RAMAIAH Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Artificial Intelligence(AI)

Course Code: CS53

Credits: 3:0:1

Term: October 2021 – Feb 2022

UNIT -3

Faculty:
Dr. Sangeetha V
Assistant Professor
Department of CSE

OUTLINE

Classical Planning

- Definition
- Planning with State-Space Search
- Planning Graph
- Other Planning Approaches Analysis

Uncertainty

Learning From Examples

Introduction

- Intelligent robots or self-driving cars or smart cities, they will all use different aspects of Artificial Intelligence!!!
- But to create such AI model, Planning is very important.
- Planning is required for every task(Job)

Introduction

For example, reaching a particular destination requires planning.

- Finding the best route and then identify a set of actions to be done at a particular time is very important.

Introduction

For Example: Imagine to build a house.

- On a building site we can assume all the materials are ready to be used and we essentially now have to decide which order to perform construction in.
- Each step of construction in this situation could be modelled as an action (for example laying foundations, building the walls, installing the plumbing and electricals etc.)
- Use of predicate logic we can express this problem by saying "if there is no foundation, we cannot build a wall".

RAMAIAH
Institute of Technology

Introduction

SR. NO.	PARAMETERS	PLANNING	PROBLEM SOLVING
1	Involved in	Plan Generation	Plan Execution
2	States	Logical Sentences	Data Structures
3	Actions	Preconditions/Outcomes	Codes
4	Plan	Constraints on actions	Sequential
5	Definition	It is a task of coming up with a sequence of actions to achieve a goal.	It is process of performing systematic search, through a range of possible actions, to reach some predefined solution.
6	Solution	Viewed as generator of solution	Demonstrates one specific solution
7	Planning Approach	Backward Planning	Forward Planning
8	Eg.	STRIPS, PDDL	A* Algorithm

Definition

- Planning is deciding a set of actions to be performed, in agent environment based on the perception to achieve the desired goal.
- Classical Planning: When the environment is
  - Fully observable
  - Deterministic
  - Static
  - Discrete
- Planning is a problem solving procedure before executing them.

Languages for Planning Problem

- STRIPS
  - Stanford Research Institute Problem Solver
  - Historically important
- ADL
  - Action Description Languages
- PDDL
  - Planning Domain Definition Language
  - Revised & enhanced for the needs of the International Planning Competition
  - Currently version 3.1

RAMAIAH
Institute of Technology

Definition

Planning Domain Definition Language(PDDL) a language used to represent all actions into one action schema.

PDLL describes the four basic things needed in a search problem:

- **Initial state**
- **Actions**
- **Result**
- **Goal**

$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{EFFECT: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Block(b) \land (b \neq x), \\
&\quad \text{EFFECT: } On(b, Table) \land Clear(x) \land \neg On(b, x))
\end{aligned}
$$

Definition

$$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C))$$

Initial state

It is the representation of each state as the conjunction of the ground and functionless atoms.

"Ground" means that no variables are used.

Example : At(x, y) is illegal, because it uses variables x, y.

"Functionless" means that no functions are used.

Example: At(Father(George), JFK) is illegal, because it uses function Father.

Note : state descriptions must be ground (cannot include variables), but preconditions can include variables.

Definition

Preconditions and effects are conjunctions of functionless literals.

Actions

It is defined by a set of action schemas which implicitly define the **ACTION()** and **RESULT()** functions.

- Each action has its own set of preconditions
- All the preconditions to be satisfied before performing the action
- Some effects can be positive or negative

Action schema:
- ACTION: specifies name and parameter list
- PRECONDITION: conjunction of positive literals
- EFFECT: conjunction of literals (positive or negative)

$$Action(Move(b,x,y),$$
$$\text{PRECOND: } On(b,x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y),$$
$$\text{EFFECT: } On(b,y) \wedge Clear(x) \wedge \neg On(b,x) \wedge \neg Clear(y))$$

Definition

Result

It is obtained by the set of actions used by the agent.

Goal

Is a conjunction of literals (whose value is either positive or negative).

$$Goal(On(A, B) \wedge On(B, C))$$

Definition

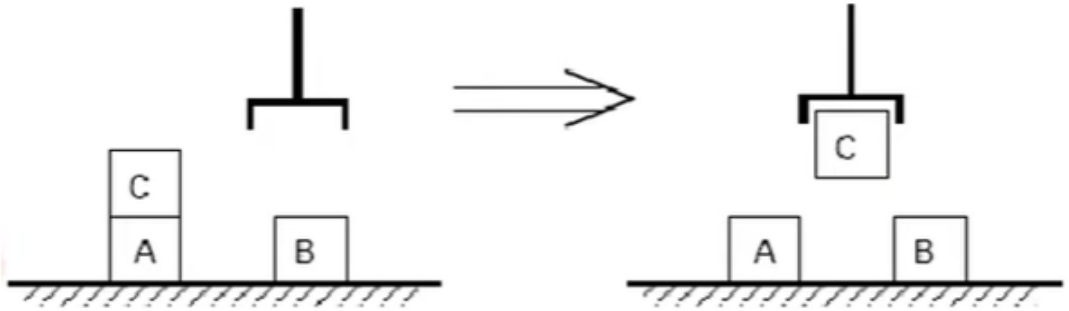
There are examples which will make PDLL understandable:

1. The Blocks world
2. Air cargo transport
3. The spare tire problem

Formulate the PDDL description of any planning problem.

The Blocks world Problem

The Blocks world consists of:



A flat surface such as a table top

An adequate set of identical blocks which are identified by letters.

The blocks can be stacked one upon another.

There is a robot arm that can manipulate the blocks.

The robot can hold one block at a time and only one block can be moved at a time.

Any number of blocks can be on the table.

The Blocks world Problem

PDLL describes the four basic things needed in a problem

- Initial state
- Actions
- Result
- Goal: To build one or more stacks of blocks

  block A on B and block B on C

The Blocks world Problem

- To represent the blocks world using PDDL, we need to define states and actions.
- To define states and actions, we need to specify constants and predicates.
- What are our constants? A, B, C, Table.
- What are our predicates?

The Blocks world Problem

Predicates

The following predicates are needed to perform an operation:

ON(A, B): Block A is on Block B.

ONTABLES(A): Block A is on the table.

CLEAR(A): There is nothing on the top of Block A.

HOLDING(A): The arm is holding Block A.

ARMEMPTY: The arm is holding nothing.

The Blocks world Problem

Actions

- **Move(b, x, y)**
  - Move block b from the top of x to the top of y
  - No other block should be on it

- **MoveToTable(b, x)**
  - Move a block b from x to the table
  - No other block should be on it

- **Clear(x)**
  There is a clear space on x to hold a block

RAMAIAH
Institute of Technology

The Blocks world Problem

Build a three-block tower.

One solution is the sequence

[

MoveToTable(C, A),

Move(B, Table, C),

Move(A, Table, B)

]

Start State

Goal State

$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{EFFECT: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Block(b) \land (b \neq x), \\
&\quad \text{EFFECT: } On(b, Table) \land Clear(x) \land \neg On(b, x))
\end{aligned}
$$

Air cargo transport

Air cargo transport problem

- Loading and unloading cargo
- Flying it from one place to other place.

Air cargo transport

Actions:

- **Load:** This action is taken to load cargo.
- **Unload:** This action is taken to unload the cargo when it reaches its destination.
- **Fly:** This action is taken to fly from one place to another.

Therefore, the Air cargo transport problem is based on loading and unloading the cargo and flying it from one place to another.

Air cargo transport

Predicates:

In(c, p) ⍰ cargo c is inside plane p

At(x, a) ⍰ object x (either plane or cargo) is at airport a

At really means “available for use at a given location.”

RAMAIAH
Institute of Technology

Air cargo transport

$$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$$
$$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$$
$$\land Airport(JFK) \land Airport(SFO))$$
$$Goal(At(C_1, JFK) \land At(C_2, SFO))$$
$$Action(Load(c, p, a),$$
$$\text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$$
$$\text{EFFECT: } \neg At(c, a) \land In(c, p))$$
$$Action(Unload(c, p, a),$$
$$\text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$$
$$\text{EFFECT: } At(c, a) \land \neg In(c, p))$$
$$Action(Fly(p, from, to),$$
$$\text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$$
$$\text{EFFECT: } \neg At(p, from) \land At(p, to))$$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

The spare tire problem

Consider the problem of changing a flat tire

Initial state Flat tire on the axle and Good spare tire in the trunk.

Goal Good spare tire properly mounted onto the car's axle

Assume that the car is parked in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.

The spare tire problem

Actions: 4

- removing the spare from the trunk
- removing the flat tire from the axle
- putting the spare on the axle
- leaving the car unattended overnight

$$
\begin{aligned}
&Init(At(Flat, Axle) \wedge At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(Spare, Trunk),\\
&\quad \text{PRECOND: } At(Spare, Trunk)\\
&\quad \text{EFFECT: } \neg At(Spare, Trunk) \wedge At(Spare, Ground))\\
&Action(Remove(Flat, Axle),\\
&\quad \text{PRECOND: } At(Flat, Axle)\\
&\quad \text{EFFECT: } \neg At(Flat, Axle) \wedge At(Flat, Ground))\\
&Action(PutOn(Spare, Axle),\\
&\quad \text{PRECOND: } At(Spare, Ground) \wedge \neg At(Flat, Axle)\\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \wedge At(Spare, Axle))\\
&Action(LeaveOvernight,\\
&\quad \text{PRECOND:}\\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)\\
&\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))
\end{aligned}
$$

The spare tire problem

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
PRECOND: $At(Spare, Trunk)$
EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
PRECOND: $At(Flat, Axle)$
EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

OUTLINE

Planning

- Definition
- Planning with State-Space Search
- Planning Graph
- Other Planning Approaches Analysis

Uncertainty

Learning From Examples

Algorithms for Planning as State-Space Search

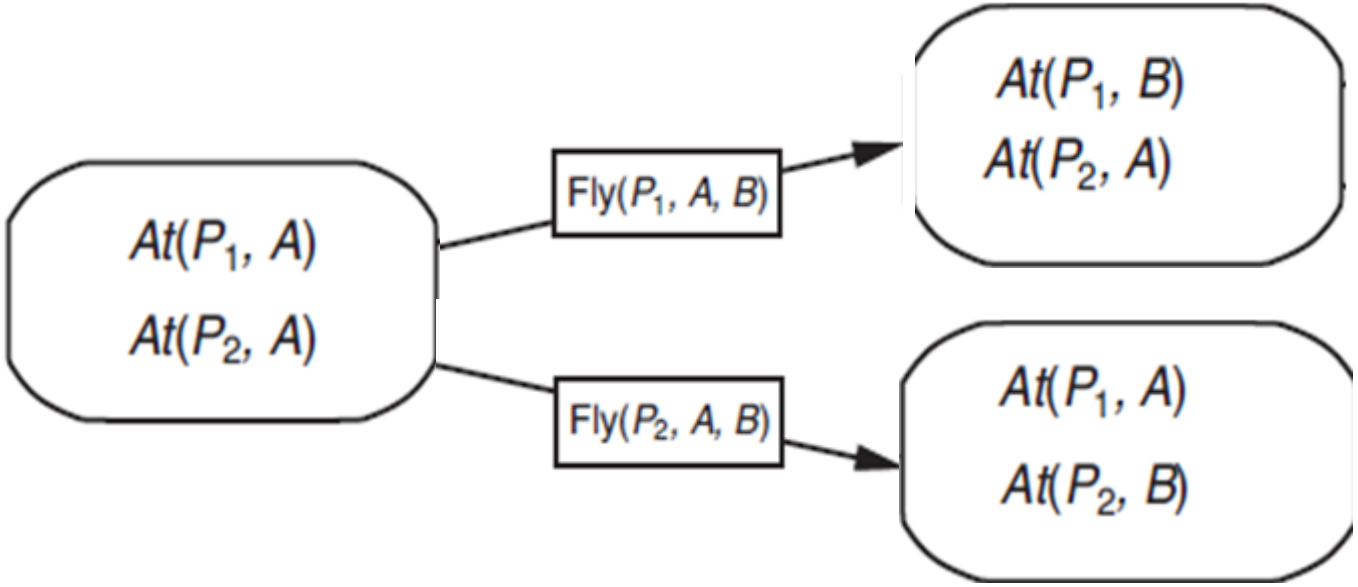
Planning problem defines a search problem:

- We can search from the initial state through the space of states, looking for a goal.

  - Forward state-space search (Progression)
  - Backward state-space search (Regression)

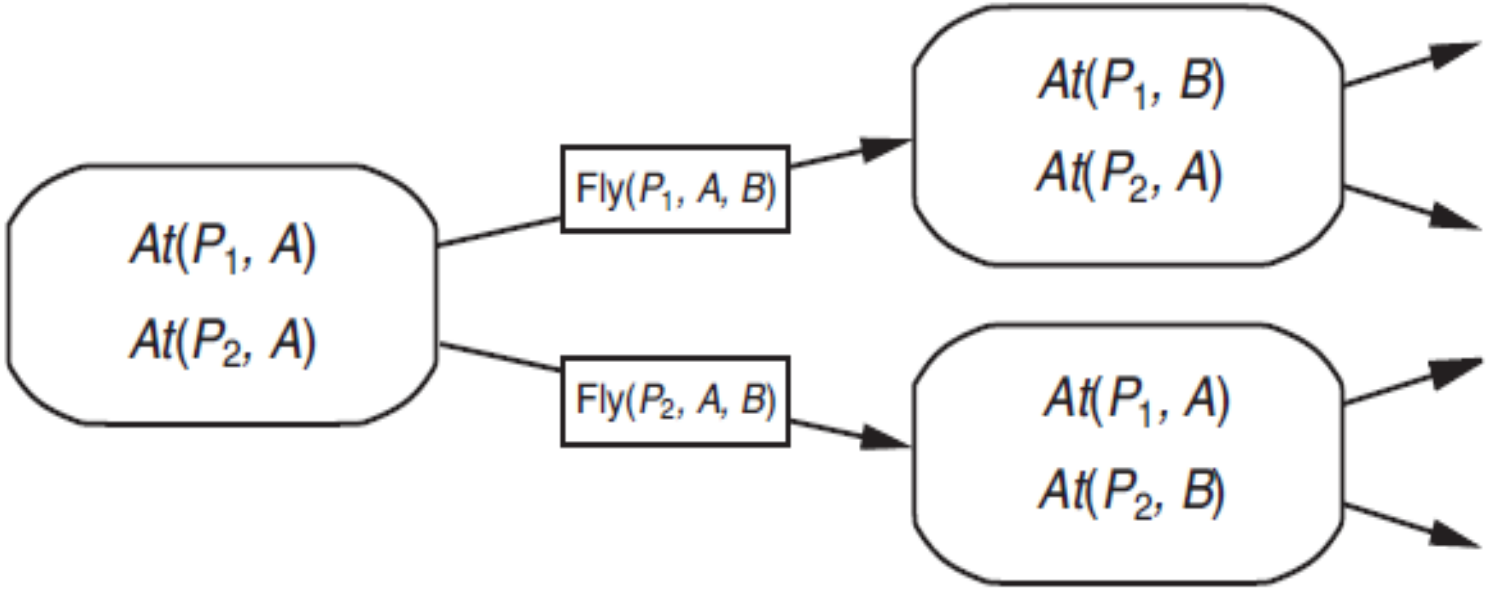
Algorithms for Planning as State-Space Search

Forward state-space search (Progression)- Air cargo problem



Algorithms for Planning as State-Space Search

Forward state-space search (Progression)



Algorithms for Planning as State-Space Search

Example 1: Consider the task of buying a copy of book

AI: A Modern Approach from an online bookseller

- Suppose there is an action schema Buy(isbn) with effect Own(isbn).
- ISBNs are 10 digits, so this action schema represents 10 billion ground actions.
- An uninformed forward-search algorithm would have to start enumerating these 10 billion actions to find one that leads to the goal.

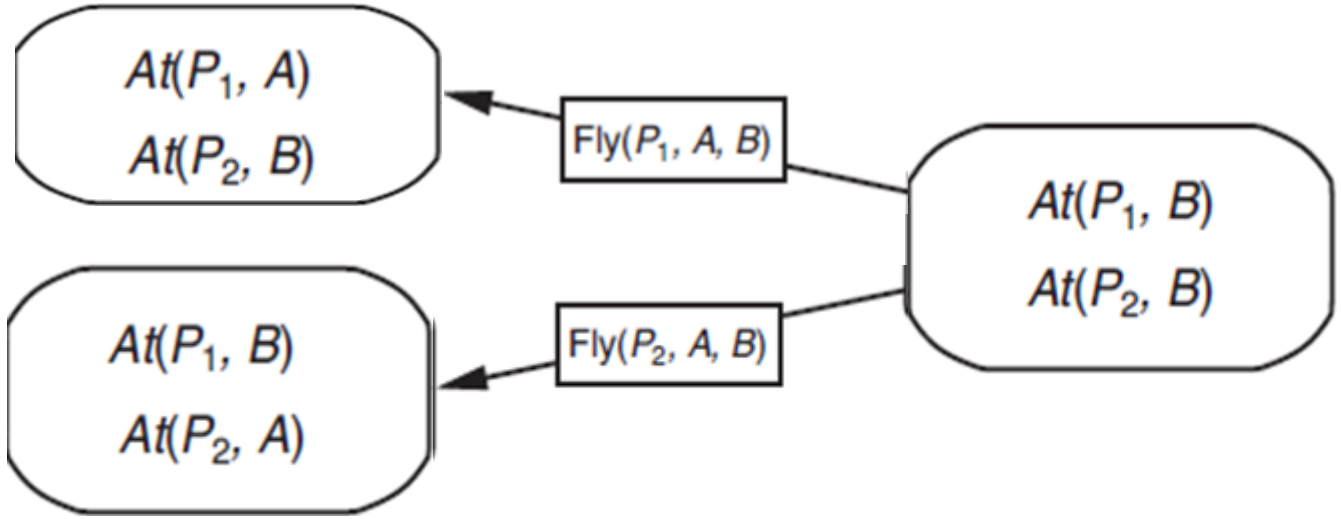
Algorithms for Planning as State-Space Search

Example 2: Air cargo problem with 10 airports, where each airport has 5 planes and 20 pieces of cargo.

- The goal is to move all the cargo at airport A to airport B.
- There is a simple solution to the problem: load the 20 pieces of cargo into one of the planes at A, fly the plane to B, and unload the cargo.

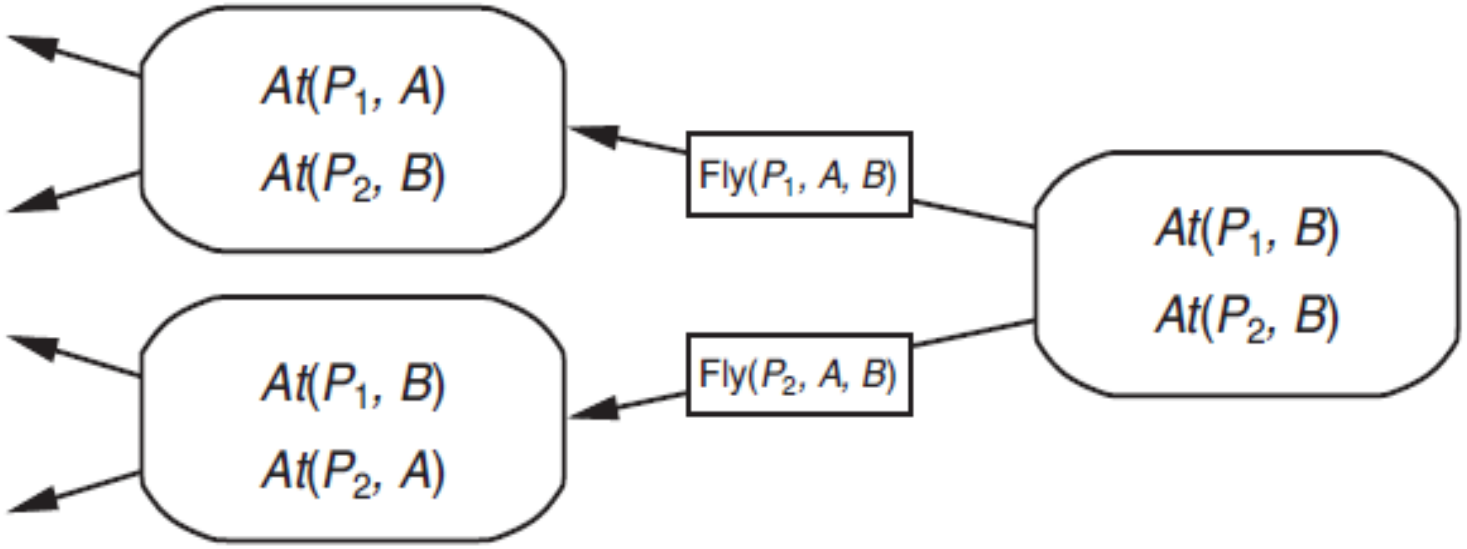
Algorithms for Planning as State-Space Search

Backward state-space search (Regression)



Algorithms for Planning as State-Space Search

Backward state-space search (Regression)



Algorithms for Planning as State-Space Search

- Planning problems often have large state spaces.
- Clearly, relatively small problem instance is hopeless without an accurate heuristic.
- Neither forward nor backward search is efficient without a good heuristic function.

Heuristics for planning

- Think of a search problem as a graph where the nodes are states and the edges are actions.
- The problem is to find a path connecting the initial state to a goal state.
- There are two ways we can relax this problem to make it easier:

1. Adding more edges to the graph, making it strictly easier to find a path.
2. Grouping multiple nodes together, forming an abstraction of the state space that has fewer states, and thus is easier to search

OUTLINE

Planning

- Definition
- Planning with State-Space Search
- Planning Graph
- Other Planning Approaches Analysis

Uncertainty

Learning From Examples

Planning Graph

- Heuristics can still suffer from inaccuracies.
- A special data structure called a planning graph can be used to give better heuristic estimates.
- These heuristics can be applied to any of the search techniques.
- We can search for a solution over the space formed by the planning graph, using an algorithm called GRAPHPLAN.

Planning Graph

- A planning graph is a directed graph organized into levels:
- First a level S_0 for the initial state, consisting of nodes representing each fluent that holds in S_0
- Then a level A_0 consisting of nodes for each ground action that might be applicable in S_0
- Then alternating levels S_0 followed by A_i; until we reach a termination condition.

Planning Graph

- A planning graph is a directed graph organized into levels:
- First a level S_0 for the initial state, consisting of nodes representing each fluent that holds in S_0
- Then a level A_0 consisting of nodes for each ground action that might be applicable in S_0
- Then alternating levels S_0 followed by A_i; until we reach a termination condition.

Planning Graph Example

Consider the Cake problem:

$$
\begin{aligned}
&Init(Have(Cake))\\
&Goal(Have(Cake) \wedge Eaten(Cake))\\
&Action(Eat(Cake)\\
&\quad \text{PRECOND: } Have(Cake)\\
&\quad \text{EFFECT: } \neg Have(Cake) \wedge Eaten(Cake))\\
&Action(Bake(Cake)\\
&\quad \text{PRECOND: } \neg Have(Cake)\\
&\quad \text{EFFECT: } Have(Cake))
\end{aligned}
$$

Figure 10.7 The "have cake and eat cake too" problem.

