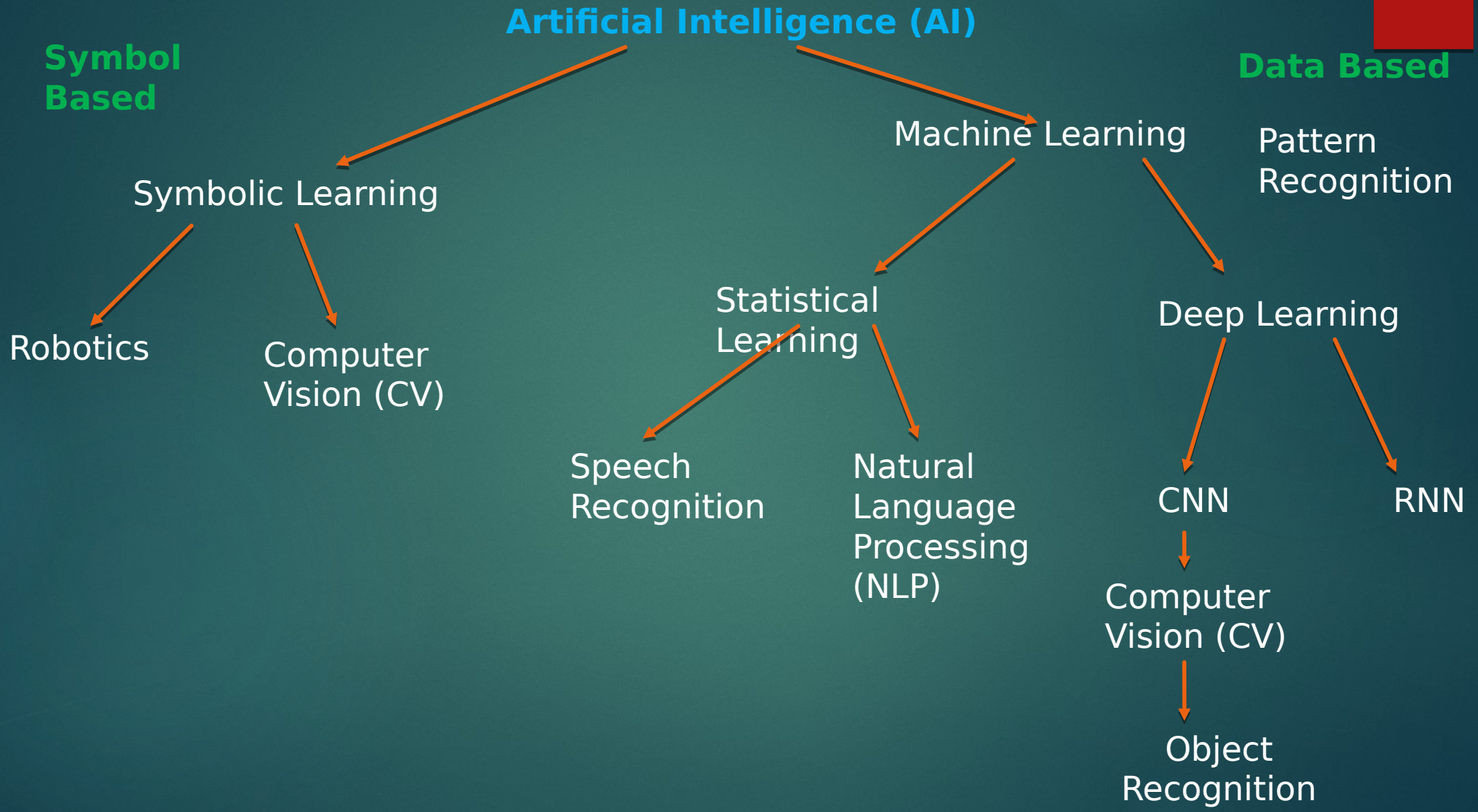
Artificial Intelligence

INTERIM SEMESTER 2021-22
BPL
CSE3007-LT-AB306
FACULTY: SIMI V.R.

