electronics engineering, mechanical engineering and computer science and so on. This branch deals with the design, construction, use to control robots, sensory feedback and information processing. These are some technologies which will replace humans and human activities in coming years. These robots are designed to be used for any purpose but these are using in sensitive environments like bomb detection, deactivation of various bombs etc. Robots can take any form but many of them have given the human appearance. The robots which have taken the form of human appearance may likely to have the walk like humans, speech, cognition and most importantly all the things a human can do. Most of the robots of today are inspired by nature and are known as bio-inspired robots. Robotics is that branch of engineering that deals with conception, design, operation, and manufacturing of robots. There was an author named Issac Asimov, he said that he was the first person to give robotics name in a short story composed in 1940's. In that story, Issac suggested three principles about how to guide these types of robotic machines. Later on, these three principals were given the name of Issac's three laws of Robotics. These three laws state that:

- Robots will never harm human beings.
- Robots will follow instructions given by humans with breaking law one.
- Robots will protect themselves without breaking other rules.

### Characteristics

There are some characteristics of robots given below:

- **Appearance:** Robots have a physical body. They are held by the structure of their body and are moved by their mechanical parts. Without appearance, robots will be just a software program.
- **Brain:** Another name of brain in robots is On-board control unit. Using this robot receive information and sends commands as output. With this control unit robot knows what to do else it'll be just a remote-controlled machine.
- **Sensors:** The use of these sensors in robots is to gather info from the outside world and send it to Brain. Basically, these sensors have circuits in them that produces the voltage in them.
- **Actuators:** The robots move and the parts with the help of these robots move is called Actuators. Some examples of actuators are motors, pumps, and compressor etc. The brain tells these actuators when and how to respond or move.
- **Program:** Robots only works or responds to the instructions which are provided to them in the form of a program. These programs only tell the brain when to perform which operation like when to move, produce sounds etc. These programs only tell the robot how to use sensors data to make decisions.

- **Behaviour:** Robots behavior is decided by the program which has been built for it. Once the robot starts making the movement, one can easily tell which kind of program is being installed inside the robot.

### 7.1.1 Sensing

Robotic sensing is **a subarea of robotics science intended to provide sensing capabilities to robots.** Robotic sensing provides robots with the ability to sense their environments and is typically used as feedback to enable robots to adjust their behavior based on sensed input.

Robotic sensors are used to estimate a robot's condition and environment. These signals are passed to a controller to enable appropriate behavior. Sensors in robots are based on the functions of human sensory organs. Robots require extensive information about their environment in order to function effectively.
A robot's sensors provide information that can be used to infer things about the world, about the robot, and about the robot's location in the world. Sensors can provide simple information (e.g., a simple collision sensor on a vacuuming robot), or rich, highly complex data (e.g., an RGB-D camera, which provides a color image along with a depth estimate for each pixel). In the ideal case, the sensing process can be modeled using the laws of physics (e.g., how light reflects off of object surfaces), or sometimes simple geometry (e.g., stereo vision relies uses simple triangulation to determine the distance to an object).
Sensors, of course, are not perfect, and each type of sensor has its own specific type of uncertainty. This uncertainty is typically characterized using models that describe the probability of a particular measurement given that certain physical conditions occur. These conditional probabilities are typically called observation models. As will be described in the next section, these observation models, along with the sensor values, can be used for inference about the world, which defines perception.

Figure 7.1: robot sensing

### 7.1.2 Manipulation

Robotic manipulation refers to the **ways robots interact with the objects around them:** grasping an object, opening a door, packing an order into a box, folding laundry… All these actions require robots to plan and control the motion of their hands and arms in an intelligent way. At Leeds, our research focuses on developing such algorithms and systems. We focus on robots manipulating objects in unstructured human environment, such as our homes, as well as robots working on manufacturing and assembly tasks.
A robot is a machine that is programmed to automatically perform specific tasks predictably. To do this, it has to handle objects and either move them or impact them through actions like welding or drilling. It is this handling or procedures performed on an object that is referred to as robotic manipulation. This definition thereby gives us context to define robotic manipulators.