We can see how to build planning graph

Planning Graph Example

S_0 A_0 S_1

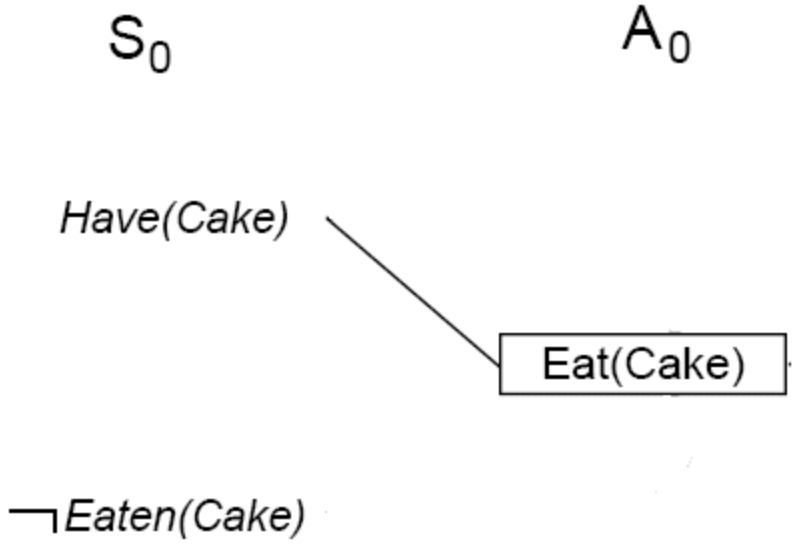
Have(Cake)

¬*Eaten(Cake)*

Create level 0 from initial problem state.

$$Init(Have(Cake))$$
$$Goal(Have(Cake) \land Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\text{PRECOND: } Have(Cake)$$
$$\text{EFFECT: } \neg Have(Cake) \land Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\text{PRECOND: } \neg Have(Cake)$$
$$\text{EFFECT: } Have(Cake))$$

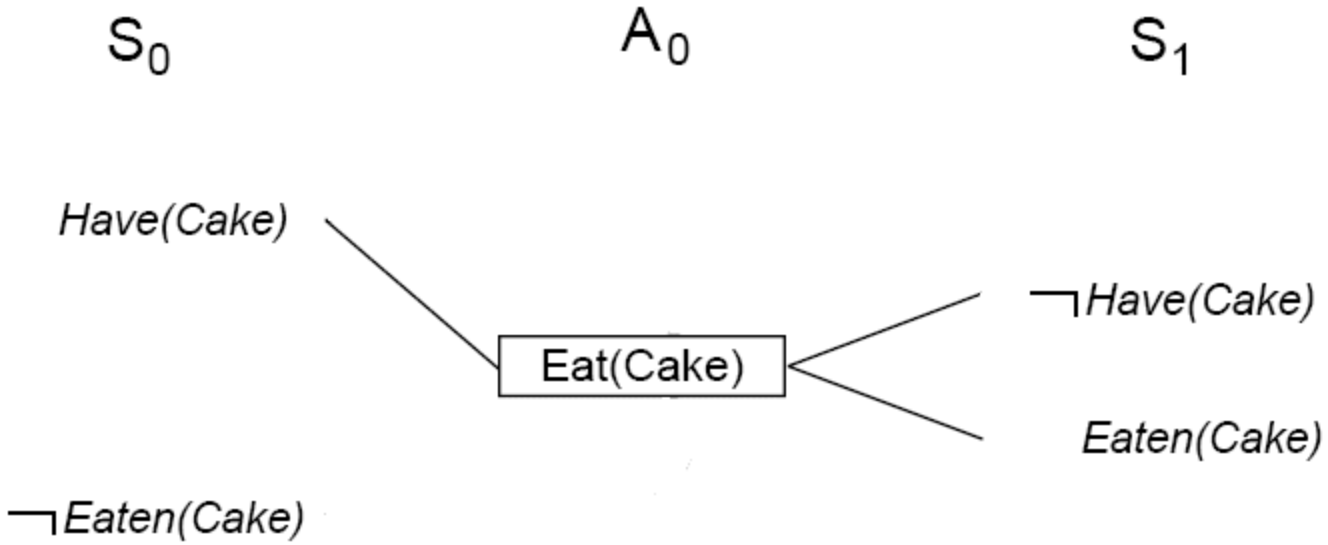
Planning Graph Example



Add all applicable actions.

$$Init(Have(Cake))$$
$$Goal(Have(Cake) \land Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\text{PRECOND: } Have(Cake)$$
$$\text{EFFECT: } \neg Have(Cake) \land Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\text{PRECOND: } \neg Have(Cake)$$
$$\text{EFFECT: } Have(Cake))$$

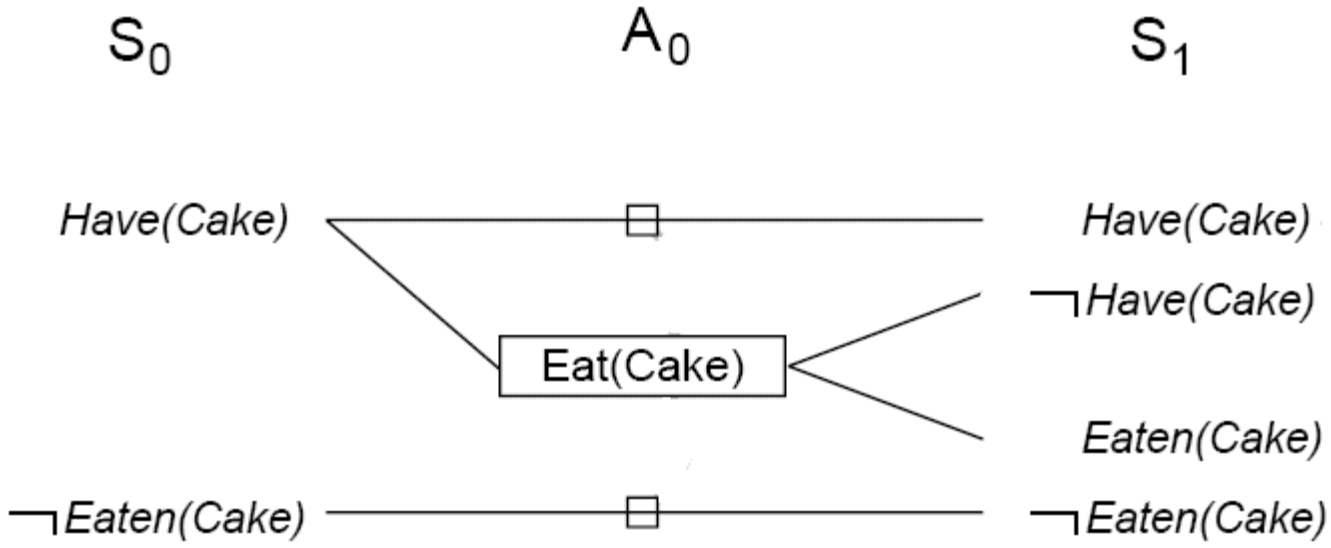
Planning Graph Example



Add all effects to the next state.

$$Init(Have(Cake))$$
$$Goal(Have(Cake) \wedge Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\text{PRECOND: } Have(Cake)$$
$$\text{EFFECT: } \neg Have(Cake) \wedge Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\text{PRECOND: } \neg Have(Cake)$$
$$\text{EFFECT: } Have(Cake))$$

Planning Graph Example



Add *persistence actions* (inaction = no-ops) to map all literals in state S_i to state S_{i+1}.

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake))$

Planning Graph Example

Identify mutual exclusions between actions and literals based on potential conflicts.

Planning Graph Example

Mutex

- A mutex between two actions indicates that it is impossible to perform these actions in parallel.

- A mutex between two literals indicates that it is impossible to have these both literals true at this stage.

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake))$

Planning Graph Example

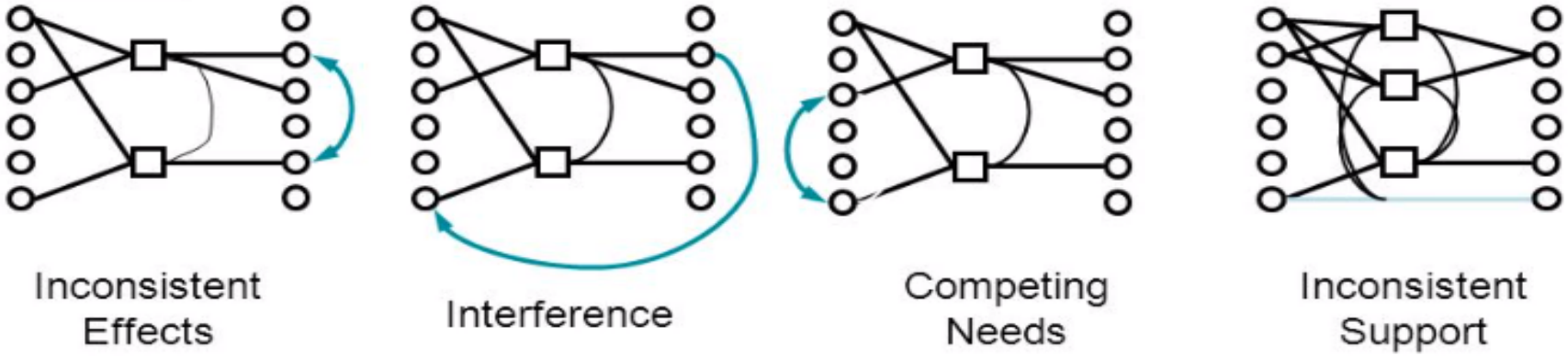
A mutex relation holds between two actions when:

- Inconsistent effects: one action negates the effect of another.
- Interference: one of the effects of one action is the negation of a precondition of the other.
- Competing needs: one of the preconditions of one action is mutually exclusive with the precondition of the other.

A mutex relation holds between two literals when:

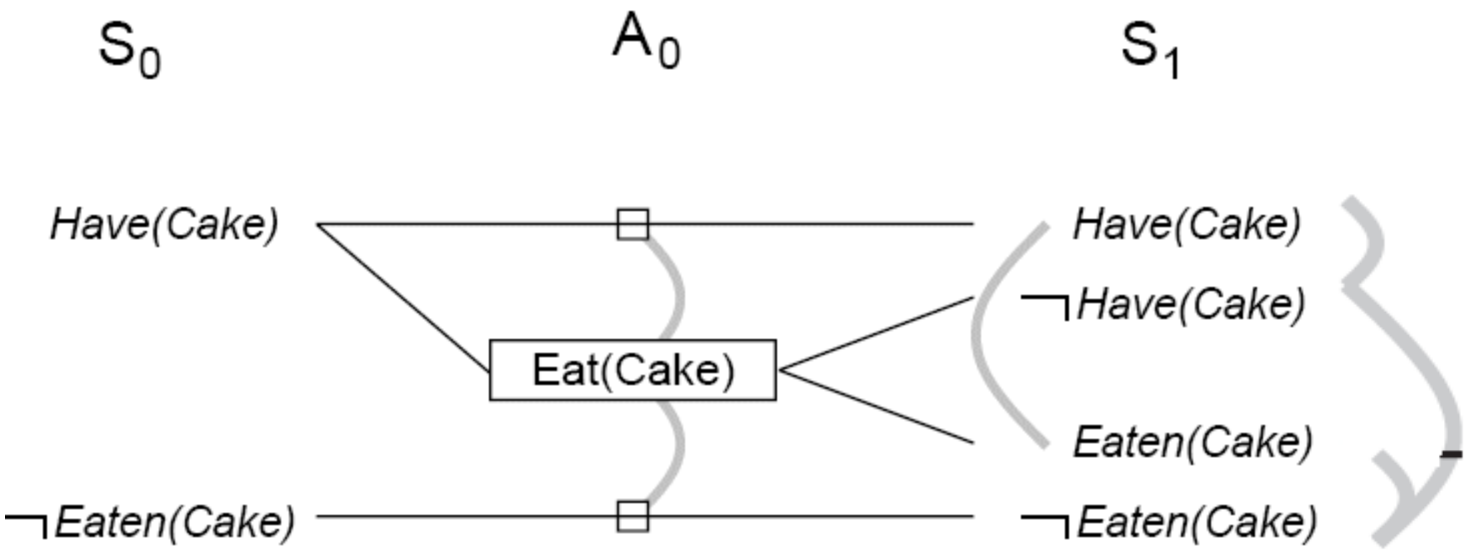
- One is the negation of the other
- Each possible action pair that could achieve the literals is mutex (inconsistent support).

Planning Graph Example



- Two actions are mutex if
  - Inconsistent effects: an effect of one negates an effect of the other
  - Interference: one effect deletes a precondition of the other
  - Competing needs: they have mutually exclusive preconditions
- Two literals are mutex if
  - Inconsistent support: one is the negation of the other, or all ways of achieving them are pairwise mutex

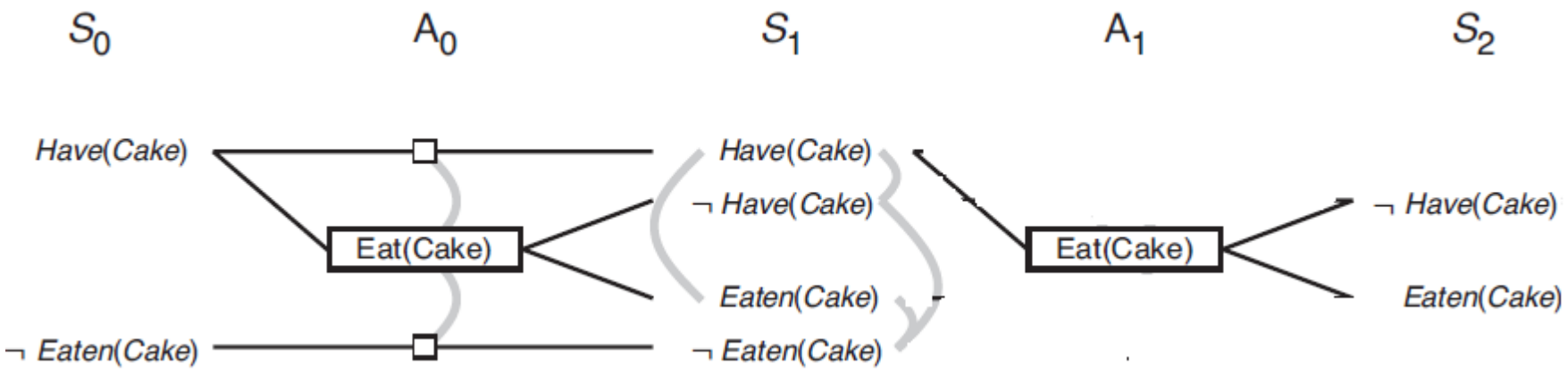
Planning Graph Example



- Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.

$$Init(Have(Cake))$$
$$Goal(Have(Cake) \wedge Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\text{PRECOND: } Have(Cake)$$
$$\text{EFFECT: } \neg Have(Cake) \wedge Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\text{PRECOND: } \neg Have(Cake)$$
$$\text{EFFECT: } Have(Cake))$$

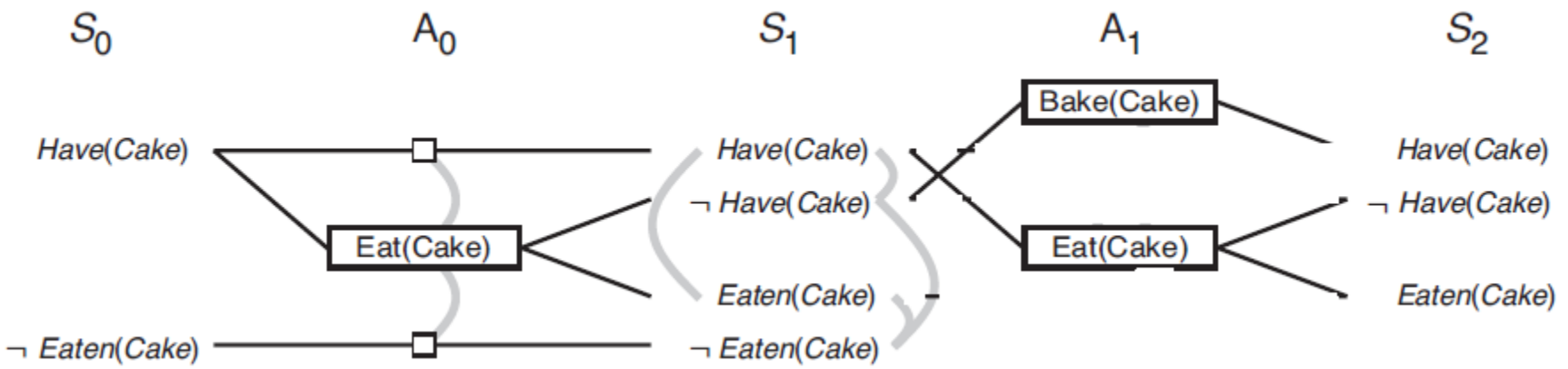
Planning Graph Example



$$Init(Have(Cake))$$
$$Goal(Have(Cake) \land Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\text{PRECOND: } Have(Cake)$$
$$\text{EFFECT: } \neg Have(Cake) \land Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\text{PRECOND: } \neg Have(Cake)$$
$$\text{EFFECT: } Have(Cake))$$

RAMAIAH
Institute of Technology

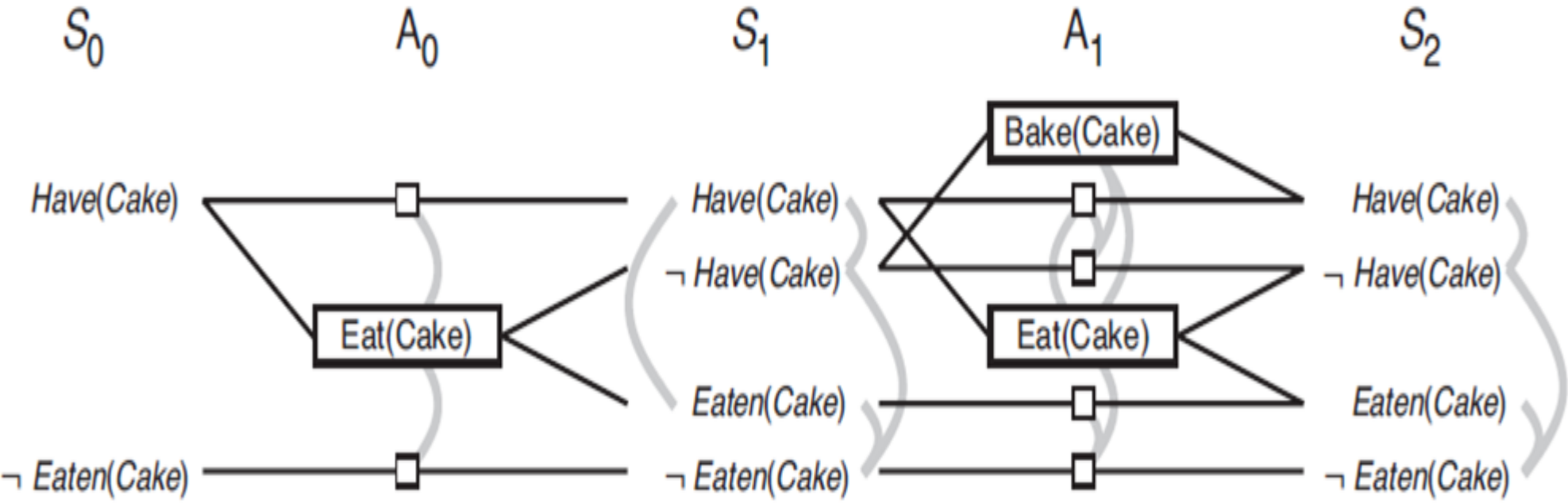
Planning Graph Example



$$Init(Have(Cake))$$
$$Goal(Have(Cake) \land Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\text{PRECOND: } Have(Cake)$$
$$\text{EFFECT: } \neg Have(Cake) \land Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\text{PRECOND: } \neg Have(Cake)$$
$$\text{EFFECT: } Have(Cake))$$

RAMAIAH Institute of Technology

Planning Graph Example



Repeat process until graph levels off:
Two consecutive levels are identical, or
Contain the same amount of literals

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake))$

Planning Graph Example

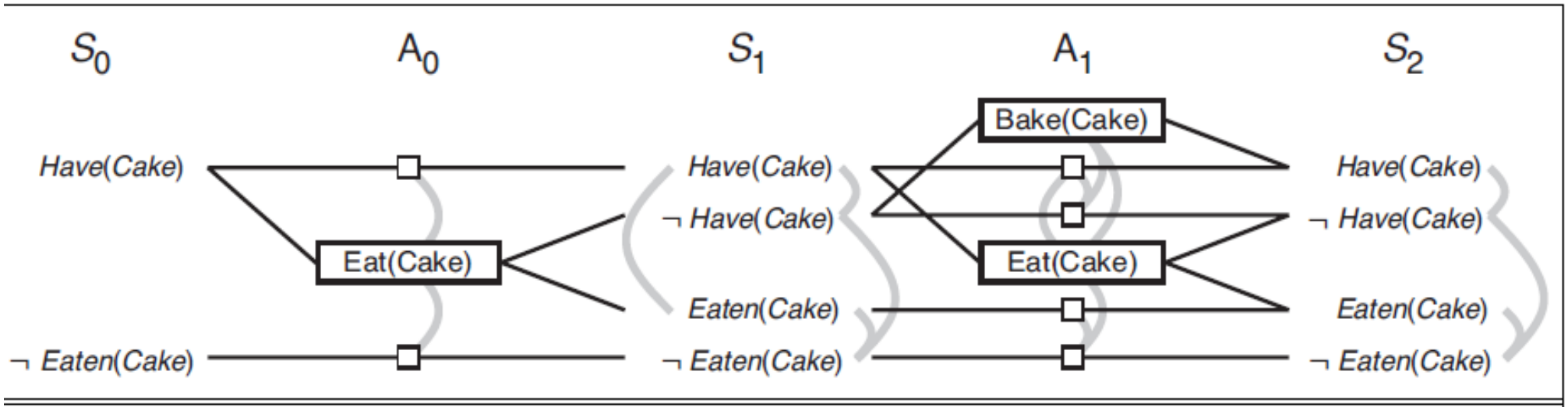


Figure 10.8 The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions), and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines. Not all mutex links are shown, because the graph would be too cluttered. In general, if two literals are mutex at S_i, then the persistence actions for those literals will be mutex at A_i and we need not draw that mutex link.

Defining Heuristics

Planning graph for heuristic search

- Using the planning graph to estimate the number of actions to reach a goal
- If a literal does not appear in the planning graph, then there is no plan that achieve this literal

$h = \infty$

Possible heuristics

- max-level: take the maximum level where any literal of the goal first appears
  -admissible

- level-sum: take the sum of the levels where any literal of the goal first appears
  -not admissible, but generally efficient

- set-level: take the minimum level where all the literals of the goal appear and are free of mutex
  -admissible

Using Planning Graphs

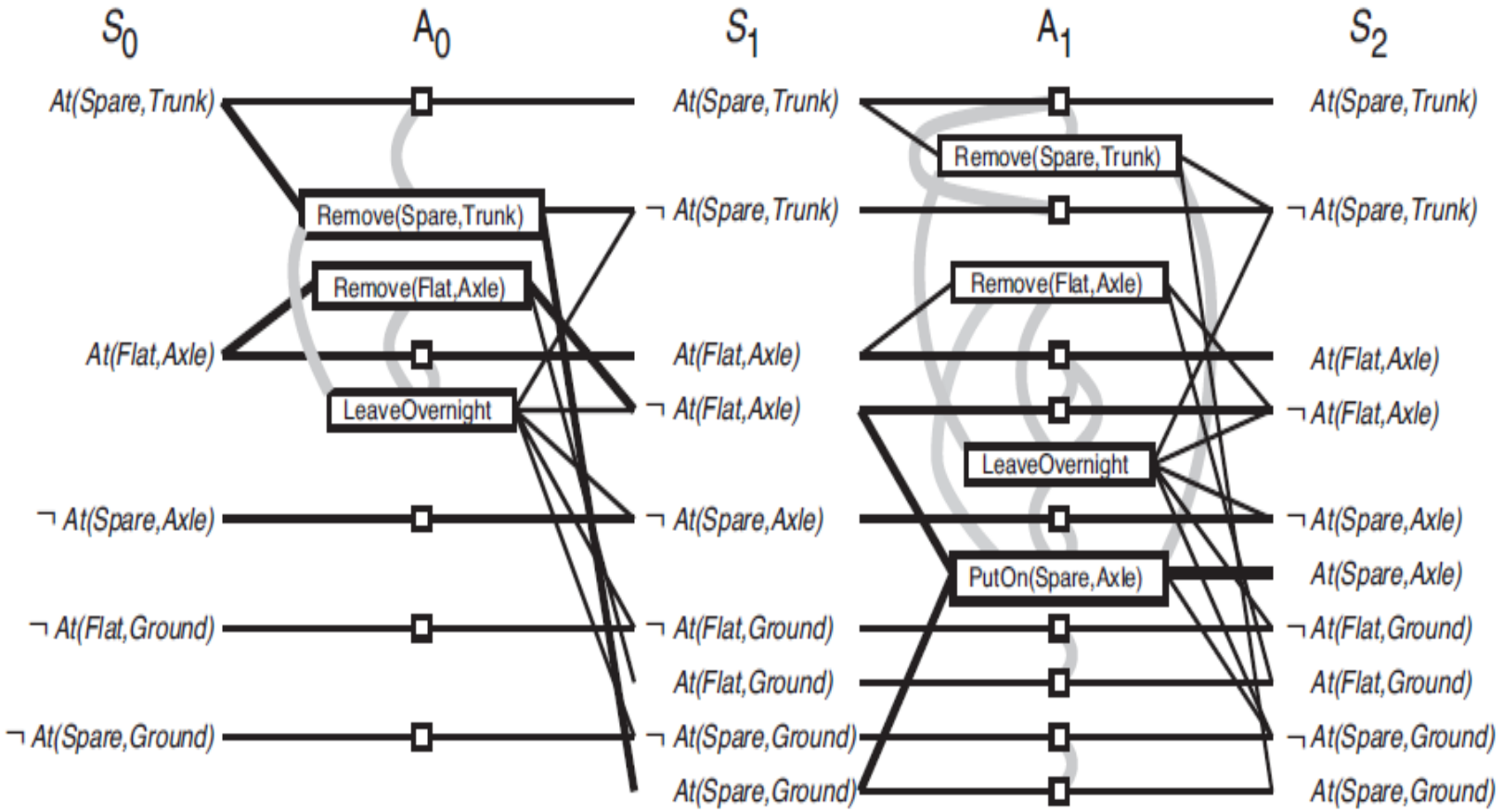
Extract a solution directly from the Planning Graph

function GRAPHPLAN(*problem*) **returns** solution or failure

graph ← INITIAL-PLANNING-GRAPH(*problem*)
goals ← CONJUNCTS(*problem*.GOAL)
nogoods ← an empty hash table
for $tl = 0$ **to** ∞ **do**
  if *goals* all non-mutex in S_t of *graph* **then**
    solution ← EXTRACT-SOLUTION(*graph*, *goals*, NUMLEVELS(*graph*), *nogoods*)
    if *solution* $\neq$ *failure* **then return** *solution*
  if *graph* and *nogoods* have both leveled off **then return** *failure*
  graph ← EXPAND-GRAPH(*graph*, *problem*)

Figure 10.9 The GRAPHPLAN algorithm. GRAPHPLAN calls EXPAND-GRAPH to add a level until either a solution is found by EXTRACT-SOLUTION, or no solution is possible.