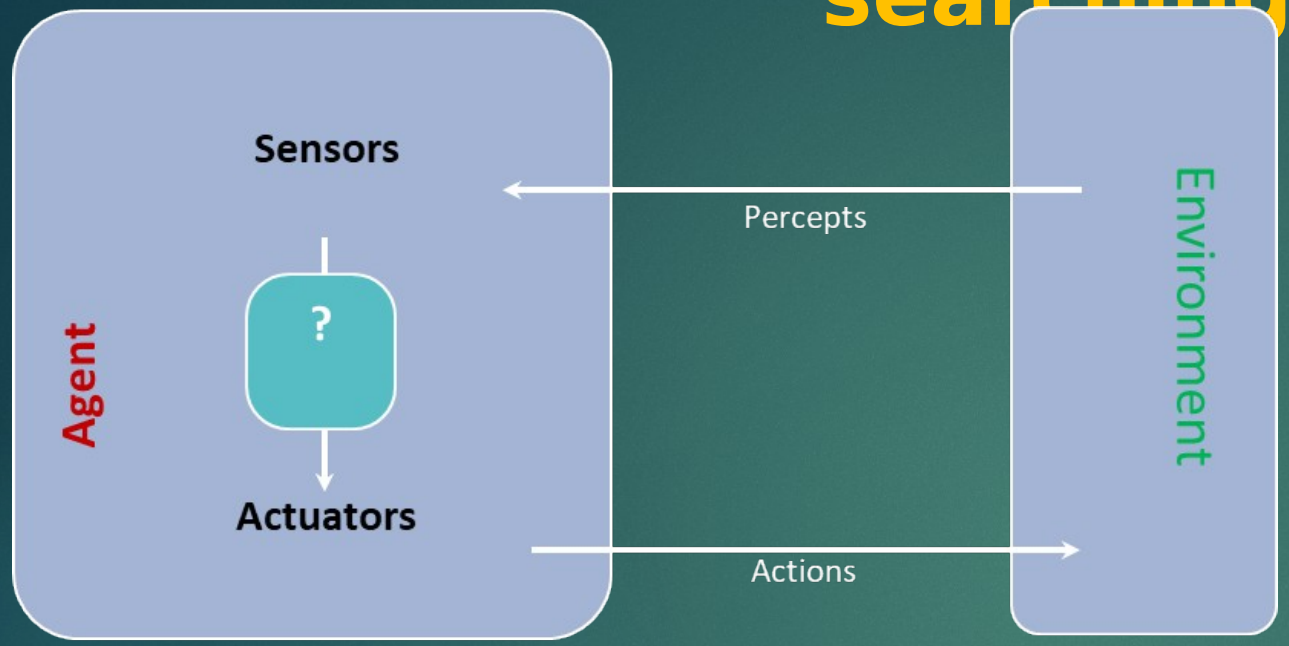
Computer Science

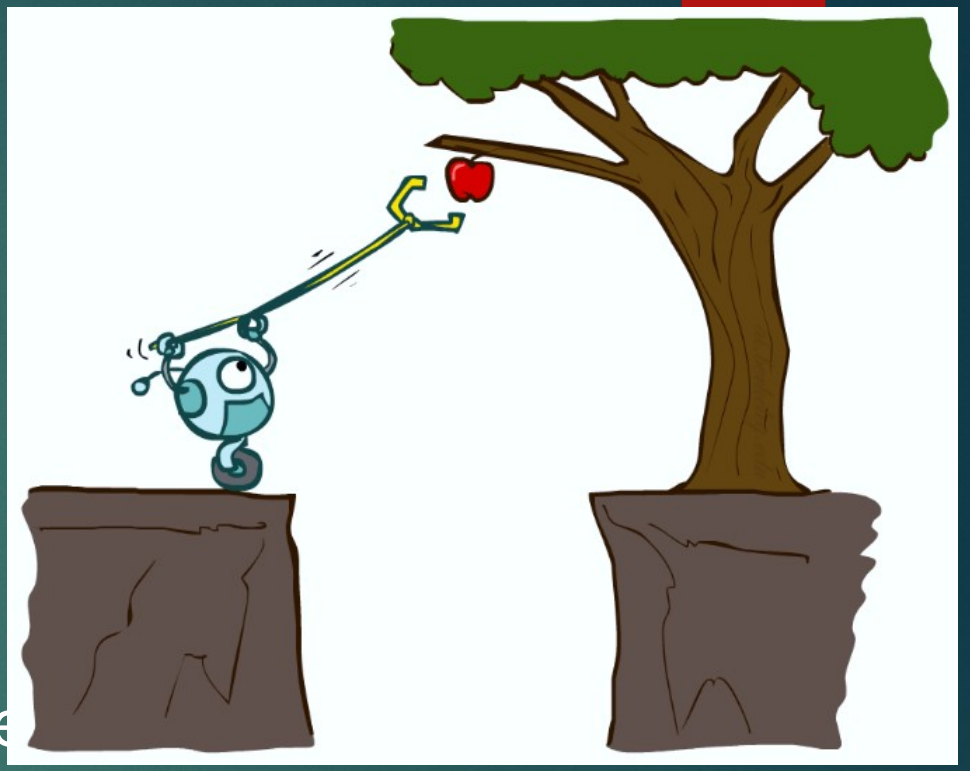


What can AI do today?

- Robotic vehicles
- Speech recognition
- Autonomous planning and scheduling
- Game playing
- Spam fighting
- Logistics planning
- Robotics
- Machine Translation