A robotic manipulator is an arm-like structure joined to the body of a robot and is used to execute tasks. A robot without a manipulator would be the equivalent of a person performing a task with their hands tied behind their back, it simply is not viable. It is, in fact, for this reason that robotic manipulators are also known as robot arms.
Robot manipulators consist of a series of joints and links fused in their interior structure. From the outside, the **robotic manipulator arm** only seems to be mounted onto the robot. However, the series of joints and links go further into the robot body to bind them. This ensures that the robot and the manipulator work together in coordinated fluid motions

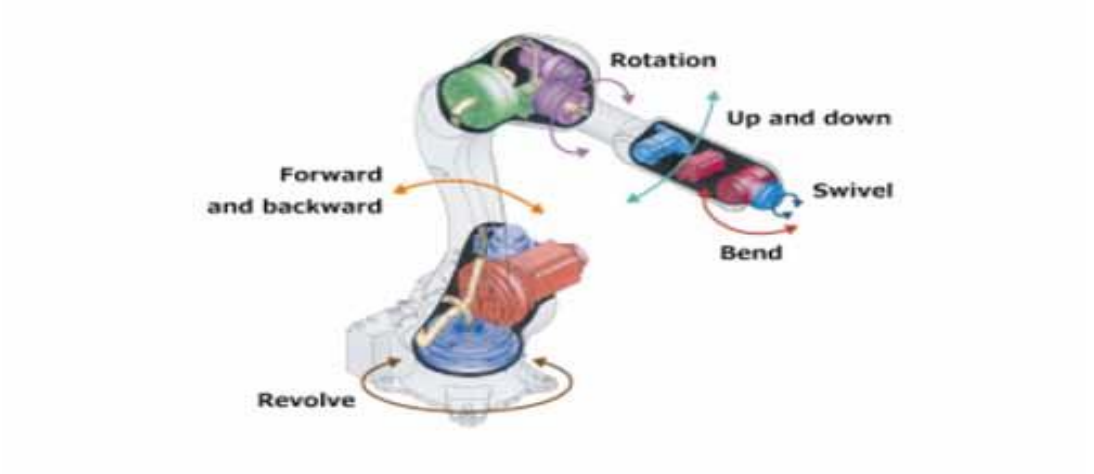



Figure 7.2: Robot manipulators arm

### 7.1.3 Human-robot interaction

Simply put human-robot interaction in the field of robotics deals with the interaction between humans and robots. There are several aspects to human-robot interaction, including the interactions between humans and computers, artificial intelligence, natural language processing, design, robotics, and psychology.
Human–robot interaction has been a topic of both science fiction and academic speculation even before any robots existed. Because much of active HRI development depends on natural

language processing, many aspects of HRI are continuations of human communications, a field of research which is much older than robotics.

The origin of HRI as a discrete problem was stated by 20th-century author Isaac Asimov in 1941, in his novel *I, Robot*. Asimov coined Three Laws of Robotics, namely:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Human-Robot Interaction (HRI) is a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with humans. Interaction, by definition, requires communication between robots and humans. Communication between a human and a robot may take several forms, but these forms are largely influenced by whether the human and the robot are in close proximity to each other or not. Thus, communication and, therefore, interaction can be separated into two general categories:

Remote interaction – The human and the robot are not co-located and are separated spatially or even temporally (for example, the Mars Rovers are separated from earth both in space and time). Proximate interactions – The humans and the robots are co-located (for example, service robots may be in the same room as humans).Within these general categories, it is useful to distinguish between applications that require mobility, physical manipulation, or social interaction. Remote interaction with mobile robots often is referred to as teleoperation or supervisory control, and remote interaction with a physical manipulator is often referred to as telemanipulation. Proximate interaction with mobile robots may take the form of a robot assistant, and proximate interaction may include a physical interaction. Social interaction includes social, emotive, and cognitive aspects of interaction.

Figure 7.3: Human-Robot Interaction

## 7.2 Navigation and path planning

Robot navigation means the robot's ability to **determine its own position in its frame of reference and then to plan a path towards some goal location**. In order to navigate in its environment, the robot or any other mobility device requires representation, i.e. a map of the environment and the ability to interpret that representation.

Navigation can be defined as the combination of the three fundamental competences:

1. Self-localization
2. Path planning
3. Map-building and map interpretation

**Robot localization** denotes the robot's ability to establish its own position and orientation within the frame of reference. Path planning is effectively an extension of localization, in that it requires the determination of the robot's current position and a position of a goal location, both within the same frame of reference or coordinates. Map building can be in the shape of a metric map or any notation describing locations in the robot frame of reference.

**Motion planning**, also **path planning** (also known as the **navigation problem** or the **piano mover's problem**) is a computational problem to find a sequence of valid configurations that moves the object from the source to destination. The term is used in computational geometry, computer animation, robotics, computer games.

**Robotic mapping** is a discipline related to computer vision and cartography. The goal for an autonomous robot is to be able to construct (or use) a map (outdoor use) or floor plan (indoor use) and to localize itself and its recharging bases or beacons in it. Robotic mapping is that branch which deals with the study and application of ability to localize itself in a map / plan and sometimes to construct the map or floor plan by the autonomous robot.