Example: Planning graph for the spare tire problem



$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
PRECOND: $At(Spare, Trunk)$
EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
PRECOND: $At(Flat, Axle)$
EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

Termination of Graph Plan

PG are monotonically increasing or decreasing:

- Literals increase monotonically
- Actions increase monotonically
- Mutexes decrease monotonically

Because of these properties and because there is a finite number of actions and literals, every PG will eventually level off

RAMAIAH
Institute of Technology

OUTLINE

Uncertainty

- Acting under Uncertainty
- Basic Probability Notations
- Inference using Full Joint Distributions
- Independence
- Bayes rule and its Use

Learning From Examples

Introduction

KR Language	Ontological Commitment	Epistemological Commitment
Propositional Logic	facts	true, false, unknown
First Order Logic	facts, objects, relations	true, false, unknown
Temporal Logic	facts, objects, relations, times	true, false, unknown
Probability Theory	facts	degree of belief

Acting Under Uncertainty

Uncertainty is a situation where information available to decision makers is imprecise to be summarized by a probabilistic nature.

An agent may never know for certain what state it's in or where it will end up after a sequence of actions.

The different causes of uncertainty are,

1. The environment is **not fully observable.**
2. The environment behaves in a **non-deterministic** way.
3. The **actions** performed may **not have desired effect**.
4. There might be **unjustified reliance on assumptions**.

Acting Under Uncertainty

The different sources of uncertainty are:

Uncertain Data : It includes missing, noisy, inconsistent, ambiguous, unreliable data.

Uncertain Knowledge : It contains an incomplete knowledge of the domain.

Uncertain Knowledge Representation : It is nothing but representation providing restrictive model of the real system, data with imprecise representation & limited expressiveness in representation mechanism.

Inference process : The derived inference might be formally correct but wrong in the real world. The new conclusions may not be well founded

Acting Under Uncertainty

Why do we need to consider uncertainty?

- Agents never have access to the whole truth about the environment. Hence they must act under uncertainty.

Consider the problem

Example for Uncertainity in AI: Automated taxi has the goal of delivering a passenger to the airport on time.

The agent forms a plan, that involves leaving home 90 minutes before the flight departs and driving at a reasonable speed.

Even though the airport is only about 5 miles away, a logical taxi agent will not be able to conclude with certainty that the plan will definitely work.

Instead, it reaches the weaker conclusion that plan will work as long as the car doesn't break down or run out of gas, and I don't get into an accident, and there are no accidents on the bridge, and the plane doesn't leave early, and no meteorite hits the car, and . . . .".

Thus, None of the actions can be deduced for certain

Consider the problem as follows

Let action A_t = leave for airport t minutes before flight .

Will A_t get me there on time?

Problems to consider :

- partial observability (road state, other drivers' plans, etc.)
- noisy sensors (traffic reports)
- uncertainty in action outcomes (flat tire, etc.)
- immense complexity of modeling and predicting traffic

Solution

Performance Measures :

- Getting to the airport in time.
- Avoiding long, unproductive wait at the airport.
- Avoid speeding tickets along the way (Must drive below the speed limit).

Based on this assume that A_{90} is expected to maximize the agent's performance measure.

Hence a purely logical approach either

1. risks falsehood: "A_{90} will get me there on time by driving at a reasonable speed ",

or

2. leads to conclusions that are too weak for decision making :

Solution

"A_{90} will get me there on time if there's no accident on the bridge and it doesn't rain and my tires remain intact etc etc.".

You could consider the solution A_{180} Reach the airport on time with a probability of 0.999 .

but I'd have to stay overnight in the airport …

*The right thing to do—the **rational decision**—therefore depends on both the relative importance of various goals and the likelihood that, and degree to which, they will be achieved.*

Summarizing uncertainty

Diagnosis—whether for medicine, automobile repair, or whatever—almost always involves uncertainty.

Consider an example of uncertain reasoning:

Diagnosing a dental patient's toothache.

Summarizing uncertainty

We consider a dental diagnosis using FOL.

Let us now consider the rule ,

$\forall p$ Symptom(p, Toothache) => Disease(p, Cavity)

This is wrong as there can be infinite other reasons why the patient has toothache such as ,swelling, gum disease , etc.

Handling Uncertain Knowledge

Consider the following simple rule: Toothache $\Rightarrow$ Cavity

The problem is that this rule is wrong.

Not all patients with toothaches have cavities; some of them have gum disease, an abscess, or one of several other problems:

Toothache $\Rightarrow$ Cavity $\vee$ GumProblem $\vee$ Abscess ...

Unfortunately, in order to make the rule true, we have to add an almost unlimited list of possible problems.

We could try turning the rule into a causal rule: Cavity $\Rightarrow$ Toothache .

But this rule is not right either; not all cavities cause pain.

Hence using FOL fails because : (with a domain like medical diagnosis thus fails for three main reasons)

Laziness: It is too much work to list the complete set of antecedents or consequents needed to ensure an exceptionless rule and too hard to use such rules.

Theoretical ignorance: Medical science has no complete theory for the domain.

Practical ignorance: Even if we know all the rules, we might be uncertain about a particular patient because not all the necessary tests have been or can be run.

Summarizing uncertainty

The agent's knowledge at the best can provide a degree of belief using probability (0 to 1).

Probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance.
Eg : P(0.8) = 80% *chance* that a patient has cavity knowing he/she has toothache.

NOTE : Degree of belief != Degree of truth (fuzzy logic)

80% degree of belief is a fairly strong expectation & 80% truth implies 80 out of 100 are true

Making rational decisions under uncertainty

Suppose I believe the following:

a)$P(A_{25}$ gets me there on time | …) = 0.04
b)$P(A_{90}$ gets me there on time | …) = 0.70
c) $P(A_{120}$ gets me there on time | …) = 0.95
d)$P(A_{1440}$ gets me there on time | …) = 0.9999

Which action to choose?

Depends on my preferences for missing flight vs. time spent waiting, etc.

- Utility theory is used to represent and infer preferences
- **Decision theory = probability theory + utility theory**

Maximum Expected Utility (MEU)

This principle states that an agent is rational if and only if it chooses an action that yields highest expected utility averaged over all possible outcomes of the action.

Making rational decisions under uncertainty

function DT-AGENT(*percept*) **returns** an *action*
  persistent: *belief_state*, probabilistic beliefs about the current state of the world
    action, the agent's action

  update *belief_state* based on *action* and *percept*
  calculate outcome probabilities for actions,
    given action descriptions and current *belief_state*
  select *action* with highest expected utility
    given probabilities of outcomes and utility information
  return *action*

Figure 13.1 A decision-theoretic agent that selects rational actions.

Summarizing uncertainty

For example, suppose that the agent has drawn a card from a shuffled pack.

- Before looking at the card, P(ace of spades) = 1/52. Prior/Unconditional Probability
- After looking at the card, P(ace of spades) = 0 or 1. Posterior/Conditional Probability

Thus, Based on evidence (percepts) in the KB an assignment of probability to a proposition is analogous to saying whether or not a given logical sentence (or its negation) is entailed by the KB, rather than whether or not it is true.

Basic Probability Notation

In probability theory, the set of all possible worlds is called the sample space.

The possible worlds are mutually exclusive and exhaustive

Basic Probability Notation

Example
If we are about to roll two (distinguishable) dice, there are 36 possible worlds to consider: (1,1), (1,2), ..., (6,6).

Ω (uppercase omega) – Sample space
ω (lowercase omega) - Elements of the space, that is, particular

possible worlds.

Basic Probability Notation

A fully specified probability model associates a numerical probability P(ω) with each possible world.

The basic axioms of probability theory say that every possible world has a probability between 0 and 1 and that the total probability of the set of possible worlds is 1:

$$0 \leq P(\omega) \leq 1 \text{ for every } \omega \text{ and } \sum_{\omega \in \Omega} P(\omega) = 1 \ .$$

Basic Probability Notation

Conditional Probability

Is defined as the probability of an event A, given that another event B has already occurred (i.e. A conditional B).

This is represented by P(A|B) and we can define it as:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability of *A* given *B*

Probability of *A* and *B*

Probability of *B*

Basic Probability Notation

Example:

Susan took two tests. The probability of her passing both tests is 0.6. The probability of her passing the first test is 0.8. What is the probability of her passing the second test given that she has passed the first test?

Solution:

$$P(\text{second} \mid \text{first}) = \frac{P(\text{first and second})}{P(\text{first})} = \frac{0.6}{0.8} = 0.75$$

Probability of *A* and *B*

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability of *A* given *B*

Probability of *B*

Basic Probability Notation

Mathematically - conditional probabilities are defined in terms of unconditional probabilities as follows:

For any propositions a and b, we have

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)}$$

Conditional probability can be written in a different form called the Product rule:

$$P(a \wedge b) = P(a \mid b)P(b)$$

Basic Probability Notation

Random variable

- A Random Variable is a variable that can take on different values randomly.
- Every random variable has a domain—the set of possible values it can take on.
- Variables can have infinite domains too either discrete (like the integers) or continuous (like the reals).
- Their names begin with an uppercase letter

Example :

Die1 is random variable.

Domain of Die1 is {1,..., 6}

Basic Probability Notation

1. **Boolean Random Variables**

   Eg., Cavity (do I have a cavity?) It can take values : <true,false>

2. **Discrete Random Variables**

   Eg., Weather is one of <sunny,rainy,cloudy,snow>

3. **Continuous Random Variables**

   Eg., Height of different people in a given population set

Basic Probability Notation

Probability Distribution Function

P statement defines a probability distribution for the random variable .

The P notation is also used for conditional distributions:

P(X | Y) gives the values of $P(X = x_i \mid Y = y_j)$ for each possible i, j pair.

Basic Probability Notation

Example : Domain of Weather

Probability Distribution for the random variable Weather .

P indicates that the result is a vector of numbers, and where we assume a predefined ordering <sunny, rain, cloudy, snow>

$$\mathbf{P}(Weather) = \langle 0.6, 0.1, 0.29, 0.01 \rangle$$

$$P(Weather = sunny) = 0.6$$
$$P(Weather = rain) = 0.1$$
$$P(Weather = cloudy) = 0.29$$
$$P(Weather = snow) = 0.01 ,$$

Basic Probability Notation

Probability Density Function (pdf)

For continuous variables, it is not possible to write out the entire distribution as a vector, because there are infinitely many values.

Instead, we can define the probability that a random variable takes on some value x as a parameterized function of x.

We write the probability density for a continuous random variable X at value x as

P(X = x) or just P(x);

$$P(NoonTemp = x) = Uniform_{[18C,26C]}(x)$$

Basic Probability Notation

$$P(a \wedge b) = P(a \mid b)P(b)$$

Full Joint Probability Distribution

We need notation for distributions on multiple variables

For example

Weather = { sunny, rain, cloud, snow}

Cavity = { cavity, ¬ cavity }

$$P(W = sunny \wedge C = true) = P(W = sunny|C = true)\, P(C = true)$$
$$P(W = rain \wedge C = true) = P(W = rain|C = true)\, P(C = true)$$
$$P(W = cloudy \wedge C = true) = P(W = cloudy|C = true)\, P(C = true)$$
$$P(W = snow \wedge C = true) = P(W = snow|C = true)\, P(C = true)$$
$$P(W = sunny \wedge C = false) = P(W = sunny|C = false)\, P(C = false)$$
$$P(W = rain \wedge C = false) = P(W = rain|C = false)\, P(C = false)$$
$$P(W = cloudy \wedge C = false) = P(W = cloudy|C = false)\, P(C = false)$$
$$P(W = snow \wedge C = false) = P(W = snow|C = false)\, P(C = false)\,.$$

P(Weather , Cavity) denotes the probabilities of all combinations of the values of Weather and Cavity.

This is a 4 × 2 table of probabilities called the joint probability distribution of Weather and Cavity.

Inference using Full Joint Distributions

How to infer a new fact in case of Uncertainity

Consider the Example :

A domain consisting of just the three Boolean variables

- Toothache
- Cavity
- Catch (the dentist's nasty steel probe catches in my tooth)

Inference using Full Joint Distributions

The full joint distribution is a 2 × 2 × 2 table as shown in Figure

	toothache		¬*toothache*	
	catch	¬*catch*	*catch*	¬*catch*
cavity	0.108	0.012	0.072	0.008
¬*cavity*	0.016	0.064	0.144	0.576

Figure 13.3 A full joint distribution for the *Toothache*, *Cavity*, *Catch* world.

Inference using Full Joint Distributions

Marginalization or summing out

marginalization rule for any sets of variables **Y** and **Z**:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z} \in \mathbf{Z}} \mathbf{P}(\mathbf{Y}, \mathbf{z})$$

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

where $\sum_{\mathbf{z} \in \mathbf{Z}}$ means to sum over all the possible combinations of values of the set of variables **Z**.

We sometimes abbreviate this as $\sum_{\mathbf{z}}$, leaving **Z** implicit. We just used the rule as

$$\mathbf{P}(Cavity) = \sum_{\mathbf{z} \in \{Catch, Toothache\}} \mathbf{P}(Cavity, \mathbf{z}) \ .$$

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

Find the Probability that there is cavity

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

$$P(cavity) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$$

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

Find the Probability P(Cavity v Toothache)

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

P(Cavity v Toothache) = 0.108 +0.012+0.072+0.008 +0.016 +0.064 = 0.28

Inference using Full Joint Distributions

Conditioning

A variant of this rule involves conditional probabilities instead of joint probabilities, using the product rule:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} \mid \mathbf{z}) P(\mathbf{z}) \ .$$

$$P(a \wedge b) = P(a \mid b) P(b)$$

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

Find the Probability of a cavity, given evidence of a toothache $P(cavity \mid toothache)$

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

Probability of A and B

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability of A given B — Probability of B

$$P(cavity \mid toothache) = \frac{P(cavity \wedge toothache)}{P(toothache)}$$

$$= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6$$

0.12 /0.2 =0.6

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

Find the Probability that there is no cavity, given a toothache $P(\neg cavity \mid toothache)$

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability of A given B | Probability of A and B | Probability of B

$$P(\neg cavity \mid toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)}$$

$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

0.08 /0.2 =0.4

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

a. $\mathbf{P}(toothache)$

P(toothache) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2

b. $\mathbf{P}(Cavity)$

c. $\mathbf{P}(Toothache \mid cavity)$

d. $\mathbf{P}(Cavity \mid toothache \vee catch)$

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

a. $\mathbf{P}(toothache)$

b. $\mathbf{P}(Cavity)$

c. $\mathbf{P}(Toothache \mid cavity)$

d. $\mathbf{P}(Cavity \mid toothache \vee catch)$

P(Cavity) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

a. $\mathbf{P}(toothache)$
b. $\mathbf{P}(Cavity)$
c. $\mathbf{P}(Toothache \mid cavity)$
d. $\mathbf{P}(Cavity \mid toothache \vee catch)$

Probability of *A* and *B*

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability of *A* given *B* — Probability of *B*

$$P(\text{Toothache} \mid \text{Cavity}) = \frac{P(\text{Toothache} \wedge \text{Cavity})}{P(\text{Cavity})}$$

$$= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.072 + 0.008} = 0.6$$

P(Cavity v Toothache) = 0.108 +0.012+0.072+0.008 +0.016 +0.064 = 0.28

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability of A given B | Probability of A and B | Probability of B

a. $\mathbf{P}(toothache)$
b. $\mathbf{P}(Cavity)$
c. $\mathbf{P}(Toothache \mid cavity)$
d. $\mathbf{P}(Cavity \mid toothache \vee catch)$

P(Cavity | toothache ∨ catch) = cavity and (toothache or catch) / (toothache or catch)

= (0.108 + 0.012 + 0.072) / 0.416

= 0.46

Inference using Full Joint Distributions

Given the full joint distribution for the toothache, cavity and catch world:

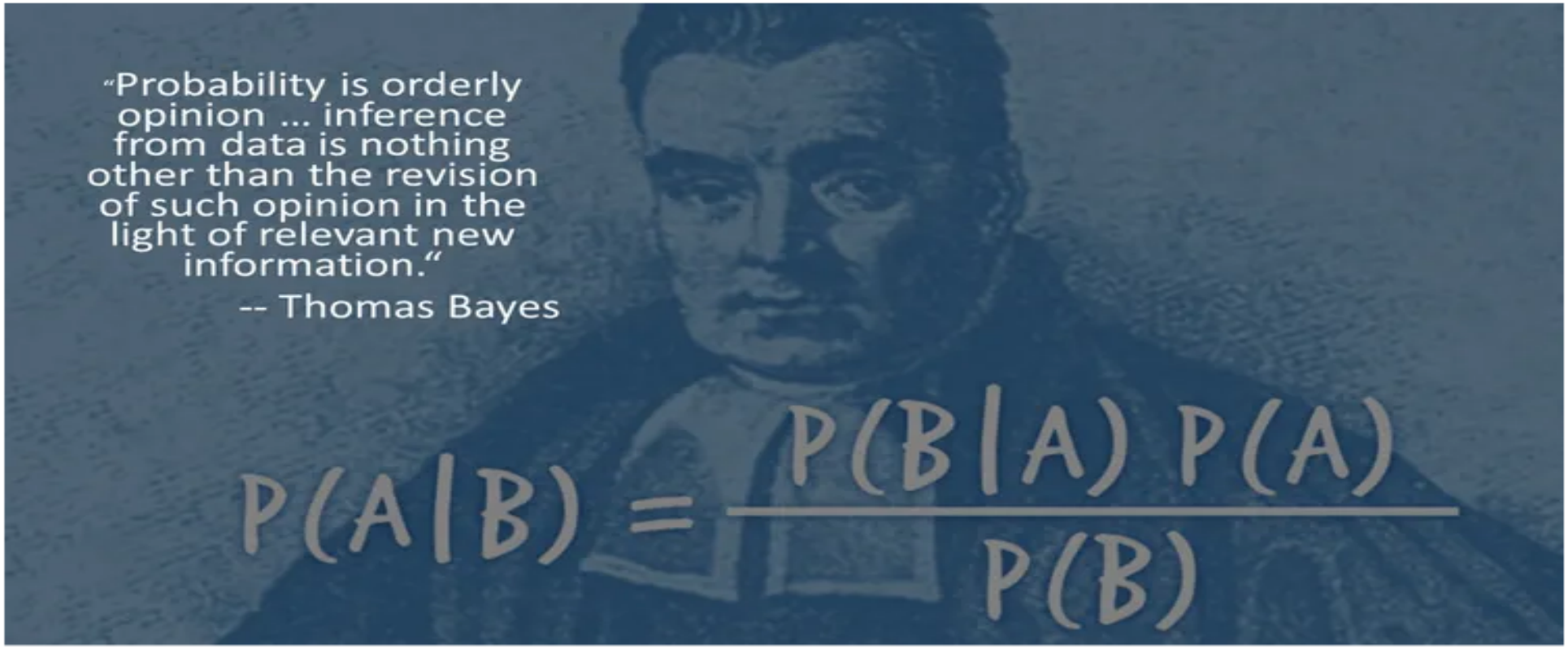
	Toothache		¬toothache	
	catch	¬catch	Catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

Calculate the following:

i) P(¬toothache)
ii) P(¬cavity)
iii) P(toothache | cavity)
iv) P (cavity | toothache V catch).

Bayes' rule

What is Bayes' Theorem?



Bayes' rule

Consider that A and B are any two events from a sample space S

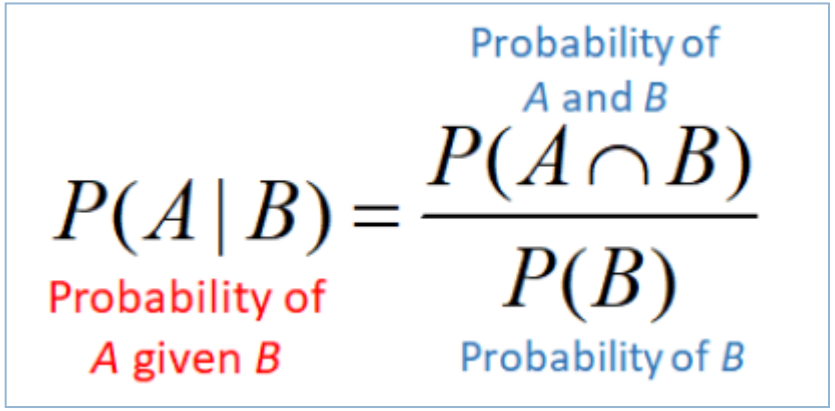
Using our understanding of conditional probability, we have:

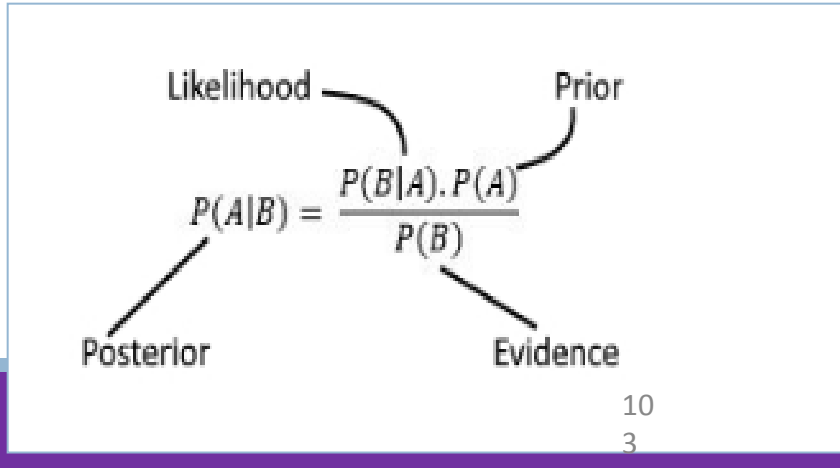
P(A|B) = P(A ∩ B) / P(B)

P(B|A) = P(A ∩ B) / P(A)

It follows that P(A ∩ B) = P(A|B) * P(B) = P(B|A) * P(A)

Thus, P(A|B) = P(B|A)*P(A) / P(B)



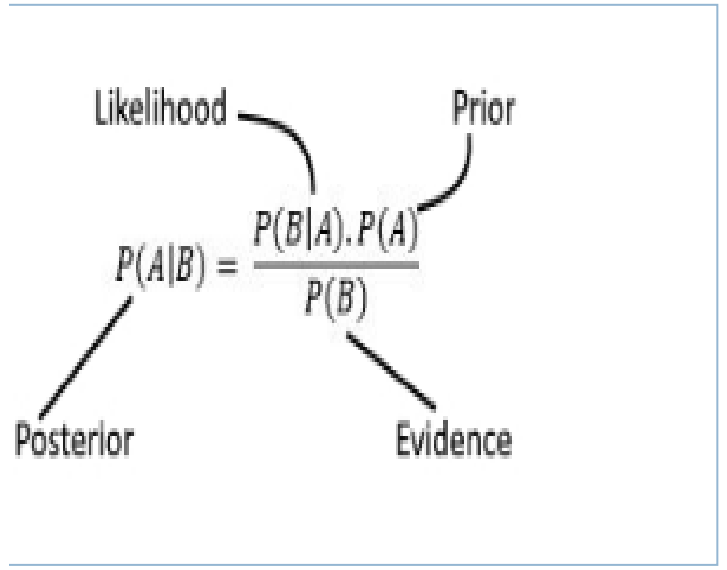


Bayes' rule

Formula for Bayes' Theorem

The Bayes' theorem is expressed in the following formula:

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



Where:

- P(A | B) – the probability of event A occurring, given event B has occurred
- P(B | A) – the probability of event B occurring, given event A has occurred
- P(A) – the probability of event A
- P(B) – the probability of event B

OUTLINE

Learning From Examples

- Forms of Learning
- Supervised Learning
- Learning Decision Trees
- Artificial Neural Networks
- Support Vector Machines
- Ensemble Learning

Introduction

Why would we want an agent to learn?