PROBLEM SOLVING-Solving problems by searching





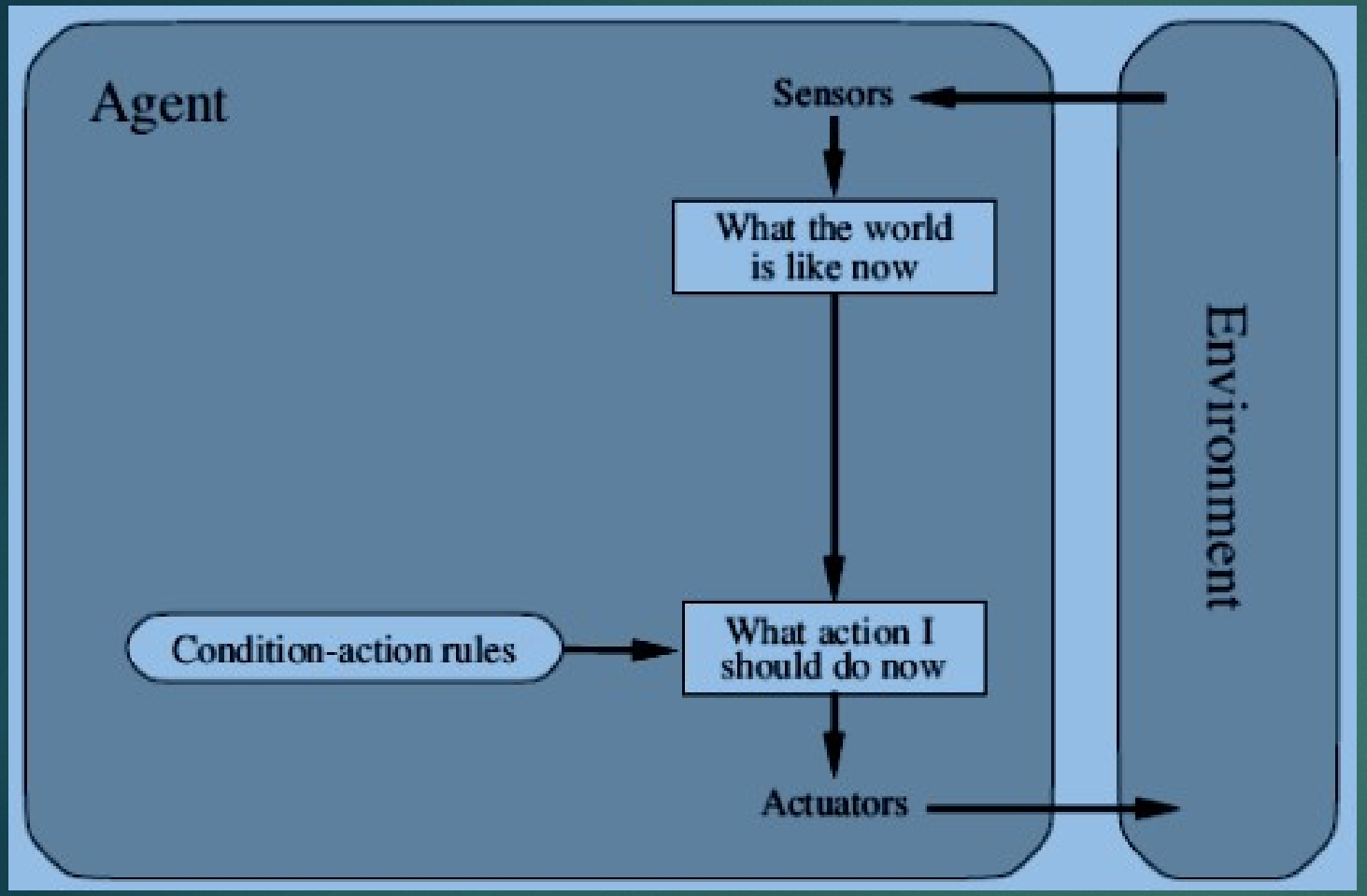
Problem-solving agent - Goal-based age

Goal formulation - is the first step in problem solving.

Problem formulation - is the process of deciding what actions and states to consider

- *an agent with several immediate options of unknown value can decide what to do by first examining* future *actions that eventually*

Problem-solving agent



Schematic diagram of a simple reflex agent.

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

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

Examples of agent types and their PEAS descriptions

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Problem solving agents

- **The agent function** for an agent specifies the action taken by the agent in response to any percept sequence.
- **The performance measure** evaluates the behaviour of the agent in an environment.
- **An agent** is something that perceives and acts in an environment.
- **A rational agent** is one that does the right thing-conceptually speaking, every entry in the table for the agent function is filled out correctly. It acts so as to maximize the expected value of the performance measure, given the percept sequence it has seen so far.
- **A task environment** specification includes the performance measure, the external environment, the actuators, and the sensors.

  In designing an agent, the first step must always be to specify the task environment as fully as possible. They can be fully or partially observable, single-agent or multiagent, deterministic or stochastic, episodic or sequential, static or dynamic, discrete or continuous, and known or unknown.
- **The agent program** implements the agent function. There exists a variety of basic agent-program designs reflecting the kind of information made explicit and used in the decision process. The designs vary in efficiency, compactness, and flexibility. The appropriate design of the agent program depends on the nature of the environment.
- **Simple reflex agents** respond directly to percepts, whereas model-based reflex agents maintain internal state to track aspects of the world that are not evident in the current percept.
- **Goal-based agents** act to achieve their goals
- **Utility-based agents** try to maximize their own expected “happiness.”
- All agents can improve their performance through **learning**.