Path planning is an important issue as it allows a robot to get from point A to point B. Path planning algorithms are measured by their computational complexity. The feasibility of real-time motion planning is dependent on the accuracy of the map (or floor plan), on robot localization and on the number of obstacles. Topologically, the problem of path planning is related to the shortest path problem of finding a route between two nodes in a graph.

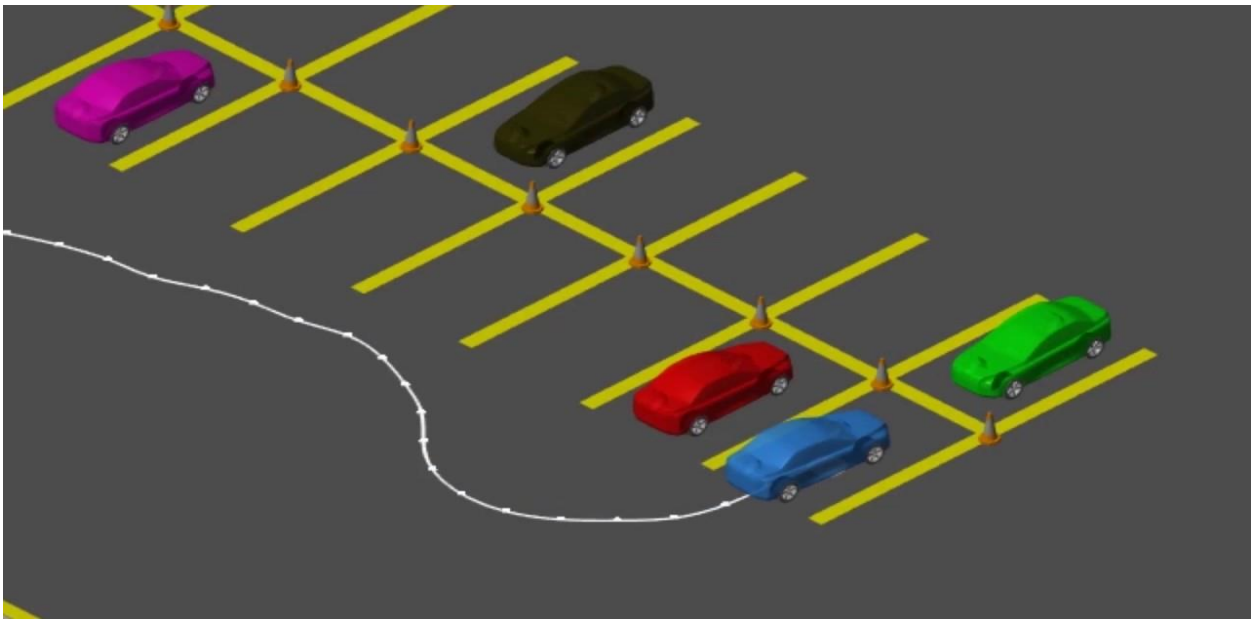


Figure 7.4: path planning and navigation for autonomous robot

### 7.2.1 Autonomous robotic systems

*Robotics and Autonomous Systems* will carry articles describing fundamental developments in the field of **robotics**, with special emphasis on **autonomous systems**. An important goal of this journal is to extend the state of the art in both **symbolic** and **sensory** based **robot control** and **learning** in the context of autonomous systems.
*Robotics and Autonomous Systems* will carry articles on the theoretical, computational and experimental aspects of autonomous systems, or modules of such systems.
An autonomous robot is **a robot that acts without recourse to human control.** The first autonomous robots environments were known as Elmer and Elsie, which were constructed in the late 1940s by W. Grey Walter.

# Chapter 8: Ethical and Legal Considerations in AI

Today, artificial intelligence plays a role in billions of people's lives. Sometimes unnoticed but often with profound consequences, it transforms our societies and challenges what it means to be human. Ethics and law are inextricably linked in modern society, and many legal decisions arise from the interpretation of various ethical issues. Artificial intelligence adds a new dimension to these questions. Systems that use artificial intelligence technologies are becoming increasingly autonomous in terms of the complexity of the tasks they can perform, their potential impact on the world and the diminishing ability of humans to understand, predict and control their functioning. Most people underestimate the real level of automation of these systems, which have the ability to learn from their own experience and perform actions beyond the scope of those intended by their creators.
Many countries are actively creating the legal conditions for the development of technologies that use artificial intelligence. For example, the "Intelligent Robot Development and Dissemination Promotion Law" has been in place in South Korea since 2008. The law is aimed at improving the quality of life and developing the economy through the creation and promotion of a strategy for the sustainable development of the smart robot industry. Every five years, the government works out a basic plan to ensure that these goals are achieved.