- First, the designers cannot anticipate all possible situations that the agent might find itself in.
  - For example, a robot designed to navigate mazes must learn the layout of each new maze it encounters.
- Second, the designers cannot anticipate all changes over time
- Third, sometimes human programmers have no idea how to program a solution themselves.

Forms of Learning

A Learning agent learn from its past experiences or it has learning capabilities.

Four components of an agent are

Performance element: Selects external action :Takes in percepts and decides on actions

Learning element :It is responsible for making improvements by observing performance

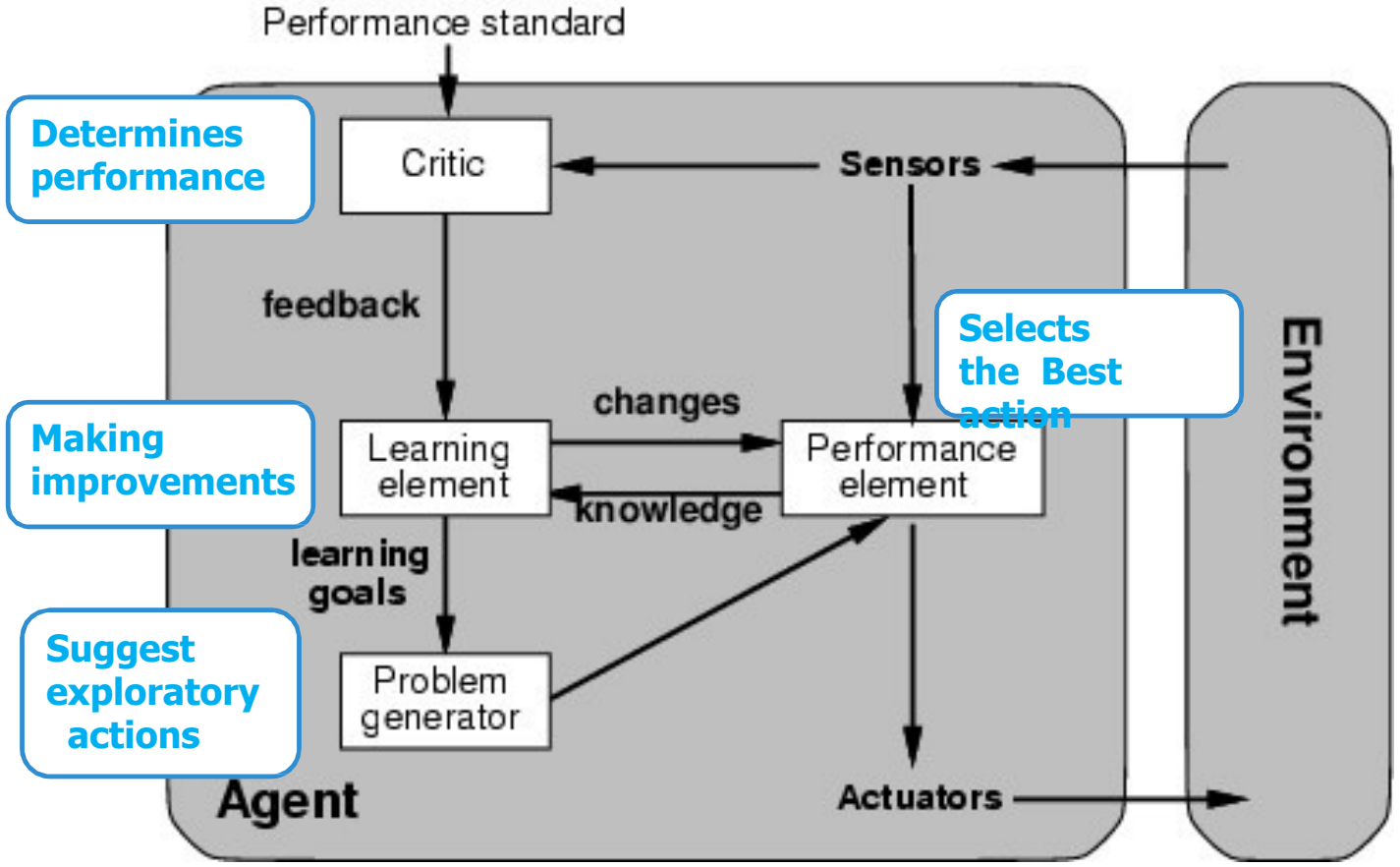
Critic: Learning element takes feedback from critic which describes how well the agent is doing with respect to a fixed performance standard.

Problem Generator: This component is responsible for suggesting actions that will lead to new and informative experiences.

Forms of Learning

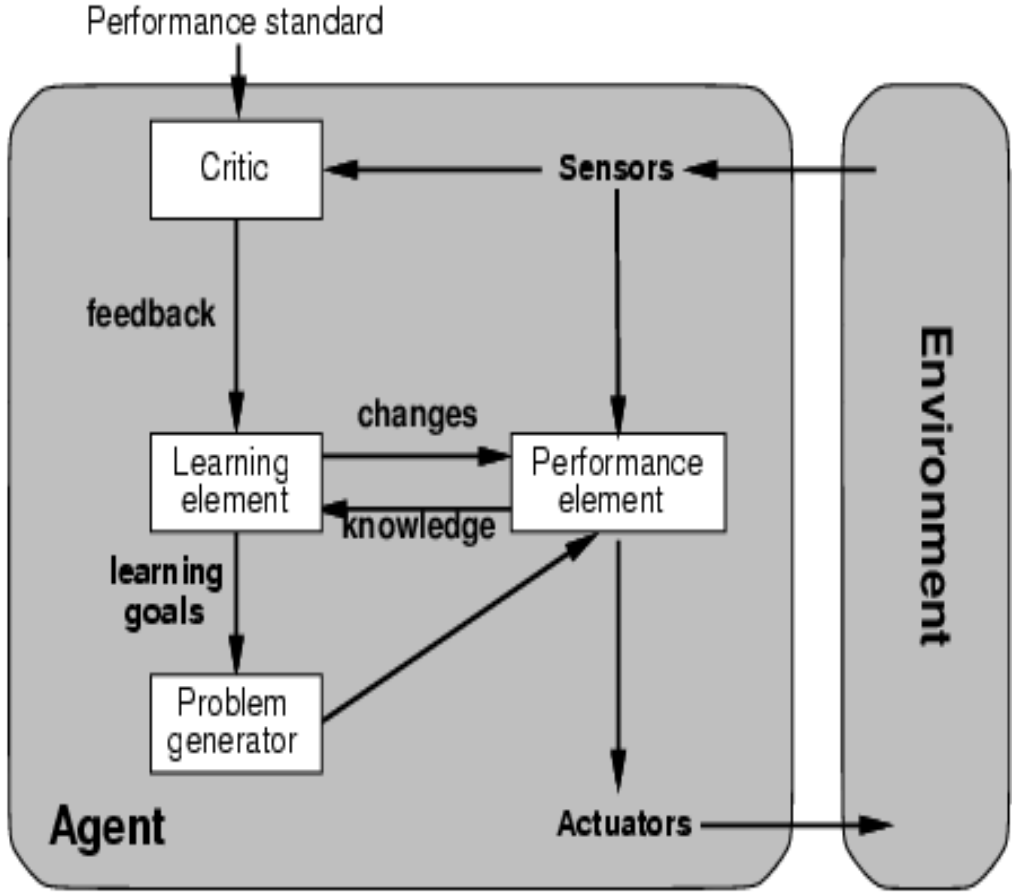
A learning agent in AI is the type of agent that can learn from its past experiences or it has learning capabilities.

It starts to act with basic knowledge and then is able to act and adapt automatically through learning.



Components Learning Agent

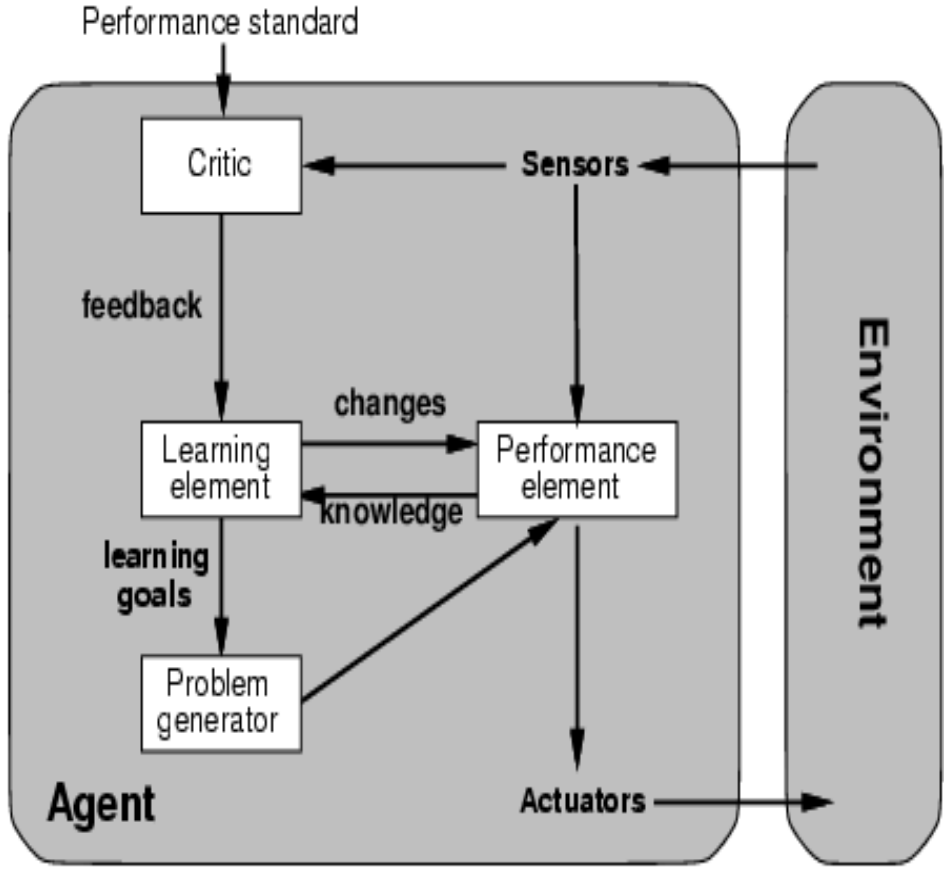
- Learning element
- Performance element
- Critic
- Problem generator



Learning Element

Responsible for making improvements.

Uses knowledge about the agent and feedback on its actions to improve performance.

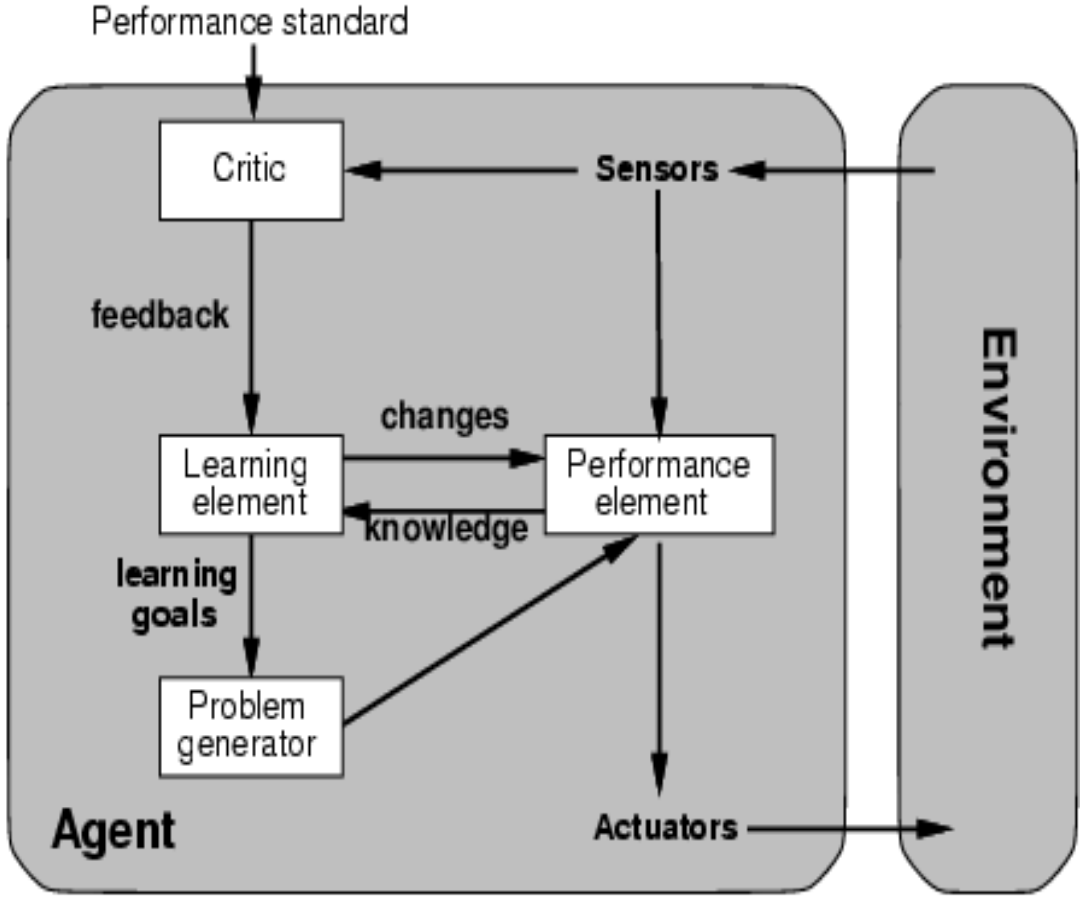


Performance Element

Selects external actions

Collects percepts, decides on actions

Incorporated most aspects of our previous agent design

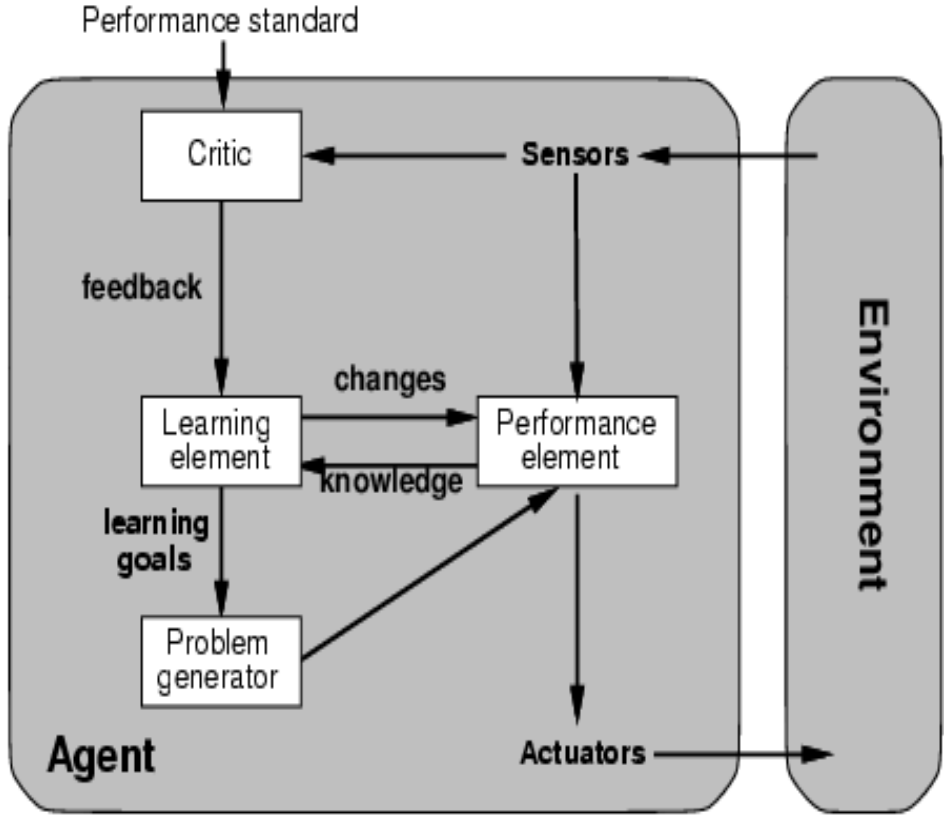


Critic

Informs the learning element about the performance of the action

must use a fixed standard of performance

- should be from the outside
- an internal standard could be modified to improve performance
  - sometimes used by humans to justify or disguise low performance



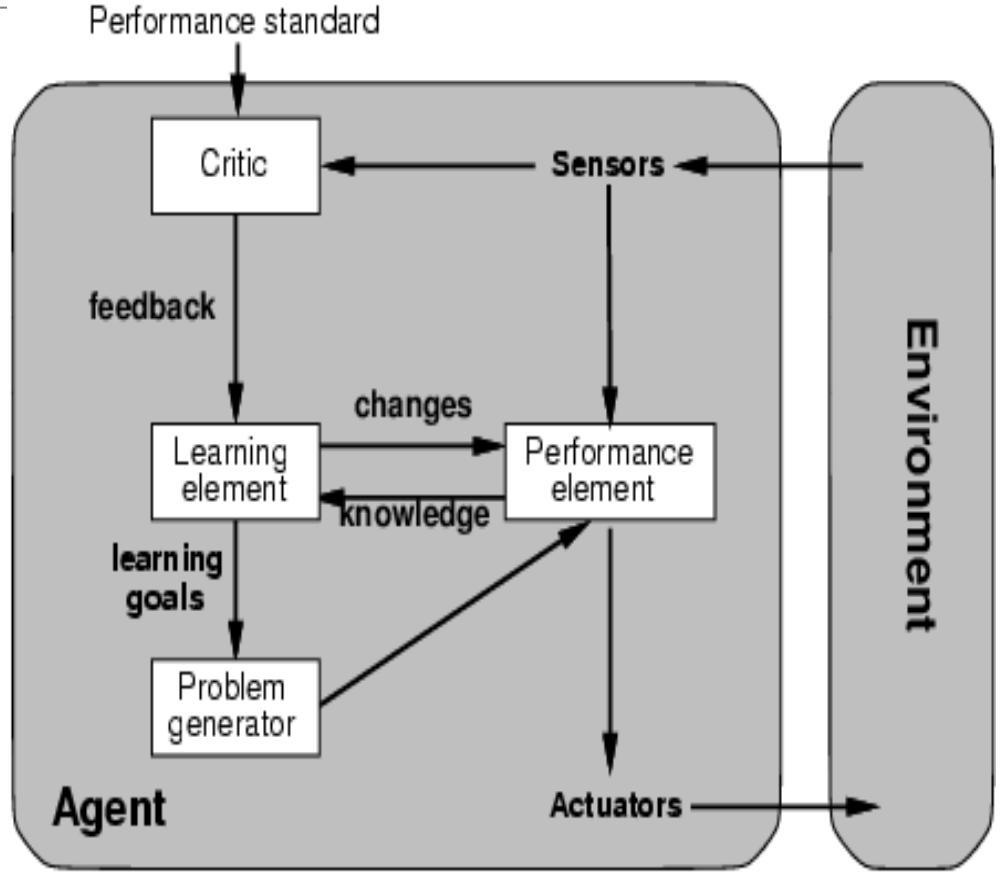
Problem Generator

Suggests actions that might lead to new experiences

may lead to some sub-optimal decisions in the short run

- in the long run, hopefully better actions may be discovered

otherwise no exploration would occur

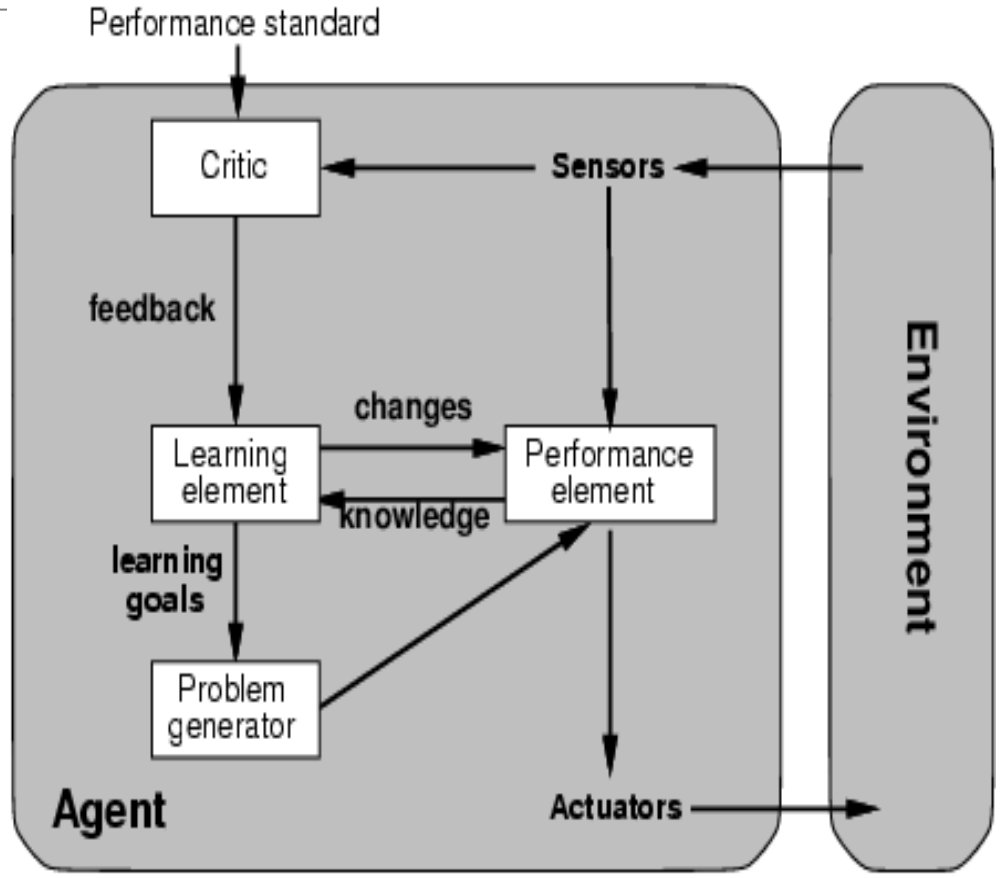


Example : Automated Taxi

Performance element : Collection of knowledge and procedures the taxi has for selecting its driving actions. The taxi goes out on the road and drives.

Critic: Observes the world and passes information along to the learning element.

For example, after the taxi makes a quick left turn across three lanes of traffic, the critic observes the shocking language used by other drivers.