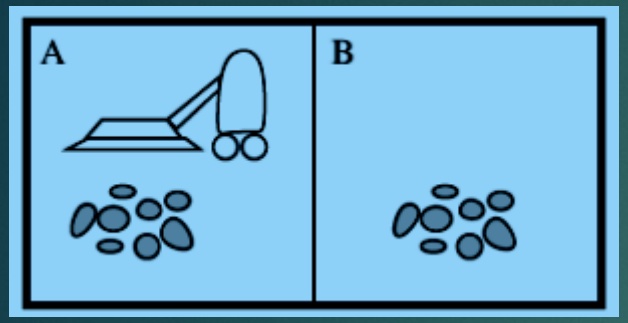
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

Toy problems - Vacuum world

- **States**: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \cdot 2^n$ states.
- **Initial state**: Any state can be designated as the initial state.
- **Actions**: In this simple environment, each state has just three actions: *Left*, *Right*, and *Suck*. Larger environments might also include *Up* and *Down*.
- **Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect.
- **Goal test**: This checks whether all the squares are clean.
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

Intelligent Agents



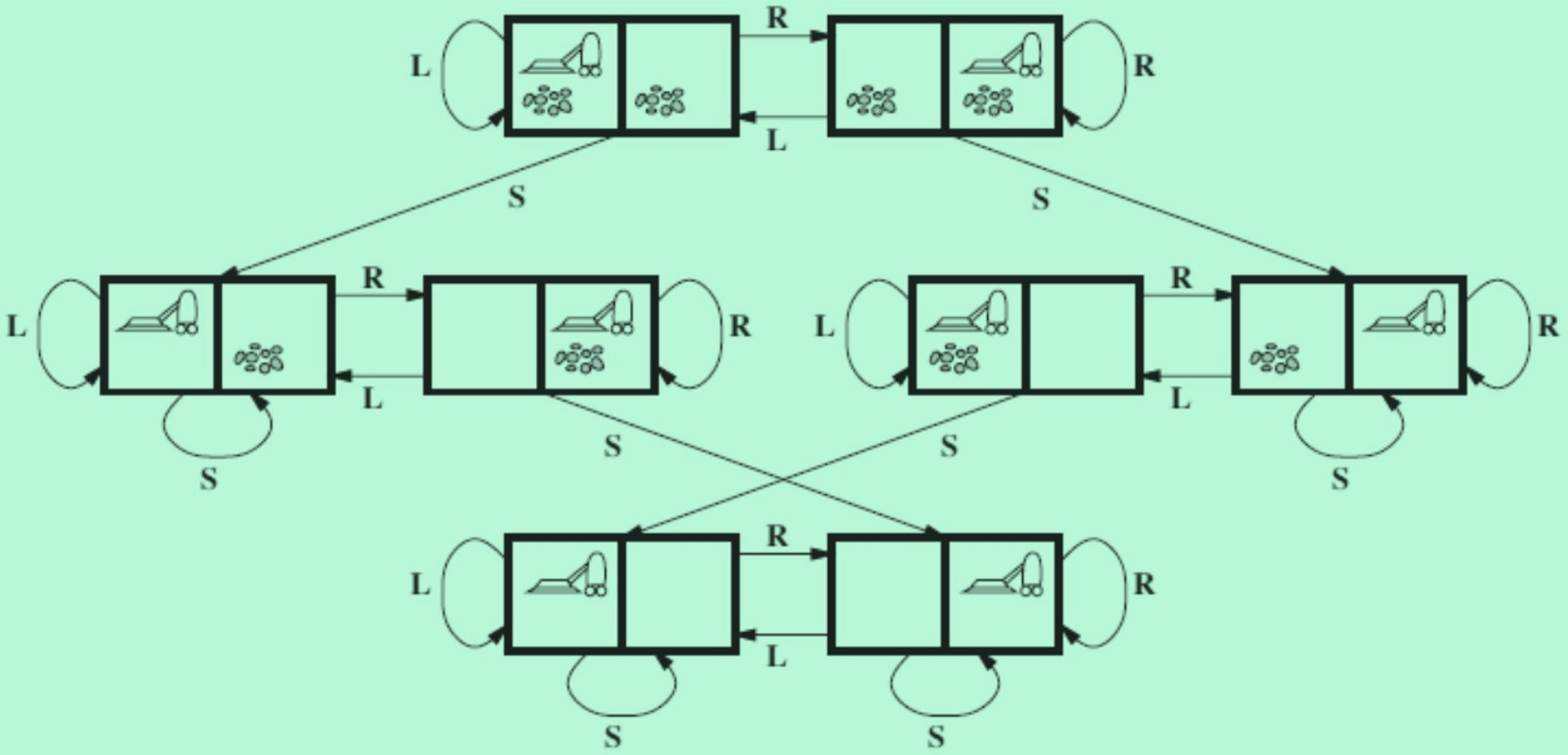
A vacuum-cleaner world with just two locations.

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

Partial tabulation of a simple agent function for the vacuum-cleaner world

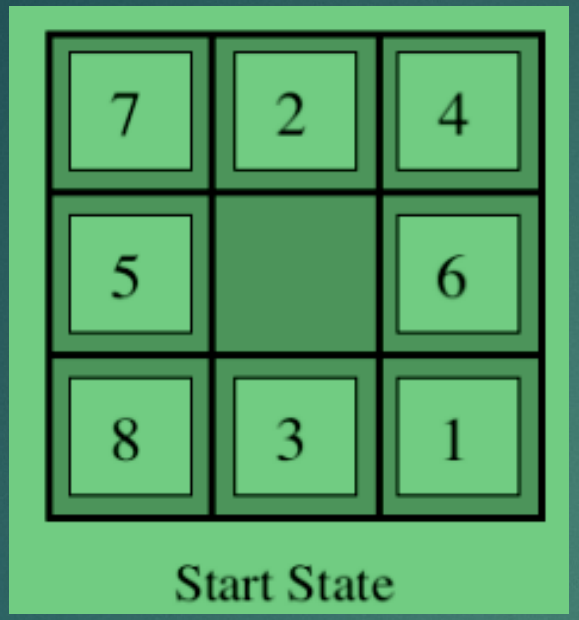
Example—the vacuum-cleaner world: This world is so simple that we can describe everything that happens. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing.