## 8.1 Privacy

Privacy is the right to keep personal information confidential and free from unauthorised access. It is an essential human right that ensures individuals have control over their personal data and how it is used. Today, privacy is more important than ever as the amount of personal data collected and analyzed continues to grow.
Privacy is crucial for a variety of reasons. For one, it protects individuals from harm, such as identity theft or fraud. It also helps to maintain individual autonomy and control over personal information, which is essential for personal dignity and respect. Furthermore, privacy allows individuals to maintain their personal and professional relationships without fear of surveillance or interference. Last but not least, it protects our free will; if all our data is publicly available, toxic recommendation engines will be able to analyze our data and use it to manipulate individuals into making certain (buying) decisions.

In the context of AI, privacy is **essential to ensure that AI systems are not used to manipulate individuals or discriminate against them based on their personal data**. AI systems that rely on personal data to make decisions must be transparent and accountable to ensure that they are not making unfair or biased decisions.
The importance of privacy in the digital era cannot be overstated. It is a fundamental human right that is necessary for personal autonomy, protection, and fairness. As AI continues to become more prevalent in our lives, we must remain vigilant in protecting our privacy to ensure that technology is used ethically and responsibly.

### Privacy challenges in the age of AI

AI presents a challenge to the privacy of individuals and organisations because of the complexity of the algorithms used in AI systems. As AI becomes more advanced, it can make decisions

based on subtle patterns in data that are difficult for humans to discern. This means that individuals may not even be aware that their personal data is being used to make decisions that affect them.

**The issue of violation of privacy**

AI presents a challenge to the privacy of individuals and organizations because of the complexity of the algorithms used in AI systems. As AI becomes more advanced, it can make decisions based on subtle patterns in data that are difficult for humans to discern. This means that individuals may not even be aware that their personal data is being used to make decisions that affect them.

### 8.2 Bias

AI bias is an anomaly in the output of machine learning algorithms, due to the prejudiced assumptions made during the algorithm development process or prejudices in the training data.

### 8.2.1. What are the types of AI bias?

AI systems contain biases due to two reasons:

- **Cognitive biases:** These are unconscious errors in thinking that affects individuals' judgments and decisions. These biases arise from the brain's attempt to simplify processing information about the world. More than 180 human biases have been defined and classified by psychologists. Cognitive biases could seep into machine learning algorithms via either
    - designers unknowingly introducing them to the model
    - a training data set which includes those biases
- **Lack of complete data:** If data is not complete, it may not be representative and therefore it may include bias. For example, most psychology research studies include results from undergraduate students which are a specific group and do not represent the whole population.

Another challenge posed by AI technology is the potential for bias and discrimination. AI systems are only as unbiased as the data they are trained on; if that data is biased, the resulting system will be too. This can lead to discriminatory decisions that affect individuals based on factors such as race, gender, or socioeconomic status. It is essential to ensure that AI systems are trained on diverse data and regularly audited to prevent bias.

At first glance, the link between bias and discrimination in AI and privacy may not be immediately apparent. After all, privacy is often thought of as a separate issue related to the protection of personal information and the right to be left alone. However, the reality is that the two issues are intimately connected, and here's why.

To start with, it is important to note that many AI systems rely on data to make decisions. This data can come from a variety of sources, such as online activity, social media posts, and public records. While this data may seem innocuous at first, it can reveal a lot about a person's life,

including their race, gender, religion, and political beliefs. As a result, if an AI system is biased or discriminatory, it can use this data to perpetuate these biases, leading to unfair or even harmful outcomes for individuals.

For example, imagine an AI system used by a hiring company to screen job applications. If the system is biased against women or people of color, it may use data about a candidate's gender or race to unfairly exclude them from consideration. This harms the individual applicant and perpetuates systemic inequalities in the workforce.

### 8.3 AI and the future of work

Automation and **artificial intelligence** (AI) are transforming businesses and will contribute to economic growth via contributions to productivity. They will also help address "moonshot" societal challenges in areas from health to climate change. At the same time, these technologies will transform the nature of work and the workplace itself. Machines will be able to carry out more of the tasks done by humans, complement the work that humans do, and even perform some tasks that go beyond what humans can do. As a result, some occupations will decline, others will grow, and many more will change. While we believe there will be enough work to go around (barring extreme scenarios), society will need to grapple with significant workforce transitions and dislocation. Workers will need to acquire new skills and adapt to the increasingly capable machines alongside them in the workplace. They may have to move from declining occupations to growing and, in some cases, new occupations.