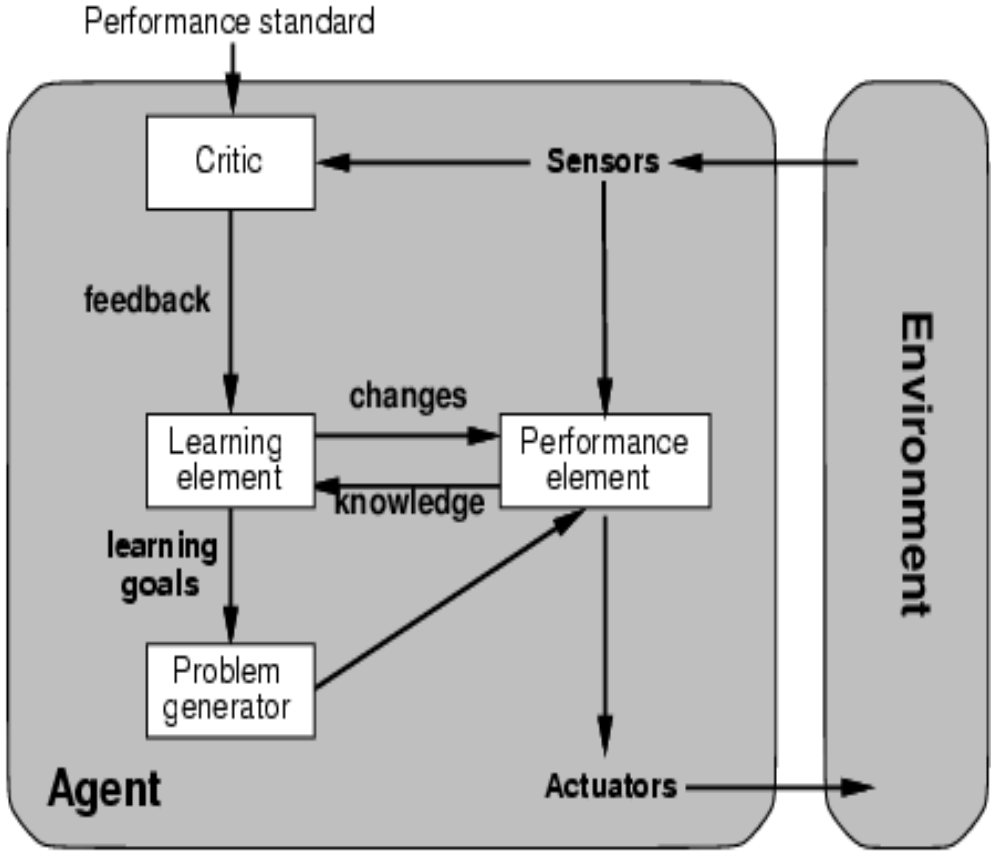


Example : Automated Taxi

Learning element : Formulate a rule saying this was a bad action, and the performance element is modified by installation of the new rule.

Problem generator: Identify certain areas of behavior in need of improvement and suggest experiments,

such as trying out the brakes on different road surfaces under different conditions.



Forms of Learning

Any component of an agent can be improved by learning from data.

The improvements depend on four major factors:

- Which component is to be improved.
- What prior knowledge the agent already has.
- What representation is used for the data and the component.
- What feedback is available to learn from

Forms of Learning

Representation and prior knowledge

- Inputs is in the form of factored representation - a vector of attribute values
- Outputs that can be either a continuous numerical value or a discrete value.

Various types of learning

Inductive learning: Learning a (possibly incorrect) general function or rule from specific input–output pairs

Deductive learning: Learning from a known general rule to a new rule that is logically entailed

Inductive

Case – Mrs. Smith takes calcium

Result – Mrs. Smith could be constipated

Rule – If a patient takes calcium, she could have constipation

Deductive

Rule – If a patient takes calcium, she could have constipation

Case – Mrs. Smith takes calcium

Result – Mrs. Smith could be constipated

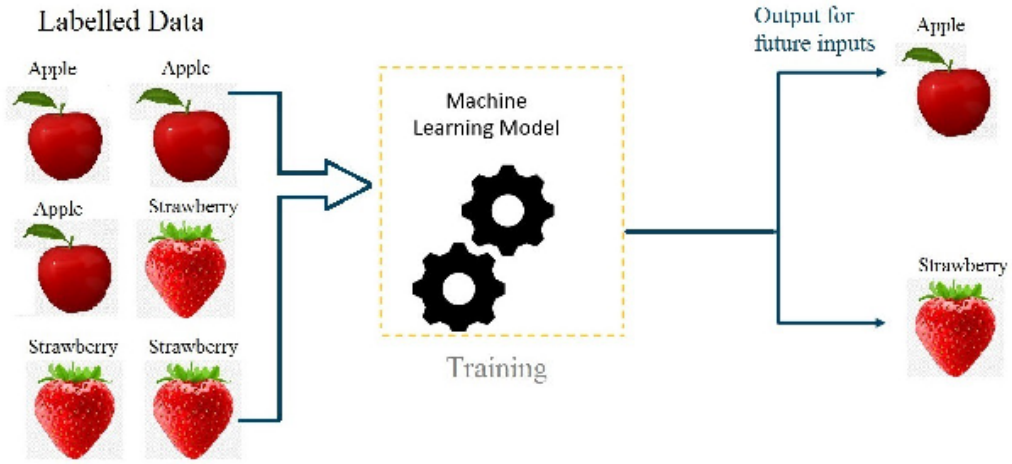
Forms of Learning

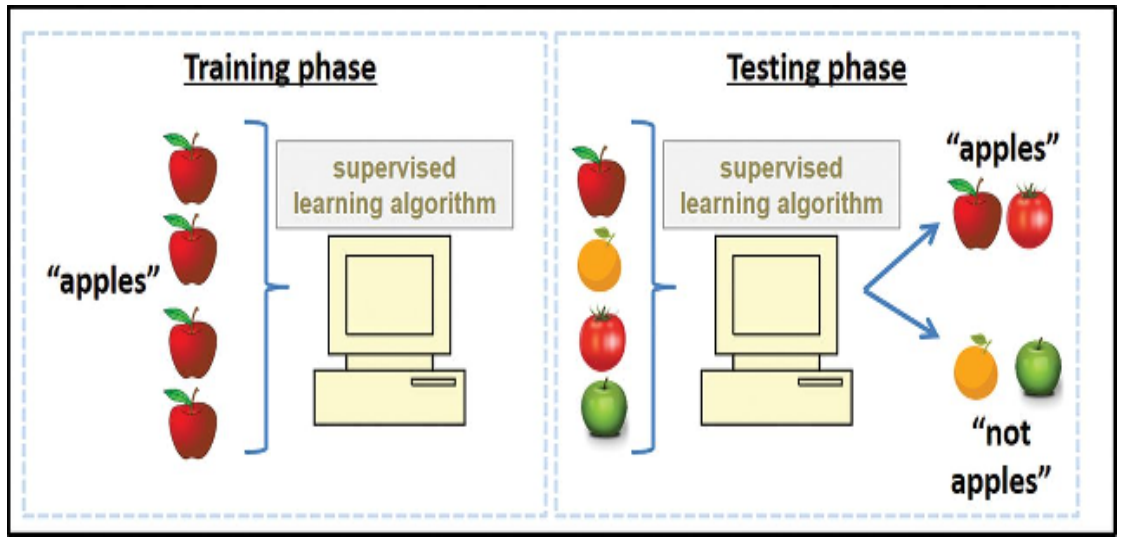
There are three types of feedback that determine the three main types of learning:

- Supervised learning - Agent observes some example input–output pairs and learns a function that maps from input to output
- Unsupervised learning - Agent learns patterns in the input even though no explicit feedback is supplied
- Reinforcement learning - Agent learns from a series of reinforcements rewards or punishments

Supervised Learning

Supervised learning, train model on a labelled dataset that means we have both raw input data as well as its results.





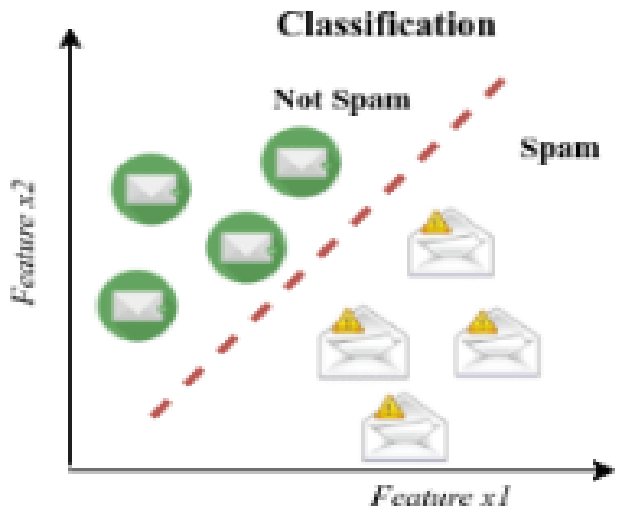
Supervised Learning

Two types of problem

Classification problems

This algorithm helps to predict a discrete value.

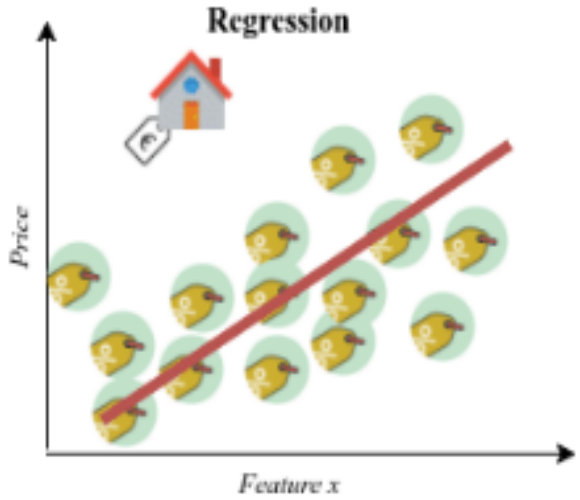
- Naive Bayes Classifier
- Support Vector Machines
- Logistic Regression



Regression problems

These problems are used for continuous data.

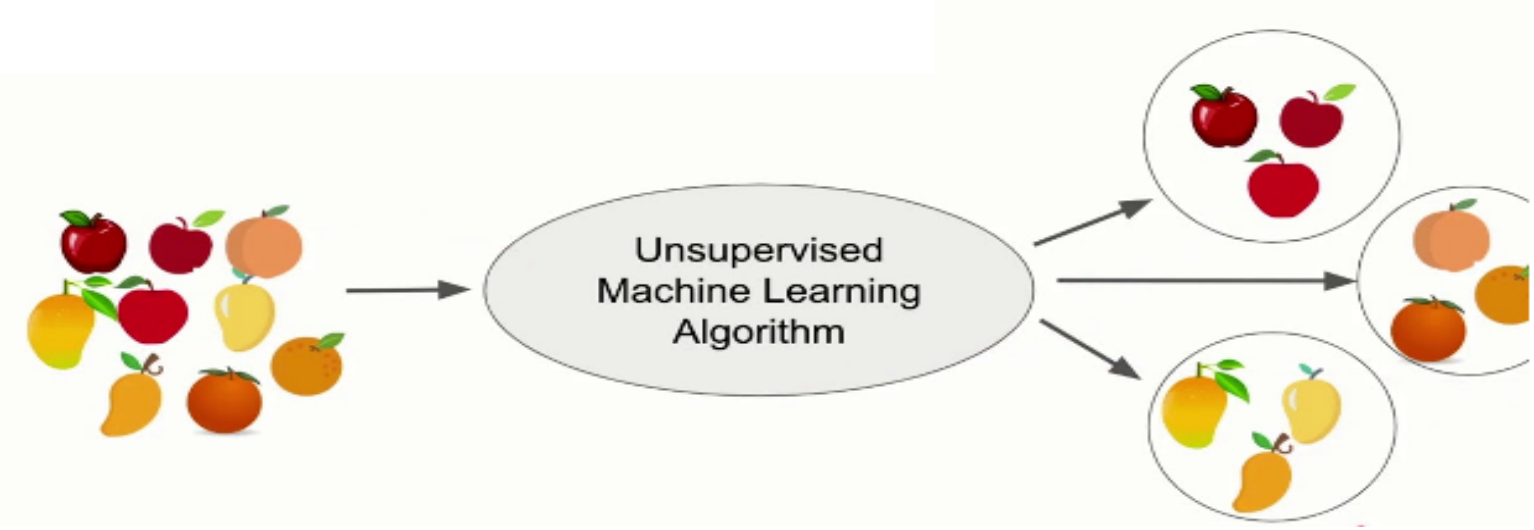
- Linear Regression
- Nonlinear Regression
- Bayesian Linear Regression



Unsupervised Learning

Unsupervised learning is self-organized learning.

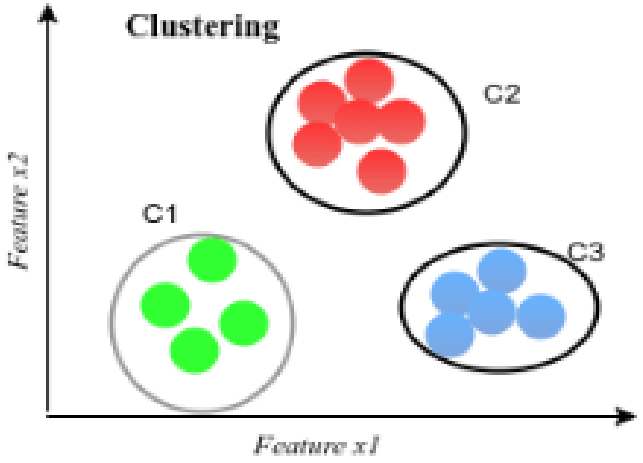
Here we basically provide the machine with data and ask to look for hidden features and cluster the data in a way that makes sense.

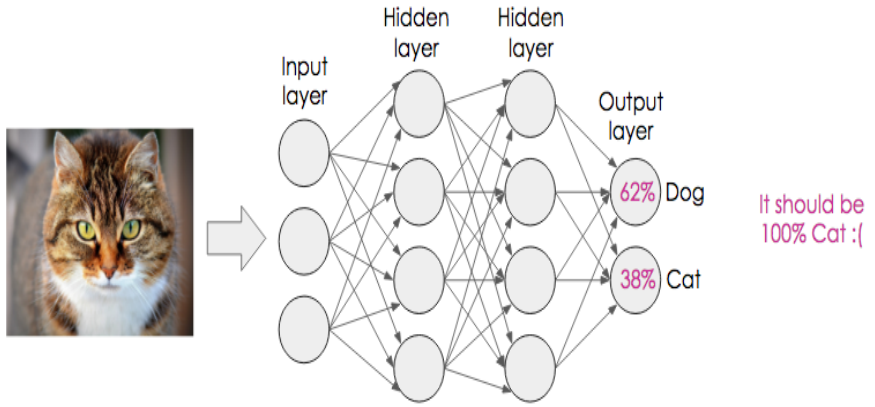


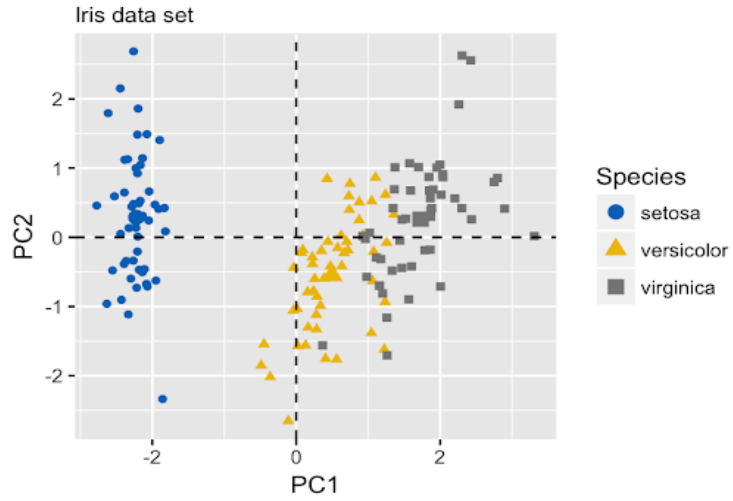
Unsupervised Learning

Algorithms available for unsupervised learning are

- K – Means clustering
- Neural Networks
- Principal Component Analysis Algorithm



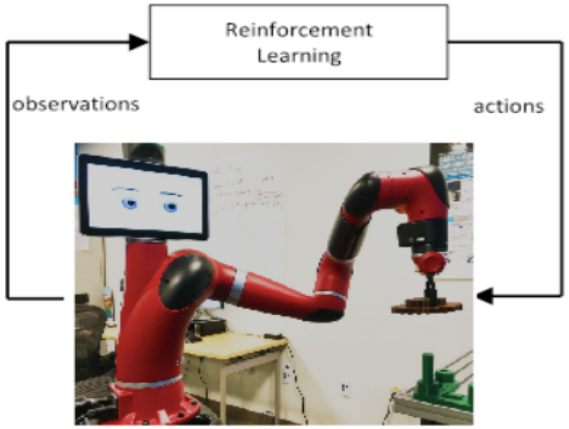


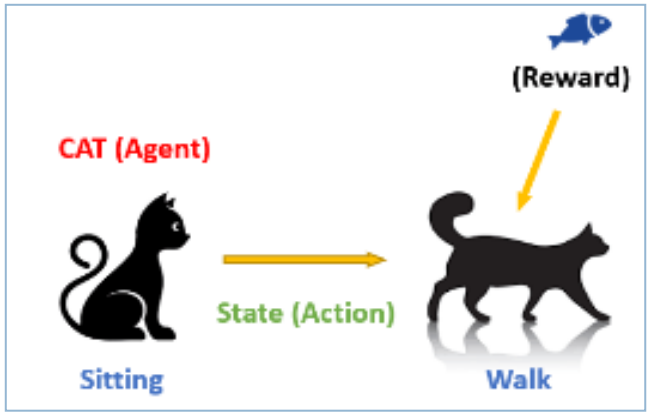


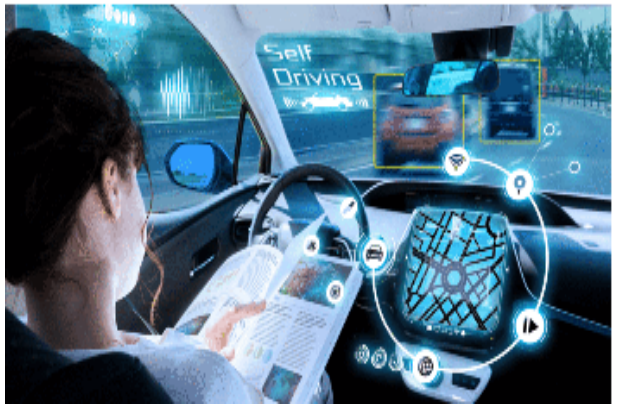
Reinforcement Learning

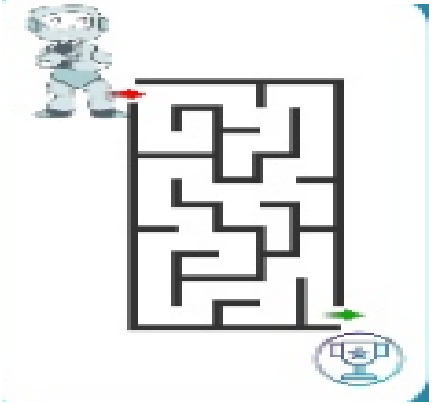
Reinforcement learning can be thought of as a hit and trial method of learning.

The machine gets a Reward or Penalty point for each action it performs. If the option is correct, the machine gains the reward point or gets a penalty point in case of a wrong response.









Supervised Learning

The task of supervised learning is this –

> Given a **training set** of N example input–output pairs
>
> $$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N) ,$$
>
> where each y_j was generated by an unknown function $y = f(x)$,
> discover a function h that approximates the true function f.

Here x and y can be any value; they need not be numbers.

The function h is a hypothesis

Learning is a search through the space of possible hypotheses for one that will perform well, even on new examples beyond the training set.

Supervised Learning

$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$ $\quad y = f(x)$

CLASSIFICATION : When the output y is one of a finite set of values (such as sunny, cloudy or rainy),the learning problem is called classification, and is called Boolean or binary classification.

REGRESSION : When the output y is a number (such as tomorrow's temperature), the learning problem is called regression.

Supervised Learning

Example: Fitting a function of a single variable to some data points.

data points are plotted in the (x, y) plane,

where y = f(x)

f - approximate it with a function h selected from a hypothesis space H, which for this example we will take to be the set of polynomials, such as $x^5 + 3x^2 + 2$.

Hypotheses

finding a suitable hypothesis can be difficult

- since the function *f* is unknown, it is hard to tell if the hypothesis *h* is a good approximation

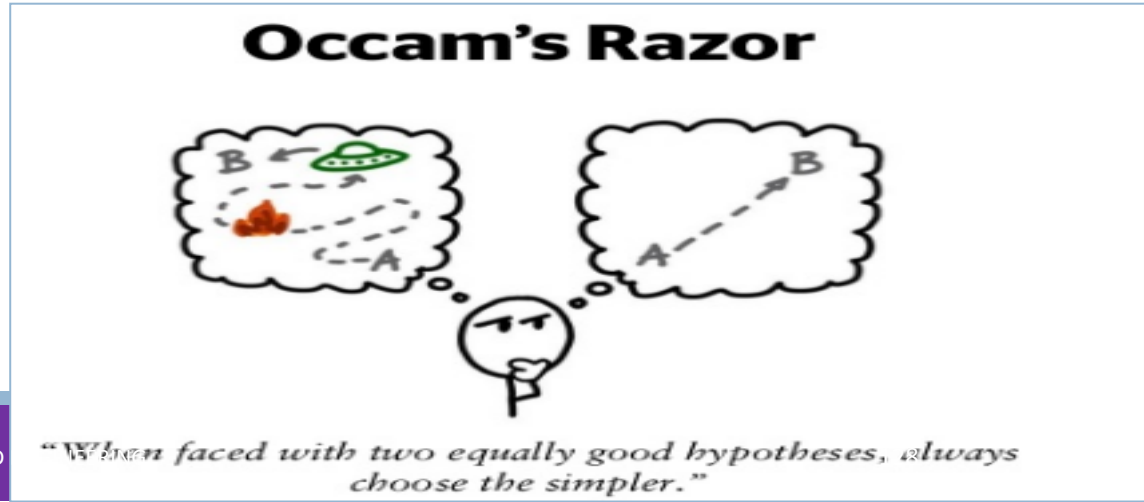
the *hypothesis space* describes the set of hypotheses under consideration

- e.g. polynomials, sinusoidal functions, propositional logic, predicate logic, ...

Hypotheses cont

- the choice of the hypothesis space can strongly influence the task of finding a suitable function
- while a very general hypothesis space (e.g. Turing machines) may be guaranteed to contain a suitable function, it can be difficult to find it

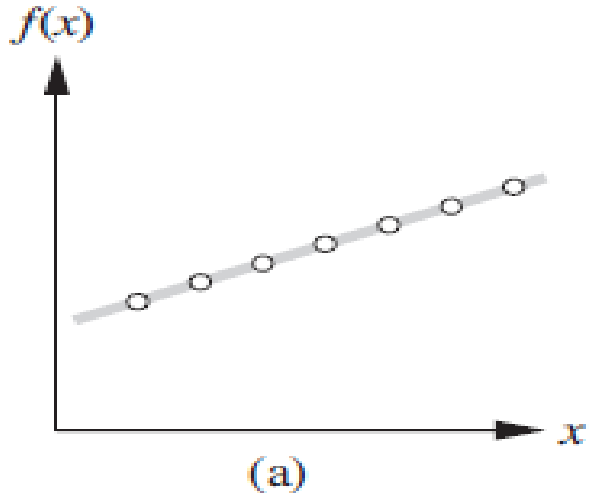
Ockham's razor: if multiple hypotheses are consistent with the data, choose the simplest one



Supervised Learning

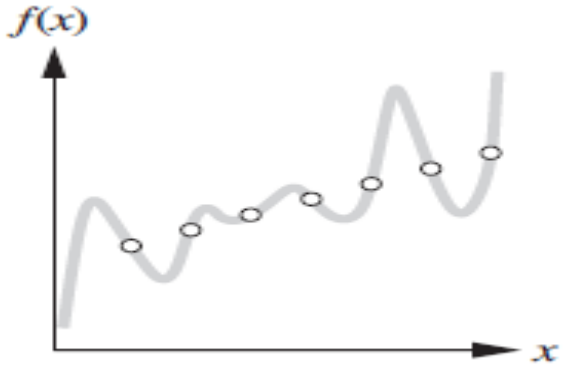
Shows some data with an exact fit by a straight line (the polynomial 0.4x + 3).

The line is called a consistent hypothesis because it agrees with all the data.