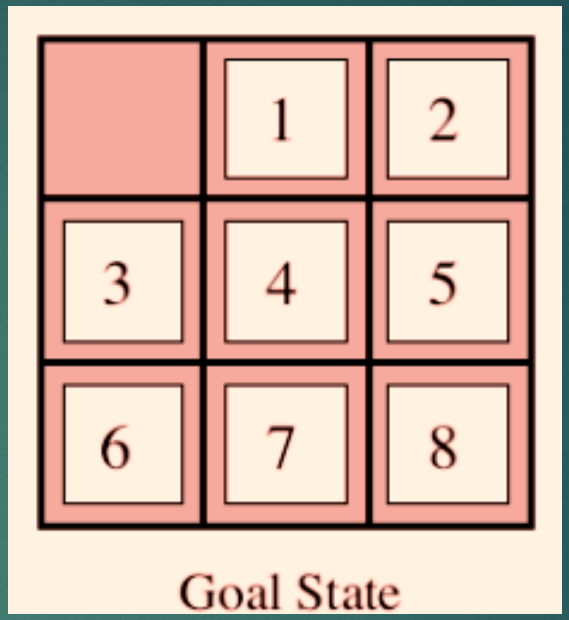
One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square.



Toy problems The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

8-puzzle (Sliding-block puzzles)



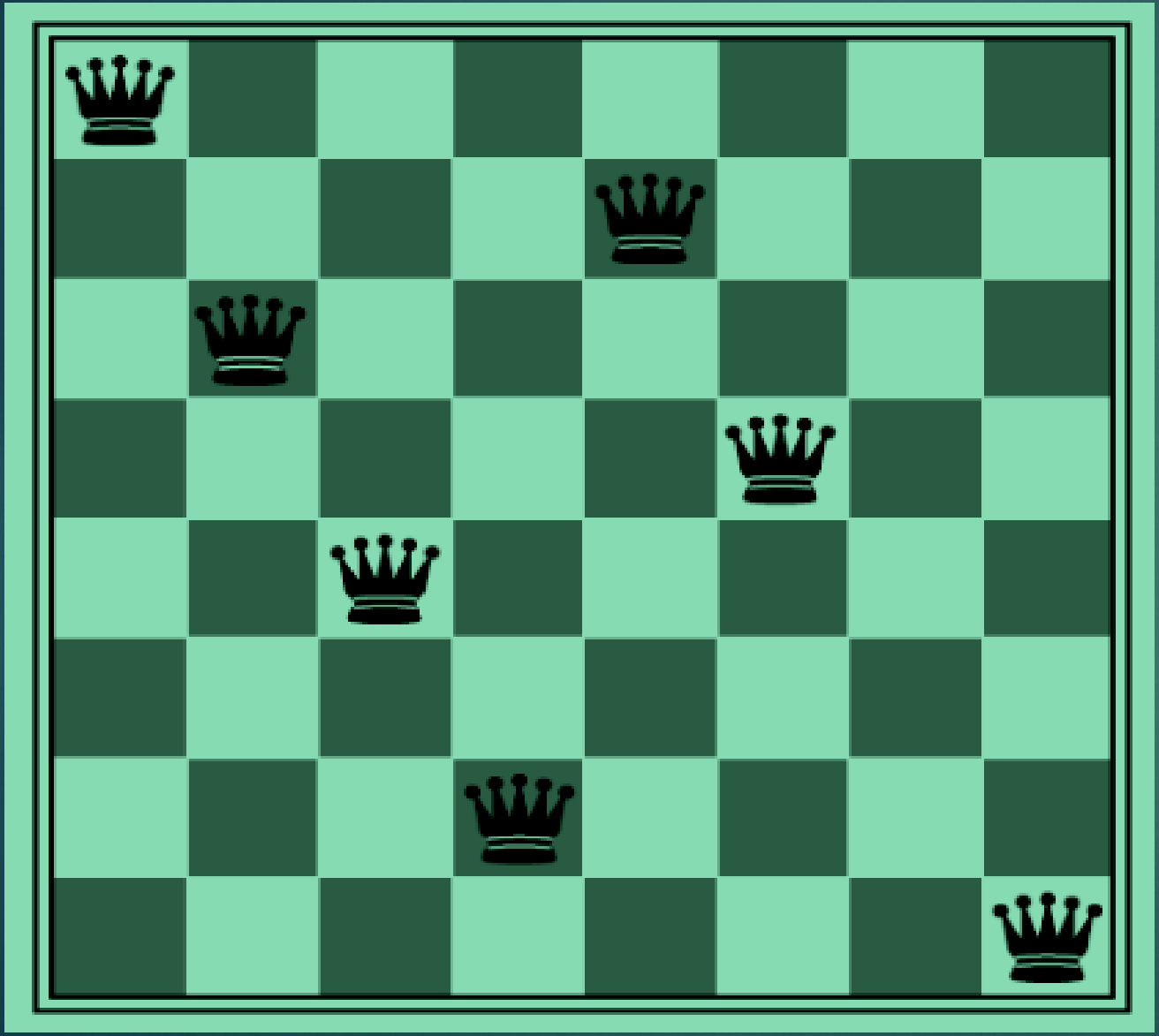


- Which are often used as test problems for new search algorithms in AI.
- An instance is shown in Figure.
- Consists of a 3×3 board with eight numbered tiles and a blank space.
- A tile adjacent to the blank space can slide into the space.
- The object is to reach a specified goal state, such as the one shown on the right of the figure.
- Sliding-block puzzles - NP-complete, so one does not expect to find methods significantly better in the worst case than the search algorithms described here.