AI, or artificial intelligence, seems to be on the tip of everyone's tongue these days. While I've been aware of this major trend in tech development for a while, I've noticed AI appearing more and more as one of the most in-demand areas of expertise for job seekers. I'm sure that for many of us, the term "AI" conjures up sci-fi fantasies or fear about robots taking over the world. The depictions of AI in the media have run the gamut, and while no one can predict exactly how it will evolve in the future, the current trends and developments paint a much different picture of how AI will become part of our lives.

### 8.4 Appropriate uses of AI

Artificial Intelligence has many practical applications across various industries and domains, including:

1. **Healthcare:** AI is used for medical diagnosis, drug discovery, and predictive analysis of diseases.
2. **Finance:** AI helps in credit scoring, fraud detection, and financial forecasting.
3. **Retail:** AI is used for product recommendations, price optimization, and supply chain management.
4. **Manufacturing:** AI helps in quality control, predictive maintenance, and production optimization.
5. **Transportation:** AI is used for autonomous vehicles, traffic prediction, and route optimization.
6. **Customer service:** AI-powered chatbots are used for customer support, answering frequently asked questions, and handling simple requests.

7. **Security:** AI is used for facial recognition, intrusion detection, and cyber security threat analysis.
8. **Marketing:** AI is used for targeted advertising, customer segmentation, and sentiment analysis.
9. **Education:** AI is used for personalized learning, adaptive testing, and intelligent tutoring systems.

>>>>>>>>>>>>>>>>>>>>>>>Laboratory Contents>>>>>>>>>>>>>>>>>>>>>>>>>>>

**Lab contents: With python or prolog**

**Prolog:** Prolog is considered as a declarative language rather than programming language as it is not used to solve computational problems. It is used to solve logical problems which use rules or facts. The full form of prolog is programming in logic. Prolog is used in much artificial intelligence, natural language processing, and machine learning problems. Prolog is used to write facts or rules for problems and then prolog's run time system checks those facts and rules whenever that problem is asked and returns a true or false statement for success or failure of the problem.

**Installing** SWI Prolog on Windows: **Follow the below steps to install SWI Prolog on Windows:**

**Lab 1: Tool installation and configuration, introduction to the tool**

**Step 1:** Visit *swi-prolog.org* website using any web browser.
**Step 2:** Click on Download which is adjacent to Home, dropdown list will appear then click on SWI-Prolog.
**Step 3:** New webpage will open, click on Stable release.
**Step 4:** After clicking on stable release new webpage will open which will contain stable versions of prolog for different platforms.
**Step5.** After clicking on SWI-Prolog 8.4.0-1 for Microsoft Windows (64 bit), a new webpage will open, check on I understand checkbox to make the download link active.
**Step 6:** Now check for the executable file in downloads in your system and run it.
**Step 7:** It will prompt confirmation to make changes to your system. Click on Yes.
**Step 8:** Setup screen will appear, click on Next.
**Step 9:** The next screen will be of License Agreement, click on I Agree.
**Step 10:** After it there will be screen of installing options so check the box for Add swipl to the system path for all users, and also check the box for create a desktop icon and then click on the Next button.
**Step 11:** The next screen will be of installing location so choose the drive which will have sufficient memory space for installation. It needed only a memory space of 50 MB.
**Step 12:** Next screen will be of choosing Start menu folder so don't do anything just click on Next Button.
**Step 13:** This last screen is of choosing components, all components are already marked so don't change anything just click on Install button.
**Step 14:** After this installation process will start and will hardly take a minute to complete the installation.
**Step 15:** Click on Finish after the installation process is complete.
**Step 16:** SWI Prolog is successfully installed on the system and an icon is created on the desktop.
**Step 17:** Run the software and see the interface.

The implementation can be either using prolog or python programming

Lab 2: Implementing search strategies

Lab 3: Knowledge representation

Lab 4: Knowledge Reasoning

Lab 5: Implementing knowledge base system

Lab 6: Implementing neural network

## References


1. Artificial Intelligence A Modern Approach Third Edition, Stuart J. Russell and Peter Norvig
2. A review: On path planning strategies for navigation of mobile robot - ScienceDirect
3. What Is Natural Language Interaction? - DZone The Ethical and Legal Issues of Artificial Intelligence - Modern Diplomacy