(a)

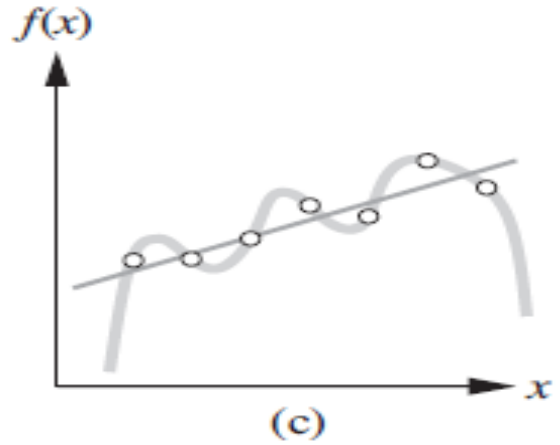
Shows a high degree-7 polynomial that is also consistent with the same data.



(b)

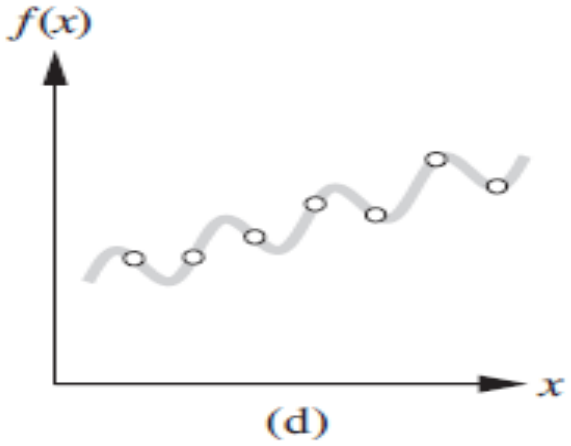
Supervised Learning

A different data set, which admits an exact degree-6 polynomial fit or an approximate linear fit.



(c)

A simple, exact sinusoidal fit to the same data set.



(d)

Agenda: Learning from Examples

- Forms of Learning
- Supervised Learning
- Learning Decision Trees
- Artificial Neural Networks
- Support Vector Machines
- Ensemble Learning

Learning Decision Trees

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- A decision tree represents a function that takes as input a vector of attribute values and returns a “decision”—a single output value.
- The input and output values can be discrete or continuous.

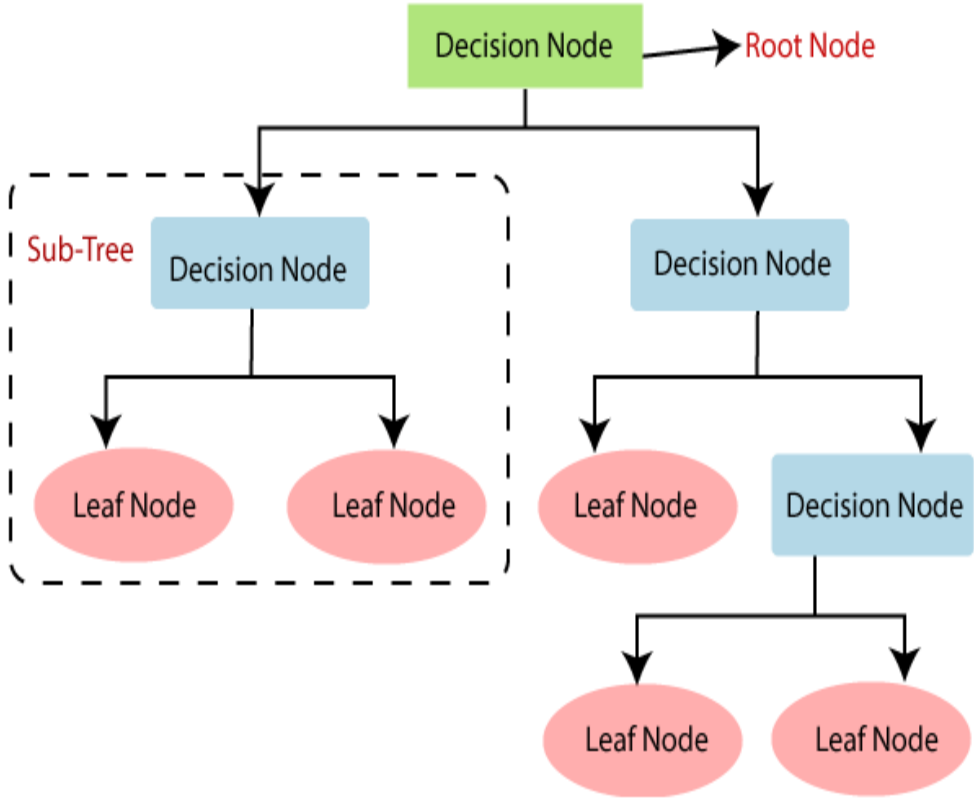
Decision tree representation

In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.**

Decision nodes are used to make any decision and have multiple branches

Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.



Learning Decision Trees

A decision tree reaches its decision by performing a sequence of tests.

Each internal node in the tree corresponds to a test of the value of one of the input attributes, Ai, and the branches from the node are labeled with the possible values of the attribute, Ai =v_{ik}.

Each leaf node in the tree specifies a value to be returned by the function.

Learning Decision Trees

Example : Build a decision tree to decide whether to wait for a table at a restaurant.

List the attributes that we will consider as part of the input

1. *Alternate*: whether there is a suitable alternative restaurant nearby.
2. *Bar*: whether the restaurant has a comfortable bar area to wait in.
3. *Fri/Sat*: true on Fridays and Saturdays.
4. *Hungry*: whether we are hungry.
5. *Patrons*: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
6. *Price*: the restaurant's price range ($, $$, $$$).
7. *Raining*: whether it is raining outside.
8. *Reservation*: whether we made a reservation.
9. *Type*: the kind of restaurant (French, Italian, Thai, or burger).
10. *WaitEstimate*: the wait estimated by the host (0–10 minutes, 10–30, 30–60, or >60).

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Learning Decision Trees

Choosing attribute tests

- To define information gain, we begin by defining a measure called entropy. *Entropy measures the impurity of a collection of examples.*
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

$$Entropy\,(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where,

$p_{\oplus}$ is the proportion of positive examples in S

$p_{\ominus}$ is the proportion of negative examples in S.

- The entropy is 0 if all members of S belong to the same class
- The entropy is 1 when the collection contains an equal number of positive and negative examples
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1

Learning Decision Trees

Choosing attribute tests

Suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this boolean classification is

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

$$Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$$

$$= 0.940$$

Learning Decision Trees

Choosing attribute tests

- ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Example: Information gain

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Let, $Values(Wind) = \{Weak, Strong\}$

$S = [9+, 5-]$

$S_{Weak} = [6+, 2-]$

$S_{Strong} = [3+, 3-]$

Information gain of attribute *Wind*:

$$Gain(S, Wind) = Entropy(S) - 8/14\ Entropy\ (S_{Weak}) - 6/14\ Entropy\ (S_{Strong})$$

$$= 0.94 - (8/14)* 0.811 - (6/14) *1.00$$

$$= 0.048$$

$$Entropy\ (S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

$$Entropy([9+, 5-]) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14)$$

$$= 0.940$$

Consider the below training dataset to draw the decision tree.

(i) Compute the entropy of the collection of training examples with respect to the target function classification?

(ii) Compute the information gain of A2 relative to these training examples?

Instances	Classification	A1	A2
1	-	F	F
2	-	F	F
3	+	F	T
4	-	T	T
5	+	T	F
6	+	T	F

Learning Decision Trees

The DECISION-TREE-LEARNING algorithm

function DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree

if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
  tree $\leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples \text{ and } e.A = v_k\}$
    subtree $\leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - A$, *examples*)
    add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
  return *tree*

Learning Decision Trees

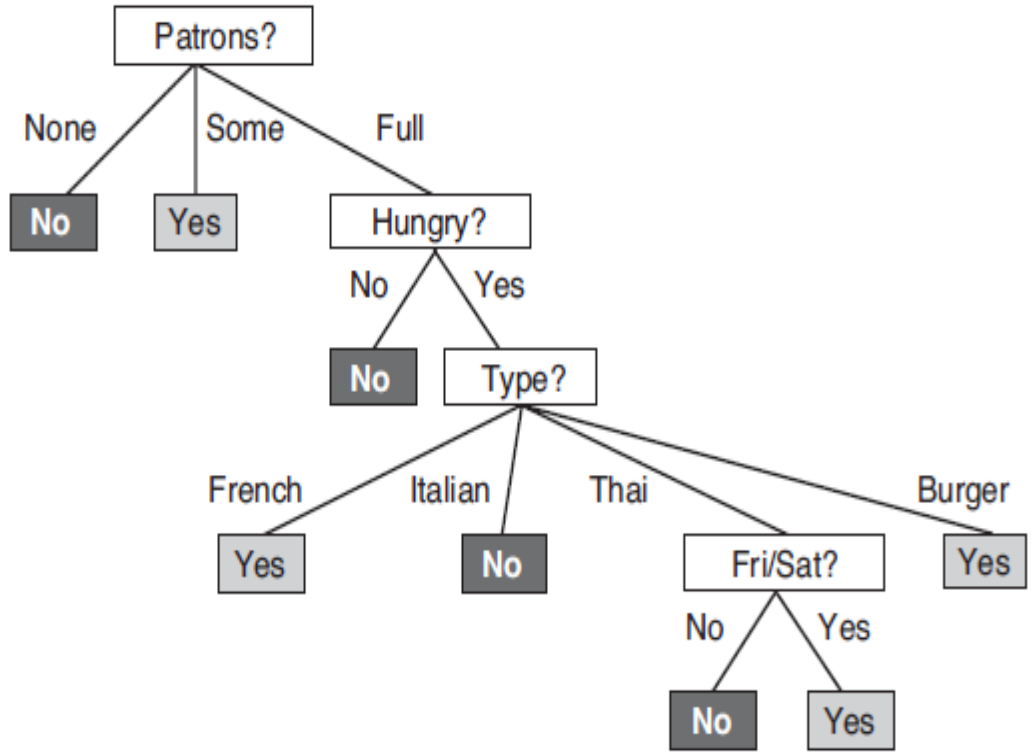


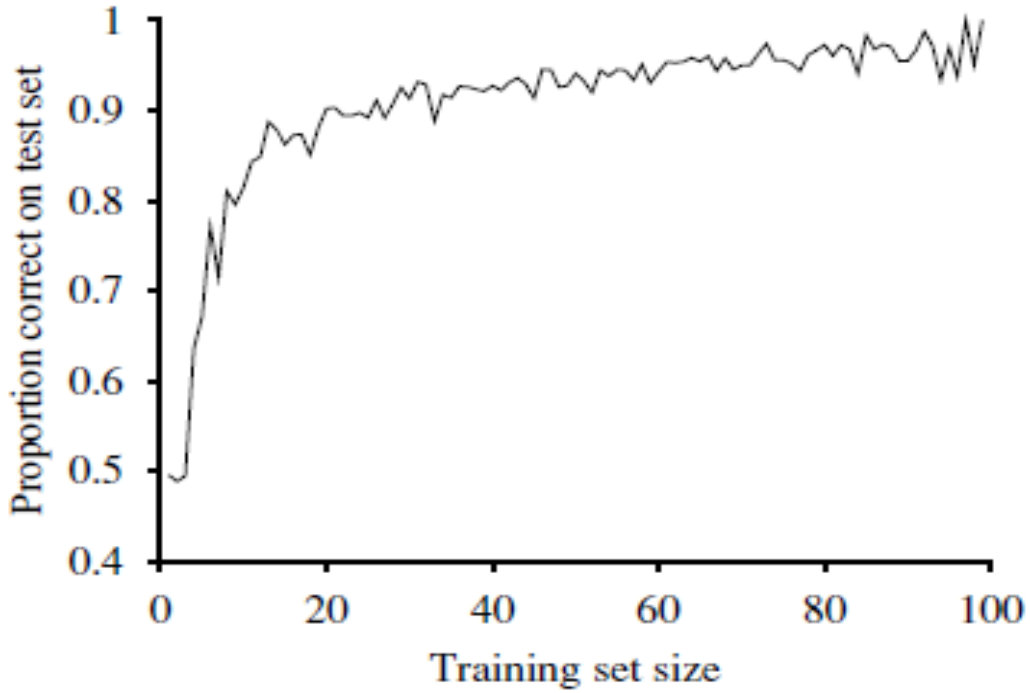
Figure 18.6 The decision tree induced from the 12-example training set.

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Learning Decision Trees

We can evaluate the accuracy of a learning algorithm with a learning curve, as shown in Figure .

We have 100 examples, which we split into a training set and a test set



Learning Decision Trees

Example

To Design a machine learning model

- A model is said to be a good machine learning model if it generalizes any new input data and make correct predictions.

Now, suppose we want to check how well our machine learning model learns and generalizes to the new data.

- Overfitting and Underfitting, which are majorly responsible for the poor performances of the machine learning algorithms.

Learning Decision Trees

Underfitting *(It's just like trying to fit undersized dress!)*

A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data.

Underfitting destroys the accuracy of our machine learning model.

- It usually happens when we have less data
- In such cases the model will probably make a lot of wrong predictions.

Underfitting can be avoided by using more data

Learning Decision Trees

Overfitting *(just like fitting ourselves in oversized dress!)*.

A statistical model is said to be overfitted, when we train it with a lot of data

When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set.

Then the model does not categorize the data correctly, because of too many details and noise.

Learning Decision Trees

Approaches to avoiding overfitting in decision tree learning

- **Pre-pruning (avoidance):** Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- **Post-pruning (recovery):** Allow the tree to overfit the data, and then post-prune the tree

Pruning is a technique in machine learning that reduce the size of decision tree by removing parts of the tree that do not provide power to classify instance.

Learning Decision Trees

Issues in Decision Tree

- Missing data
- Multivalued attributes
- Continuous and integer-valued input attributes
- Continuous-valued output attributes

Learning Decision Trees

Issues in Decision Tree

Missing data :In many domains, not all the attribute values will be known for every

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

Learning Decision Trees

Issues in Decision Tree

Multivalued attributes : When an attribute has many possible values, the information gain measure gives an inappropriate indication of the attribute's usefulness

One Approach: Use *GainRatio* instead of *Gain*

- The gain ratio measure penalizes attributes by incorporating a split information, that is sensitive to how broadly and uniformly the attribute splits the data

$$GainRatio(S,A) \equiv \frac{Gain(S,A)}{SplitInformation(S,A)}$$

$$SplitInformation(S,A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- where S_i is subset of S, for which attribute A has value v_i

Second approach is to allow a Boolean test of the form $A=v_k$, picking out just one of the possible values for an attribute, leaving the remaining values to possibly be tested later in the tree.

Learning Decision Trees

Issues in Decision Tree

Continuous and integer-valued input attributes

There are two methods for Handling Continuous Attributes

1. Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.

E.g., {high ≡ Temp > 35° C, med ≡ 10° C < Temp ≤ 35° C, low ≡ Temp ≤ 10° C}

2. Using thresholds for splitting nodes

e.g., A ≤ a produces subsets A ≤ a and A > a

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

Learning Decision Trees

Issues in Decision Tree

Continuous-valued output attributes

If we are trying to predict a numerical output value, such as the price of an apartment, then we need a regression tree rather than a classification tree.

Outline

- Forms of Learning
- Supervised Learning
- Learning Decision Trees
- Artificial Neural Networks
- Support Vector Machines
- Ensemble Learning

Artificial Neural Networks

ANN stands for Artificial Neural Networks

The inventor of the first neurocomputer is **Dr. Robert Hecht-Nielsen**

ANN is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.

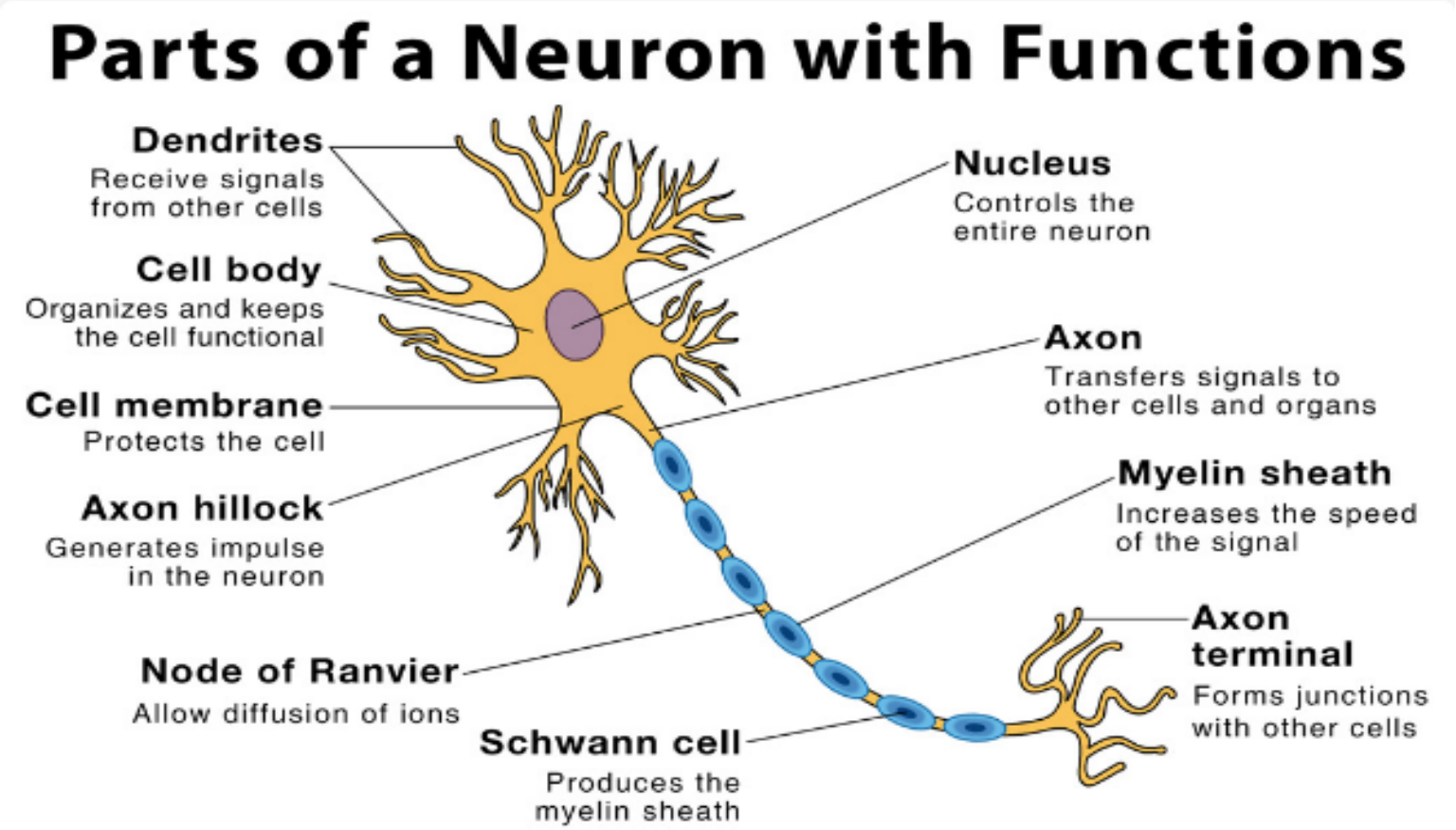
Artificial Neural Networks

Biological Motivation

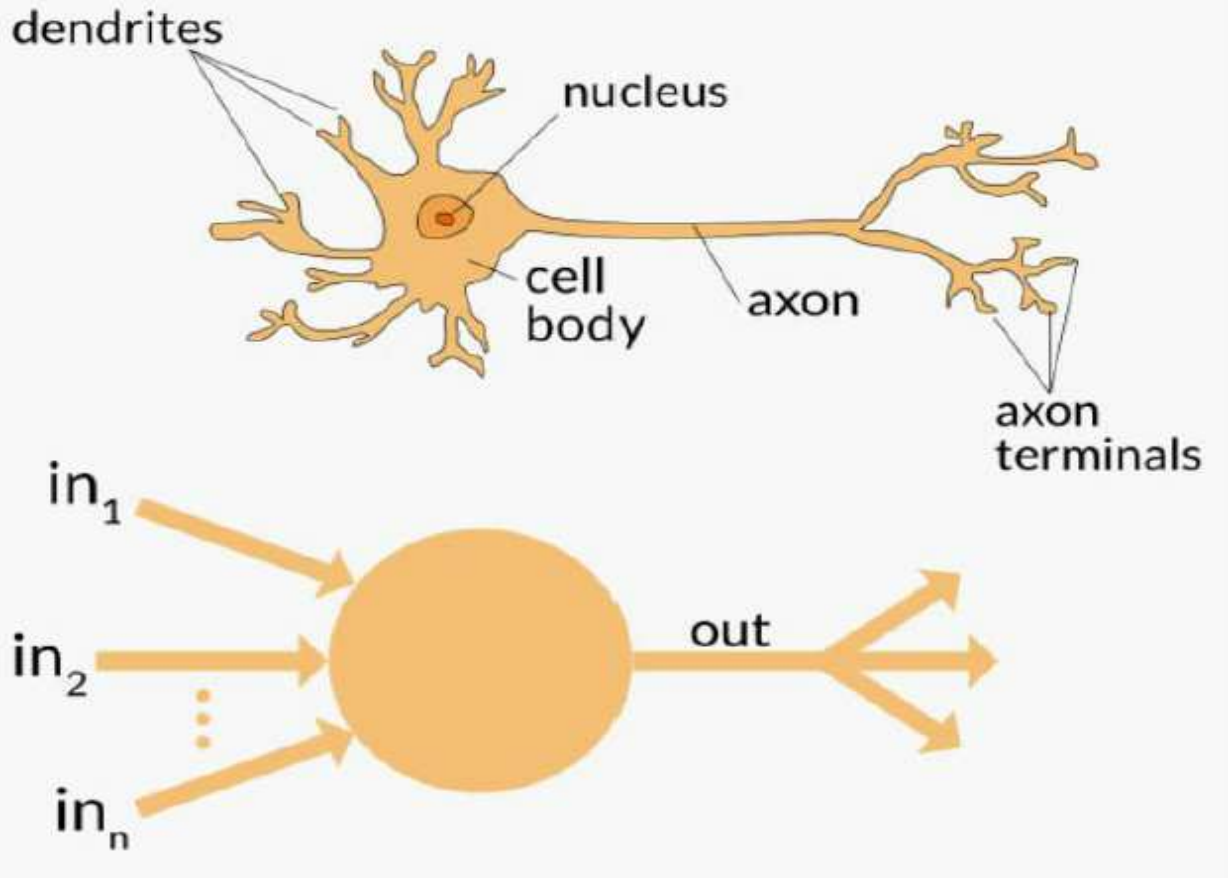
The study of artificial neural networks (ANNs) has been inspired by the observation that biological learning systems are built of very complex webs of interconnected Neurons

Human information processing system consists of brain neuron: basic building block cell that communicates information to and from various parts of body

Artificial Neural Networks



Artificial Neural Networks



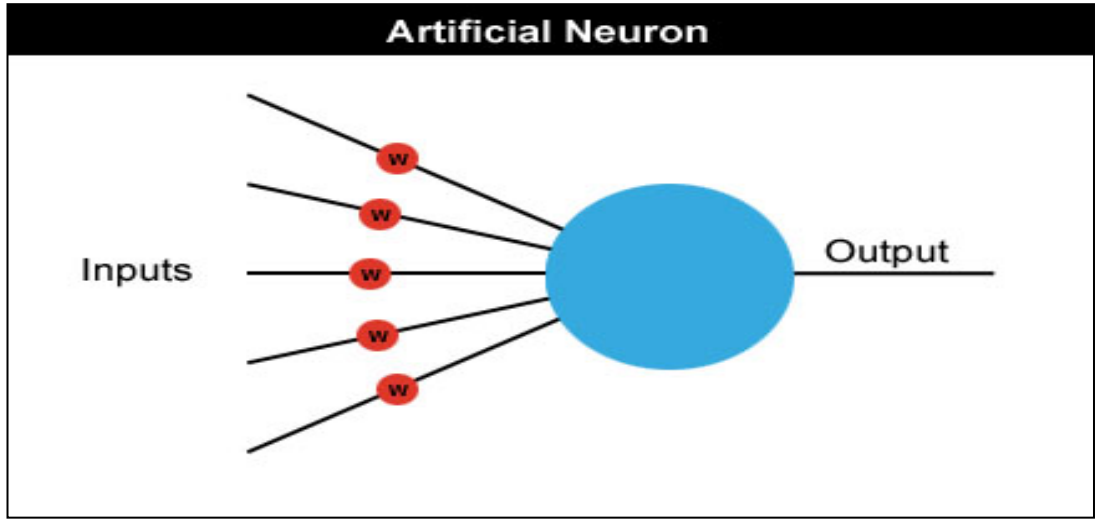
Dendrites: Input
Cell body: Processor
Axon: Output

Neural Networks

Neural networks are made up of many artificial neurons.