8-puzzle: The standard formulation

- **States**: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- **Initial state**: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states
- **Actions**: The simplest formulation defines the actions as movements of the blank space *Left*, *Right*, *Up*, or *Down*. Different subsets of these are possible depending on where the blank is.
- **Transition model**: Given a state and action, this returns the resulting state; for example, if we apply *Left* to the start state in Figure the resulting state has the 5 and the blank switched.
- **Goal test**: This checks whether the state matches the goal configuration shown in Figure
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

8-queens problem



Goal: is to place eight queens on a chessboard such that no queen attacks any other.
(A queen attacks any piece in the same row, column or diagonal.)

An attempted solution that fails: the queen in the rightmost column is attacked by the queen at the top left.

8-queens problem

1. Incremental formulation

- Involves operators that *augment* the state description, starting with an empty state.
- Each action adds a queen to the state.

2. A complete-state formulation

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

The incremental formulation

States: Any arrangement of 0 to 8 queens on the board is a state.

Initial state: No queens on the board.

Actions: Add a queen to any empty square.

Transition model: Returns the board with a queen added to the specified square.

Goal test: 8 queens are on the board, none attacked.

Real-world problems

Route-finding problem

- Web sites
- In-car systems that provide driving directions
- Routing video streams in computer networks
- Military operations planning
- Airline travel-planning systems

Touring problems

Traveling salesperson problem (TSP)

VLSI layout problem

Robot navigation

Automatic assembly sequencing

Airline travel problems solved by a travel-planning web site

States: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these “historical” aspects.

Initial state: This is specified by the user’s query.

Actions: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.

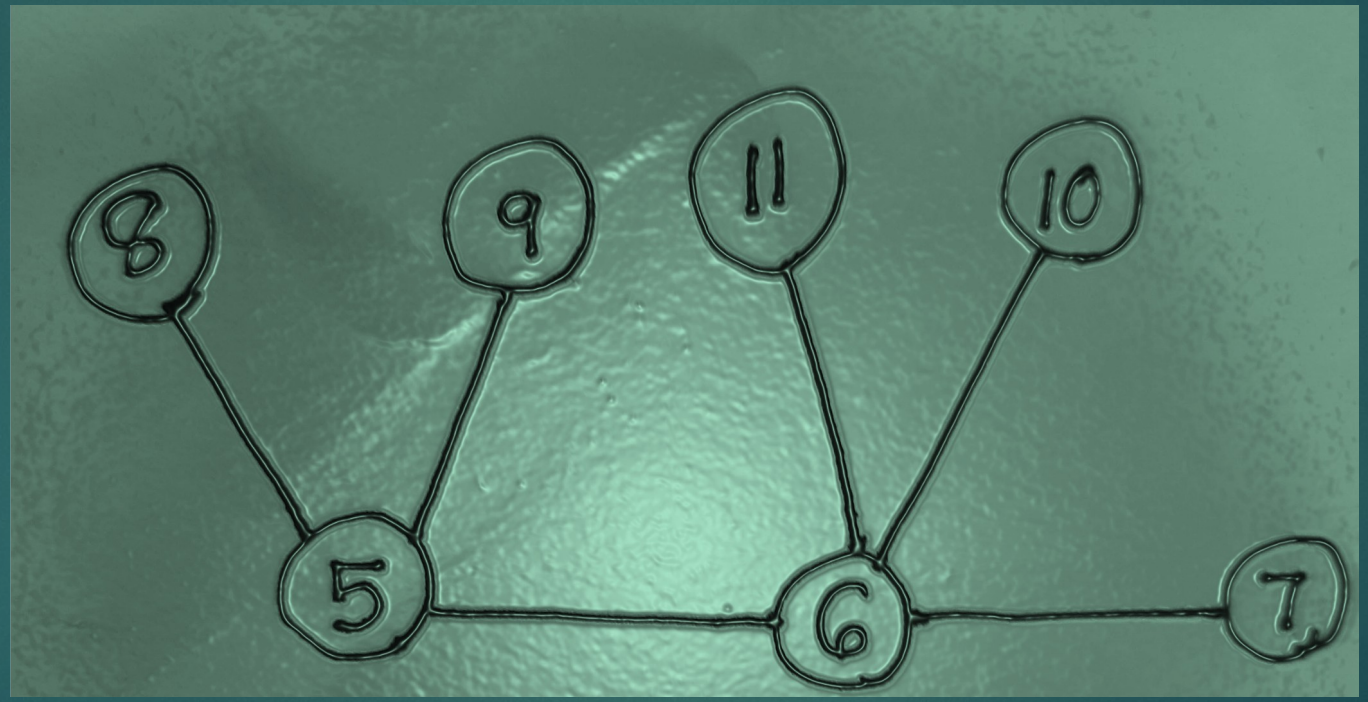
Transition model: The state resulting from taking a flight will have the flight’s destination as the current location and the flight’s arrival time as the current time.