Each input into the neuron has its own weight associated with it illustrated by the red circle.

A weight is simply a floating point number and it's these we adjust when we eventually come to train the network.



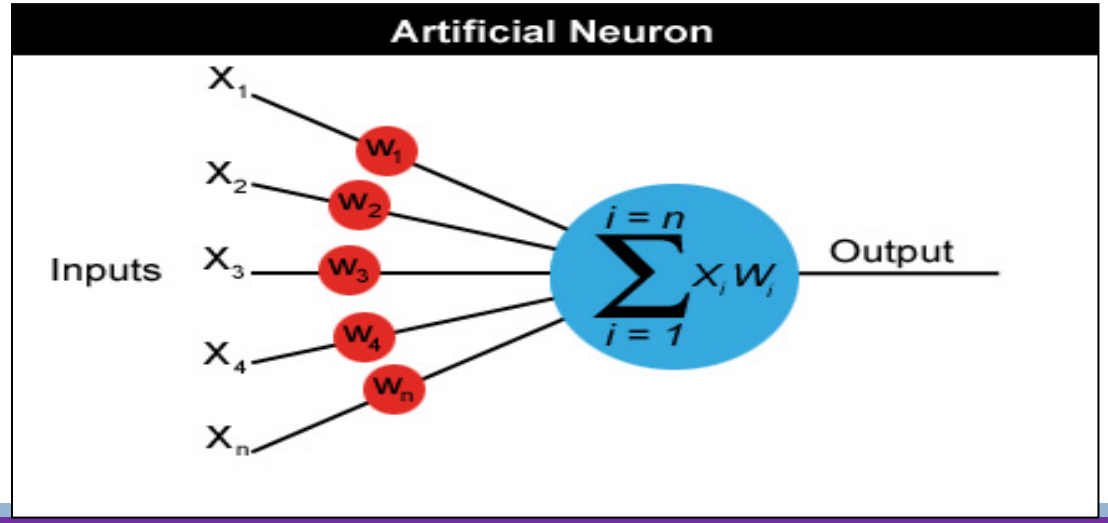
Neural networks

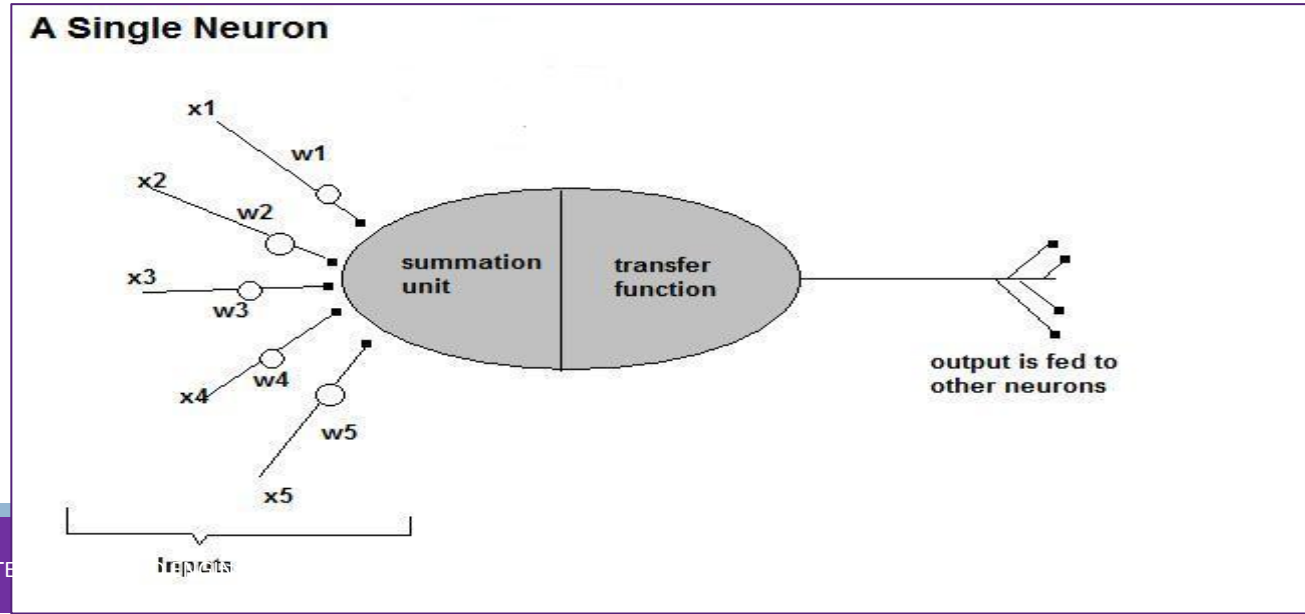
A neuron can have any number of inputs from 1 to n, where n is the total number of inputs.

The inputs may be represented therefore as *x1, x2, x3… xn.*

And the corresponding weights for the inputs as *w1, w2, w3… wn*.

Output *a* = x1w1 + x2w2 + x3w3... + …… xnwn



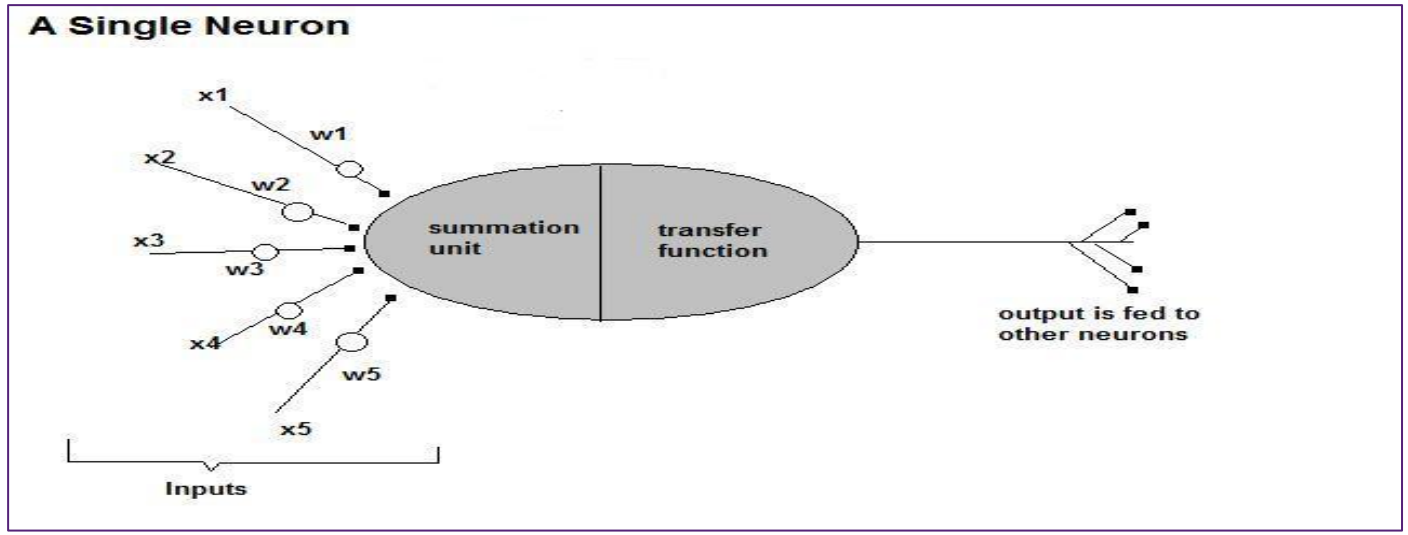


Activation Functions

- Use different functions to obtain different models.

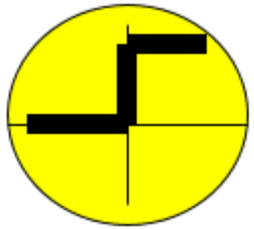
- 3 most common choices :

  1) Step function
  2) Sign function
  3) Sigmoid function



- An output of **1 represents firing** of a neuron down the axon.

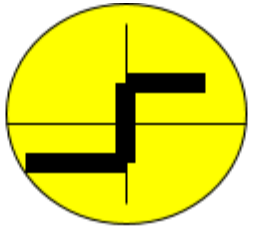
Activation Functions



Step function

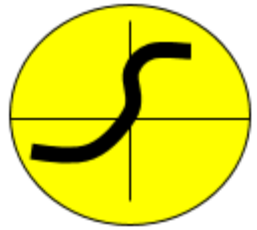
(Linear Threshold Unit)

step(x) = 1, if x >= threshold
0, if x < threshold



Sign function

sign(x) = +1, if x >= 0
-1, if x < 0

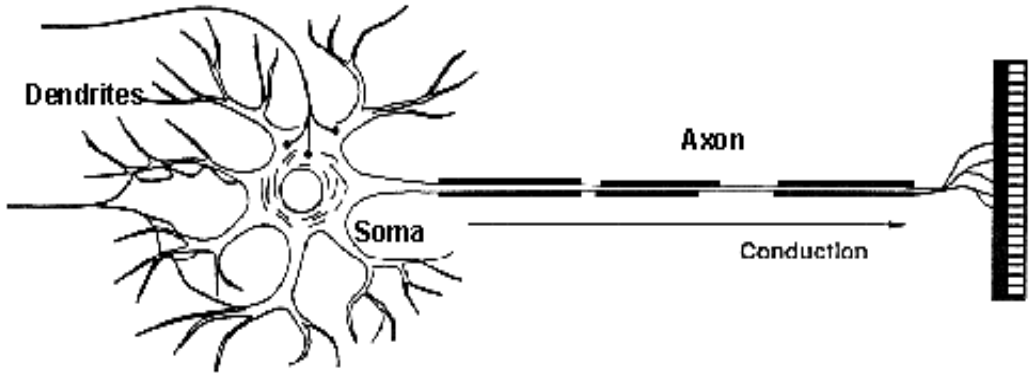


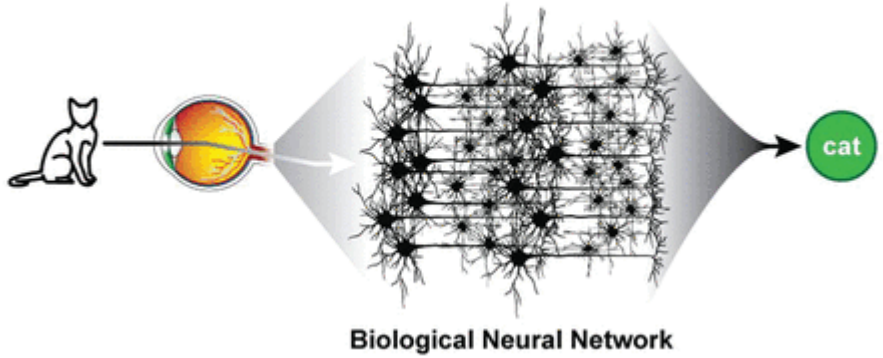
Sigmoid function

sigmoid(x) = $1/(1+e^{-x})$

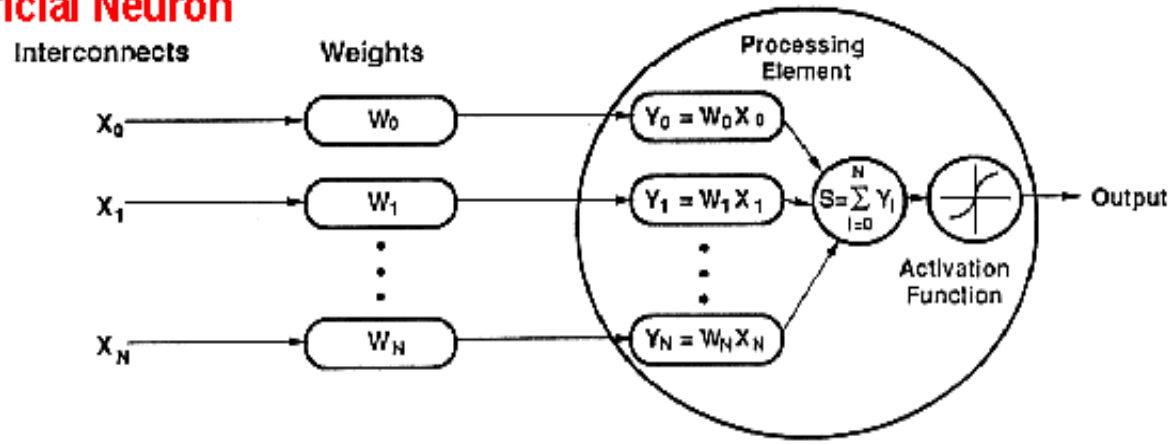
Introduction

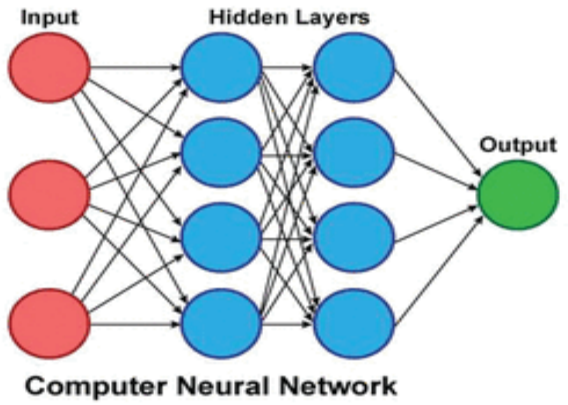
Biological Neuron





Artificial Neuron





Standard structure of an artificial neural network

Input units

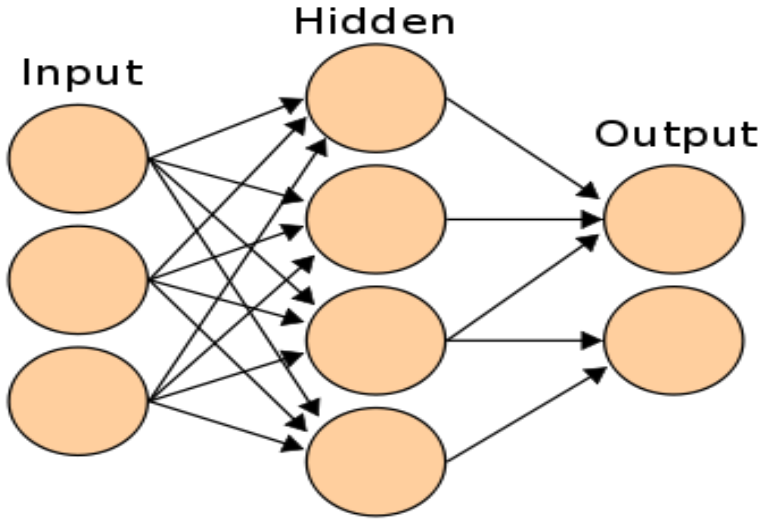
- represents the input as a fixed-length vector of numbers (user defined)

Hidden units

- calculate thresholded weighted sums of the inputs
- represent intermediate calculations that the network learns

Output units

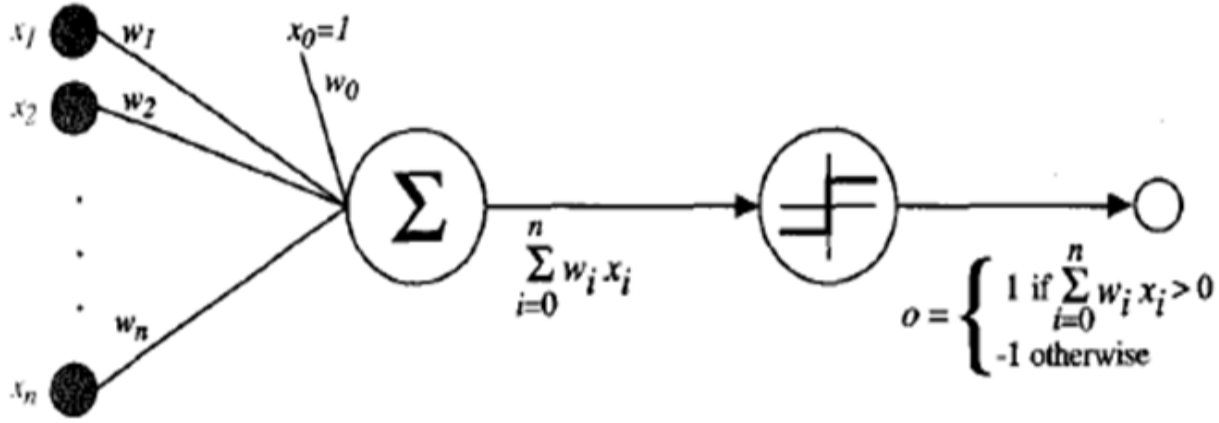
- represent the output as a fixed length vector of numbers



Perceptron

One type of ANN system is based on a unit called a perceptron, illustrated in Figure

A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise



Representational Power of Perceptrons

A single perceptron can be used to represent many boolean functions.

Perceptrons can represent all of the primitive boolean functions AND, OR,NAND (1 AND), and NOR (1 OR).

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Consider AND Gate problem. The gate returns if and only if both inputs are true. Let's say that $w_1 = 0.9$ and $w_2 = 0.9$. Activation threshold would be 0.5.

x1	x2	$\Sigma = x_1 * w_1 + x_2 * w_2$	Prediction output	$w = w + \alpha * \varepsilon$	Final target output
0	0	0 * 0.9 + 0 * 0.9 = 0 0<0.5	0		0
0	1	0 * 0.9 + 1 * 0.9 = 0.9 0.9>0.5	1 0	Error = 0-1= -1 $w_1 = w_1 + \alpha * \varepsilon$ = 0.9+0.5(-1) = 0.4 <0.5	0
1	0	1 * 0.4 + 0 * 0.4=0.4 0.4<0.5	0		0
1	1	1 * 0.4 + 1 * 0.4 = 0.8 0.8>0.5	1		1

RAMAIAH Institute of Technology

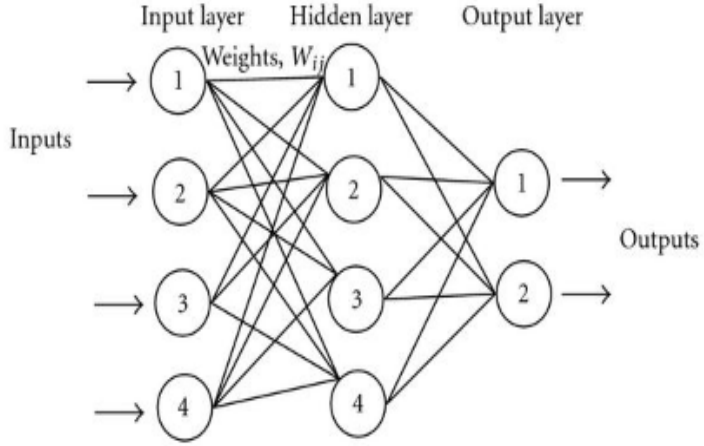
How do we actually use an artificial neuron?

Feedforward network

A feedforward neural network is an artificial neural network where the nodes never form a cycle.

Neural network has an input layer, hidden layers, and an output layer.

It is the first and simplest type of artificial neural network.



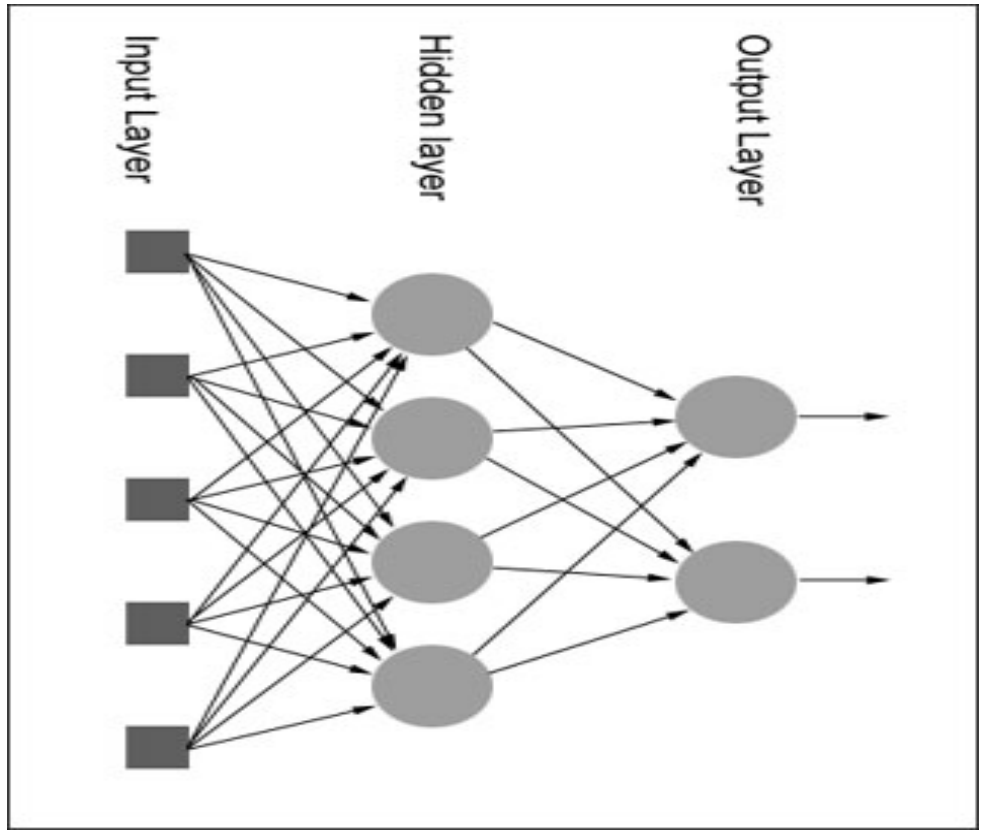
How do we actually use an artificial neuron?

Feedforward network:

The neurons in each layer feed their output forward to the next layer until we get the final output from the neural network.

There can be any number of hidden layers can exist within a feedforward network.

The number of neurons can be completely arbitrary.

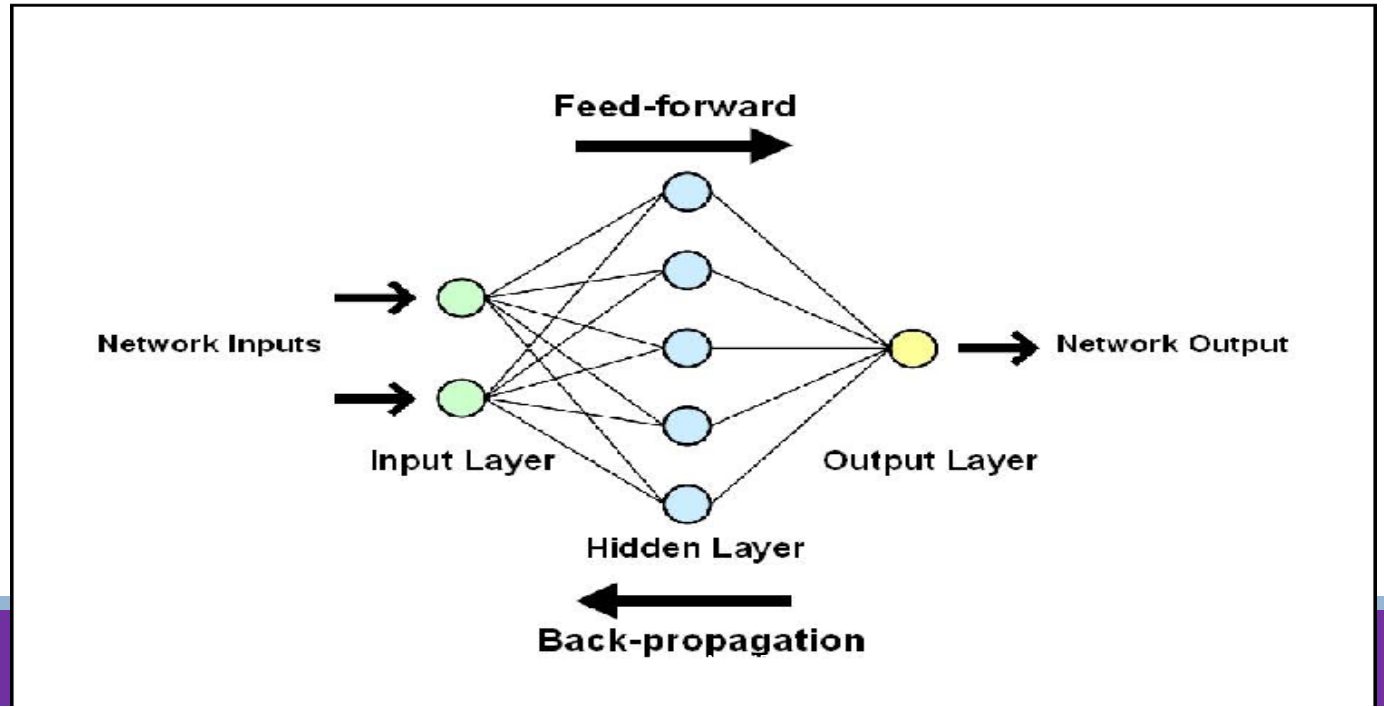


RAMAIAH
Institute of Technology

How do we actually use an artificial neuron?