Goal test: Are we at the final destination specified by the user?

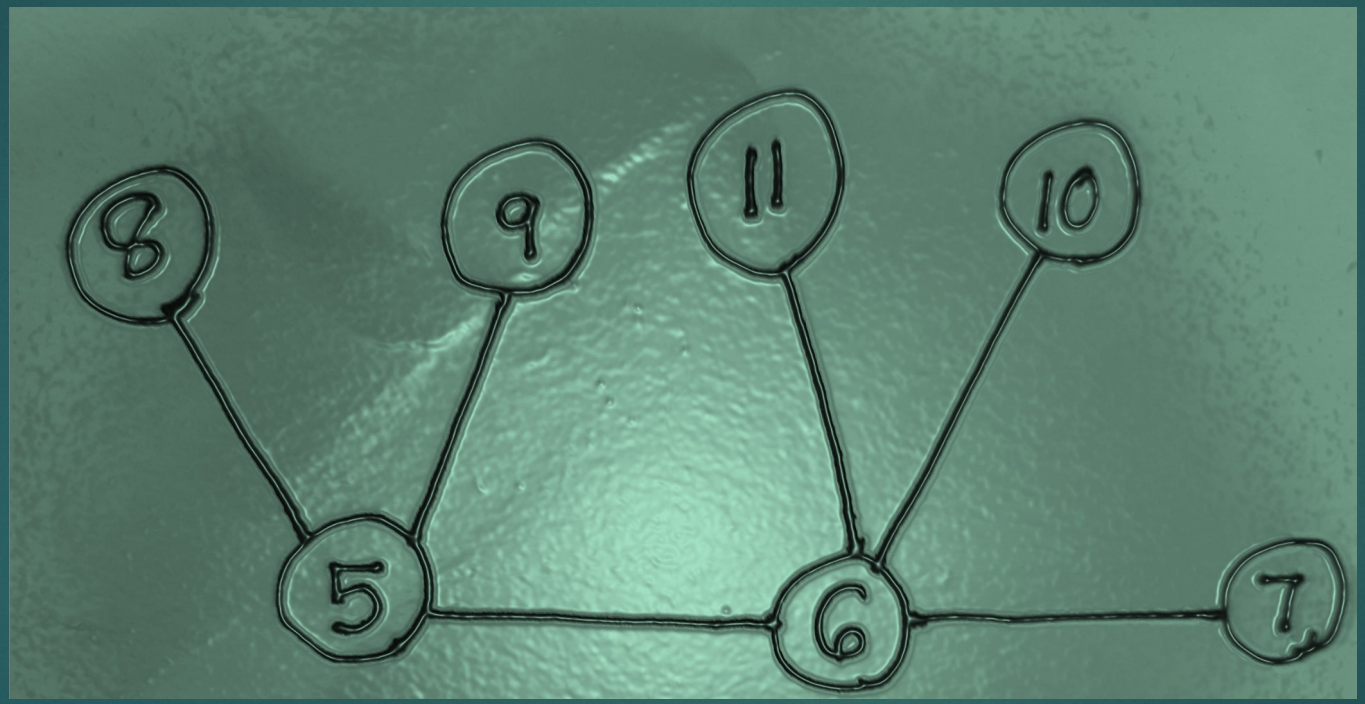
Path cost: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

BREADTH-FIRST-SEARCH (BFS)