A neural network is used to build predictive models from large databases.

It helps you to conduct image understanding, human learning, computer speech, etc.



How do we actually use an artificial neuron?

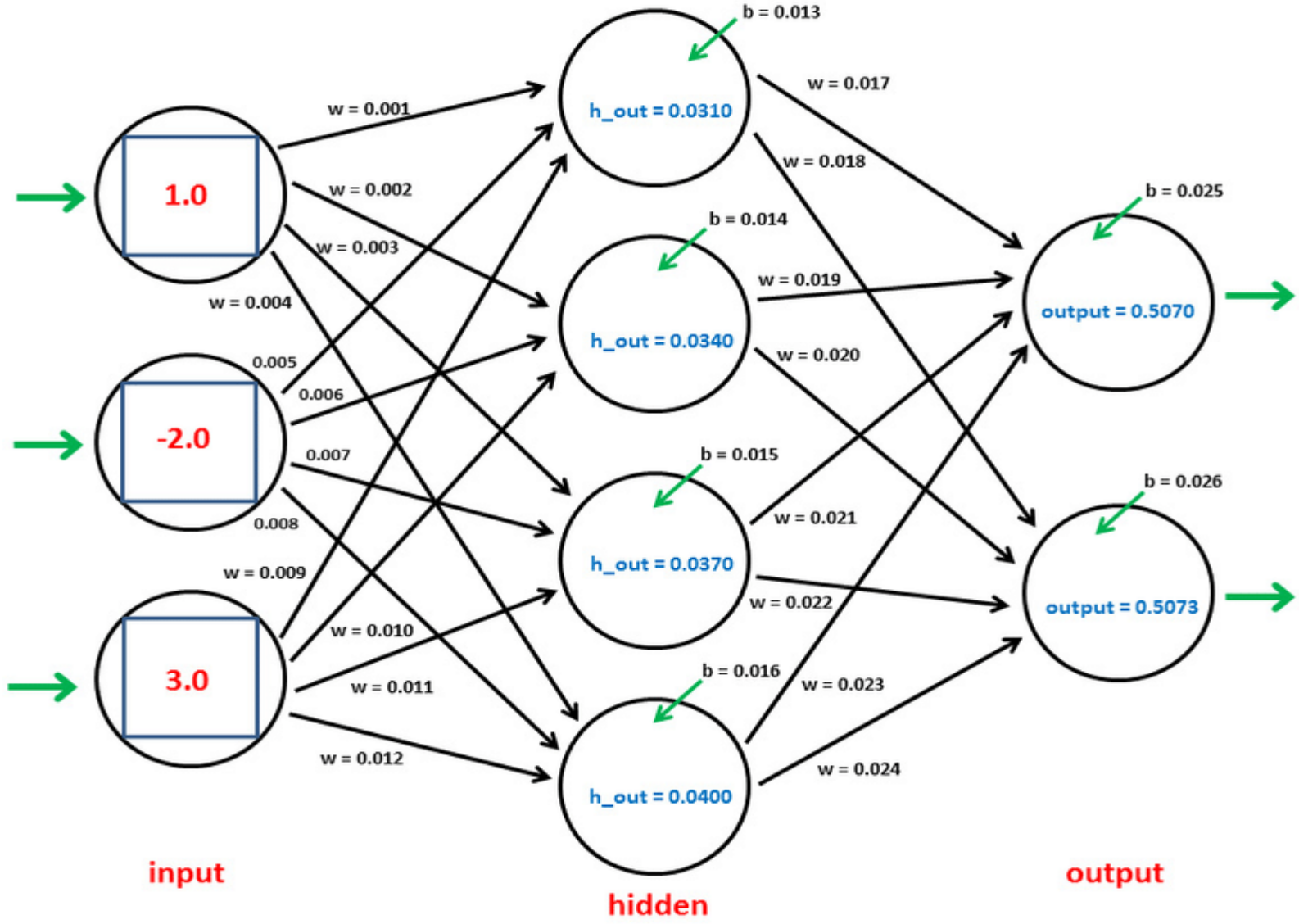
Backpropogation

Backpropagation in neural network is a short form for “backward propagation of errors.”

It is a standard method of training artificial neural networks.

It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration).

Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.


1.0
-2.0
3.0
w = 0.001
w = 0.002
w = 0.003
w = 0.004
0.005
0.006
0.007
0.008
w = 0.009
w = 0.010
w = 0.011
w = 0.012
b = 0.013
h_out = 0.0310
b = 0.014
h_out = 0.0340
b = 0.015
h_out = 0.0370
b = 0.016
h_out = 0.0400
w = 0.017
w = 0.018
w = 0.019
w = 0.020
w = 0.021
w = 0.022
w = 0.023
w = 0.024
b = 0.025
output = 0.5070
b = 0.026
output = 0.5073
input
hidden
output

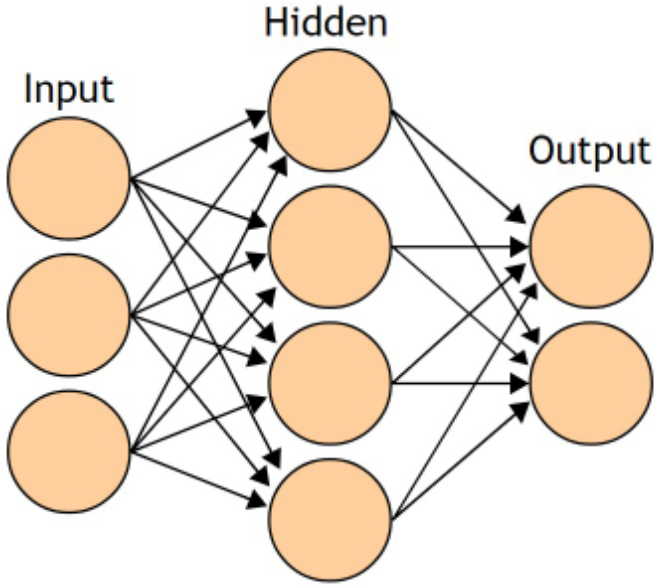
ANN and DEEP LEARNING

- ANN with a single layer is known as shallow network
- ANN with multiple hidden layers is known as deep neural network
- Not just multiple hidden layers sometimes the type of hidden layer is also different.
- This concept of solving problems with multiple hidden layers is known as **deep learning**

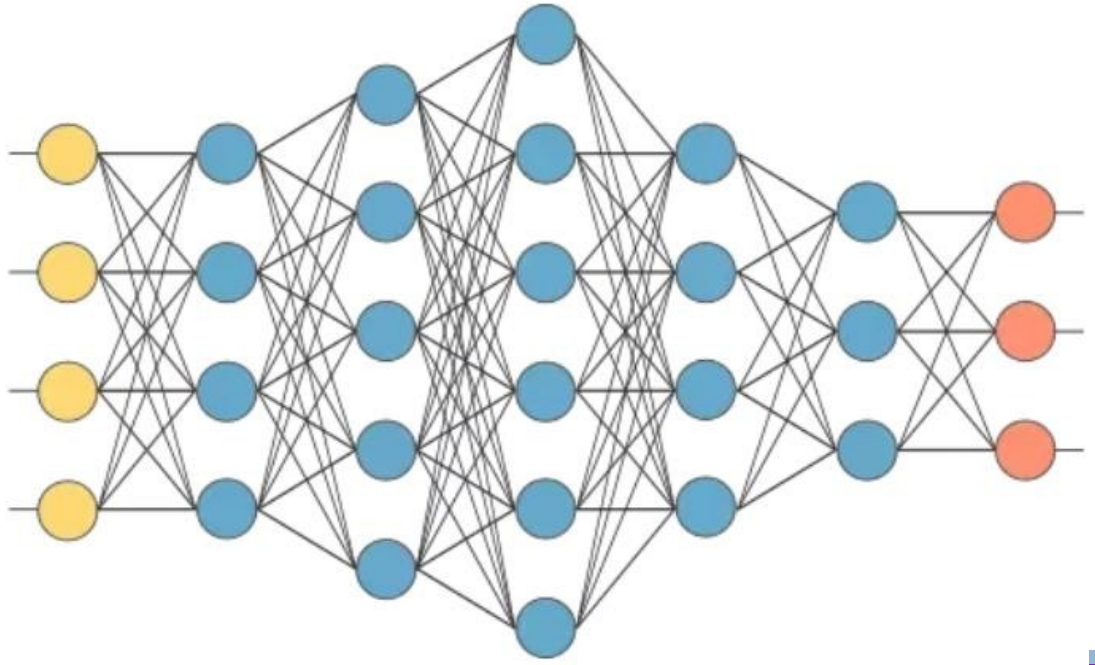
DEEP VS SHALLOW NETWORKS

A neural network with single hidden layer is called a shallow network

A neural network with more than one hidden layer is called deep neural network



shallow network



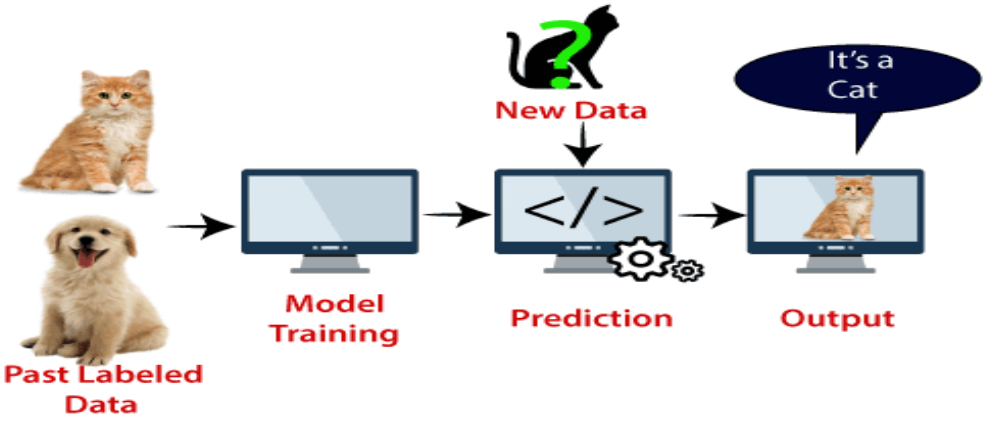
Deep network

Outline

- Forms of Learning
- Supervised Learning
- Learning Decision Trees
- Artificial Neural Networks
- Support Vector Machines
- Ensemble Learning

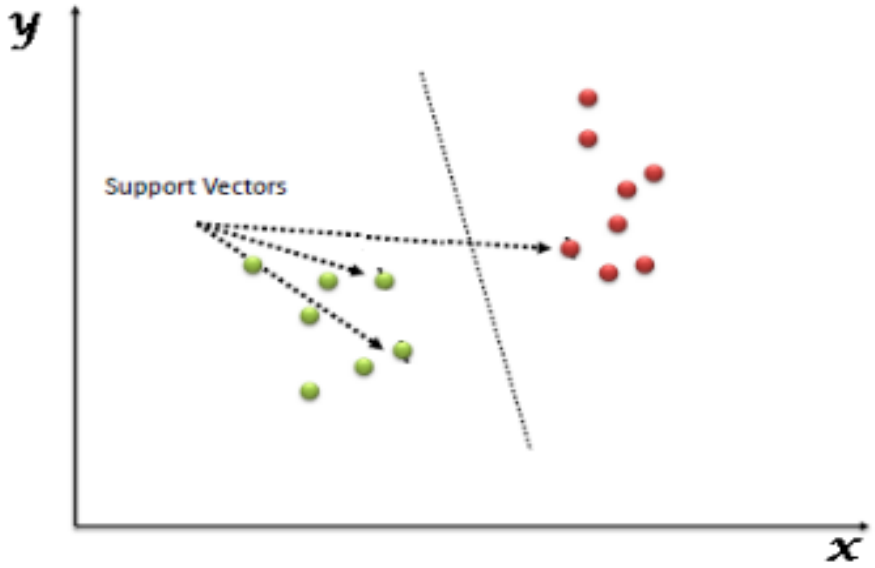
Support Vector Machine(SVM)

- SVM is Supervised Learning algorithm which is used for Classification as well as Regression problems.
- Primarily, it is used for Classification problems in Machine Learning.
- SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.



Support Vector Machine(SVM)

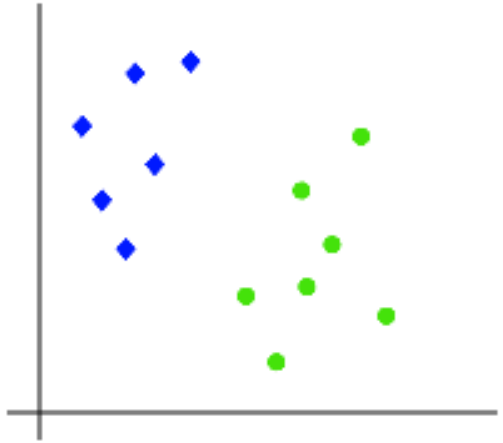
- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes.
- This best decision boundary is called a Hyperplane.



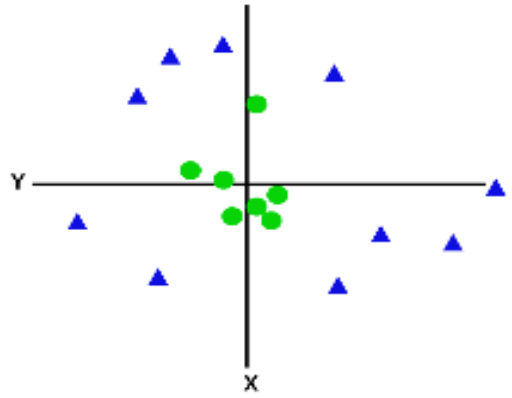
Support Vector Machine(SVM)

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line.



Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line.



Support Vector Machine(SVM)

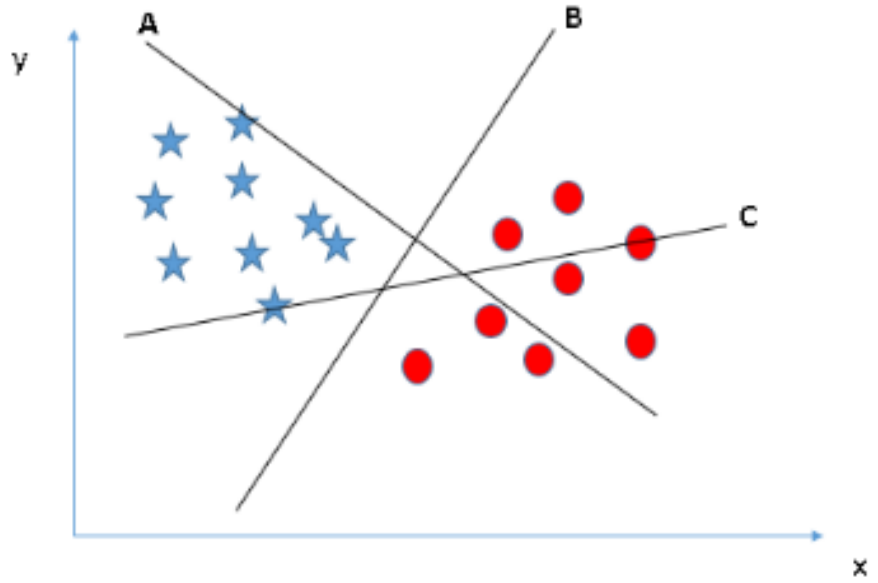
Hyperplane

There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space

But we need to find out the best decision boundary that helps to classify the data points.

Support Vector Machine(SVM)

How can we identify the right hyper-plane



Support Vector Machine(SVM)

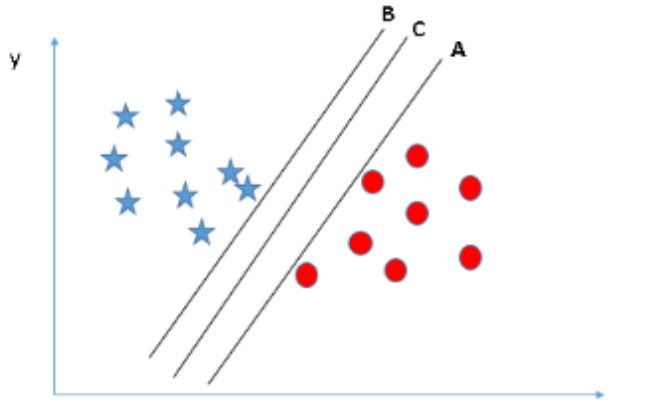
Linear Data

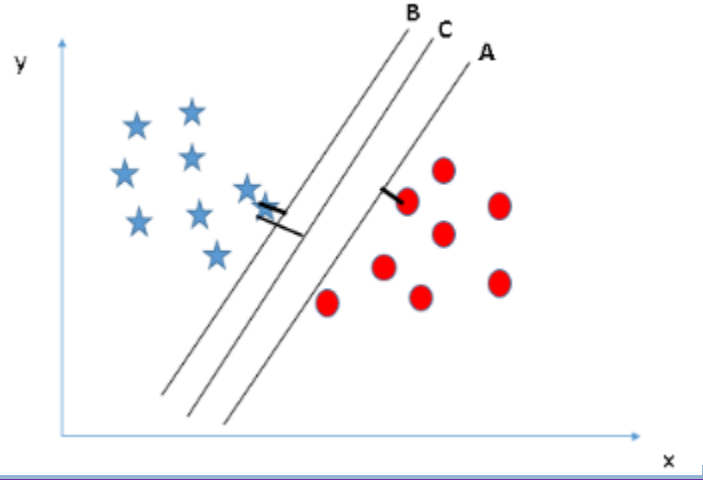
The distance between the vectors and the hyperplane is called as **margin**.

Goal of SVM is to maximize this margin.

The hyperplane C with maximum margin is called the **optimal hyperplane**.

If we select a hyper-plane having low margin then there is high chance of miss-classification.

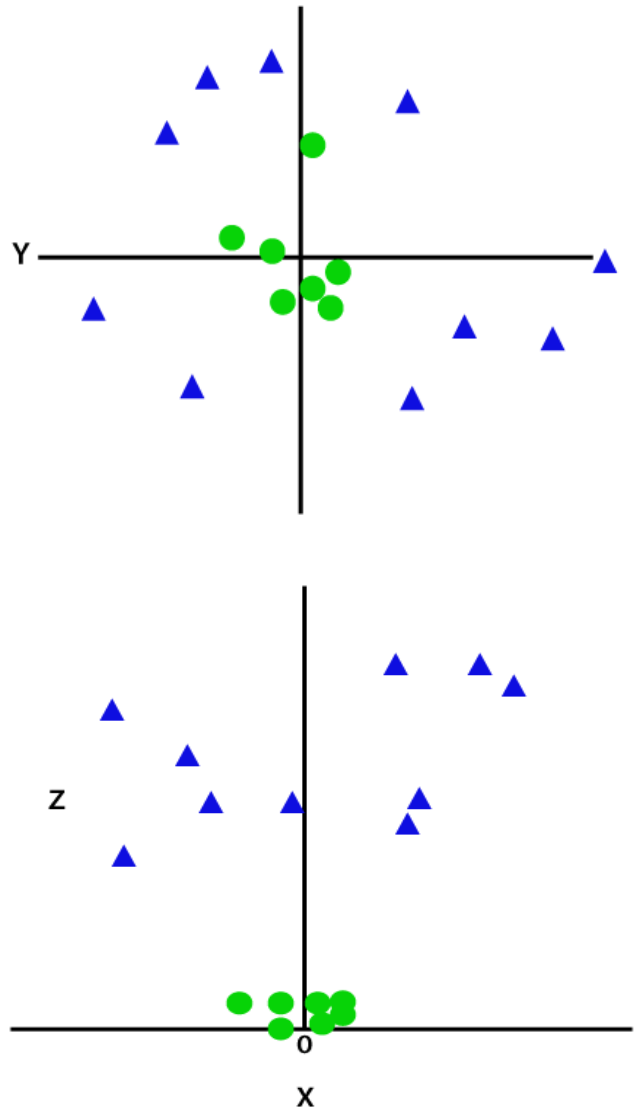




Support Vector Machine(SVM)

Nonlinear Data

- To separate these data points, need to add one more dimension.
- For non-linear data, will add a third dimension z.
- It can be calculated as: $z=x^2+y^2$

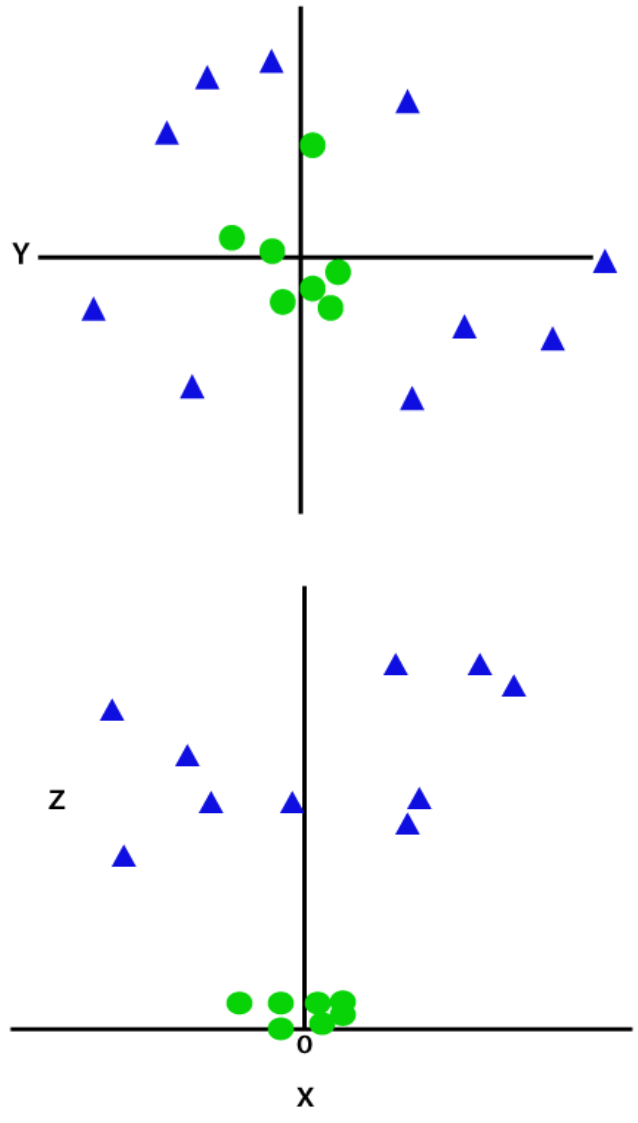


Support Vector Machine(SVM)

Nonlinear Data

SVM algorithm has a technique called the kernel trick.

The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.



Support Vector Machine(SVM)

Properties of SVM

1. SVMs construct a **maximum margin separator**
2. SVMs create a linear separating hyperplane, but they have the ability to embed the data into a higher-dimensional space, using the so-called **kernel trick.**
3. SVMs are a nonparametric method—they retain training examples and potentially need to store them all.

Outline

- Forms of Learning
- Supervised Learning
- Learning Decision Trees
- Artificial Neural Networks
- Support Vector Machines
- Ensemble Learning

Ensemble Learning

- Most learning methods use single hypothesis from a hypothesis space to make predictions.
- But ensemble learning methods is to select a collection, or ensemble, of hypotheses from the hypothesis space and combine their predictions.
- For example, during cross-validation we might generate twenty different decision trees, and have them vote on the best classification for a new example.

Thank you