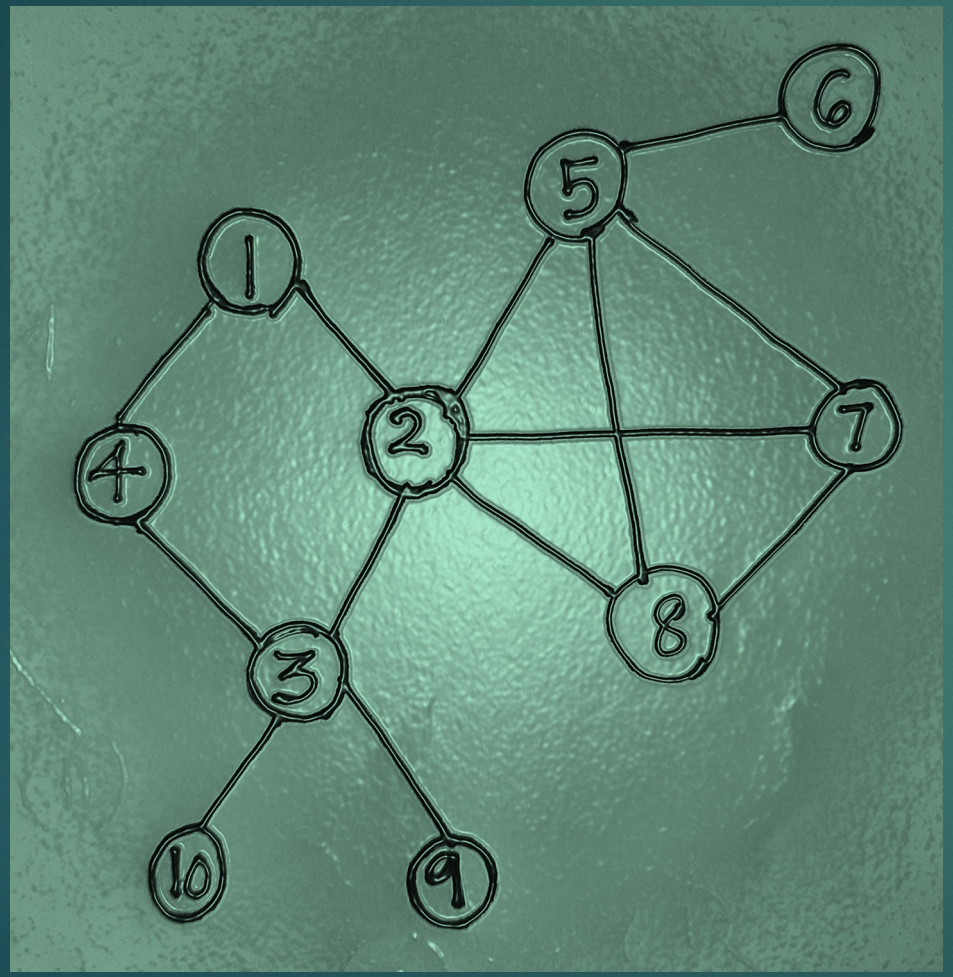
Visiting Vertex
Exploration of Vertex

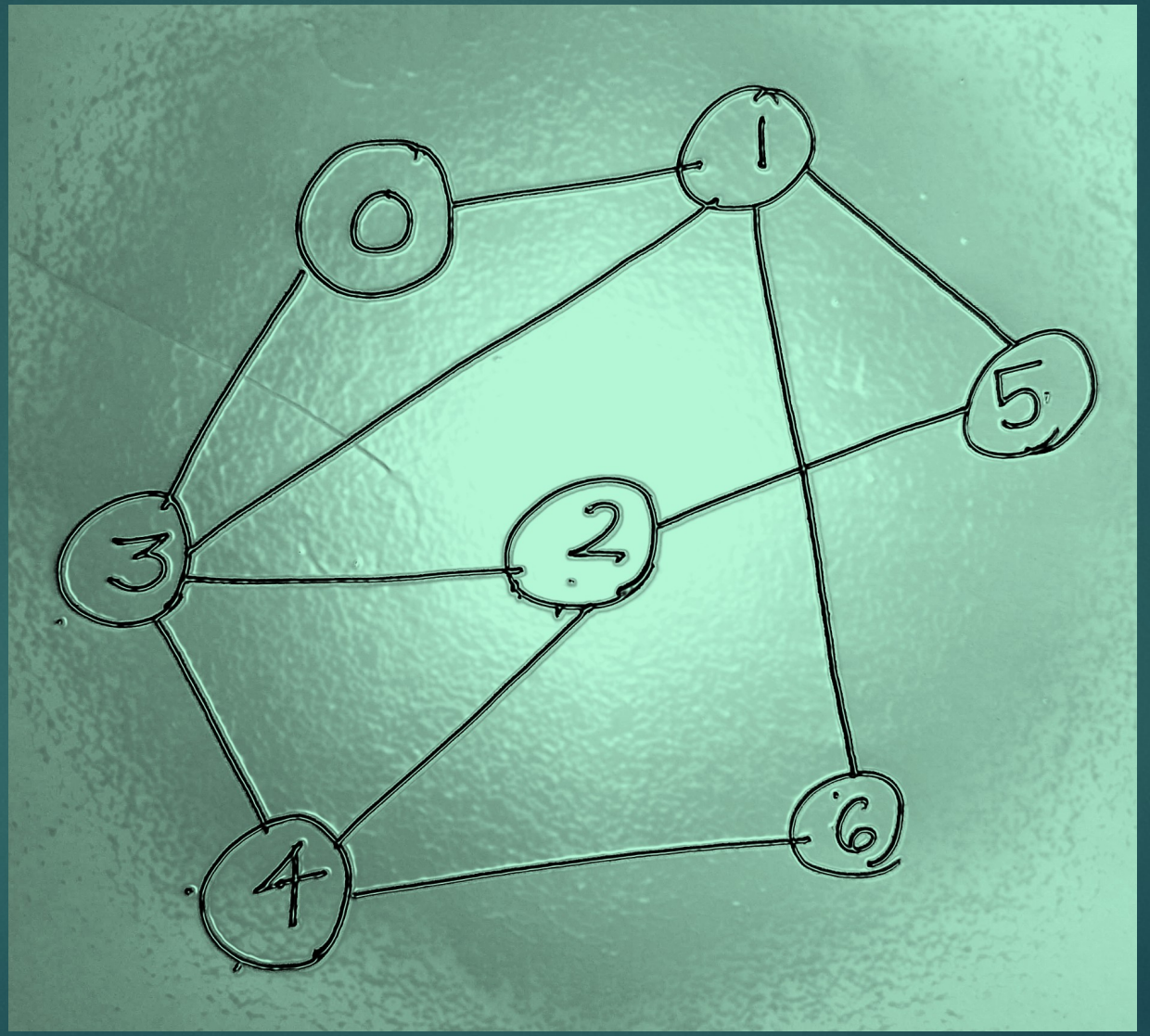


DEPTH-FIRST-SEARCH (DFS)

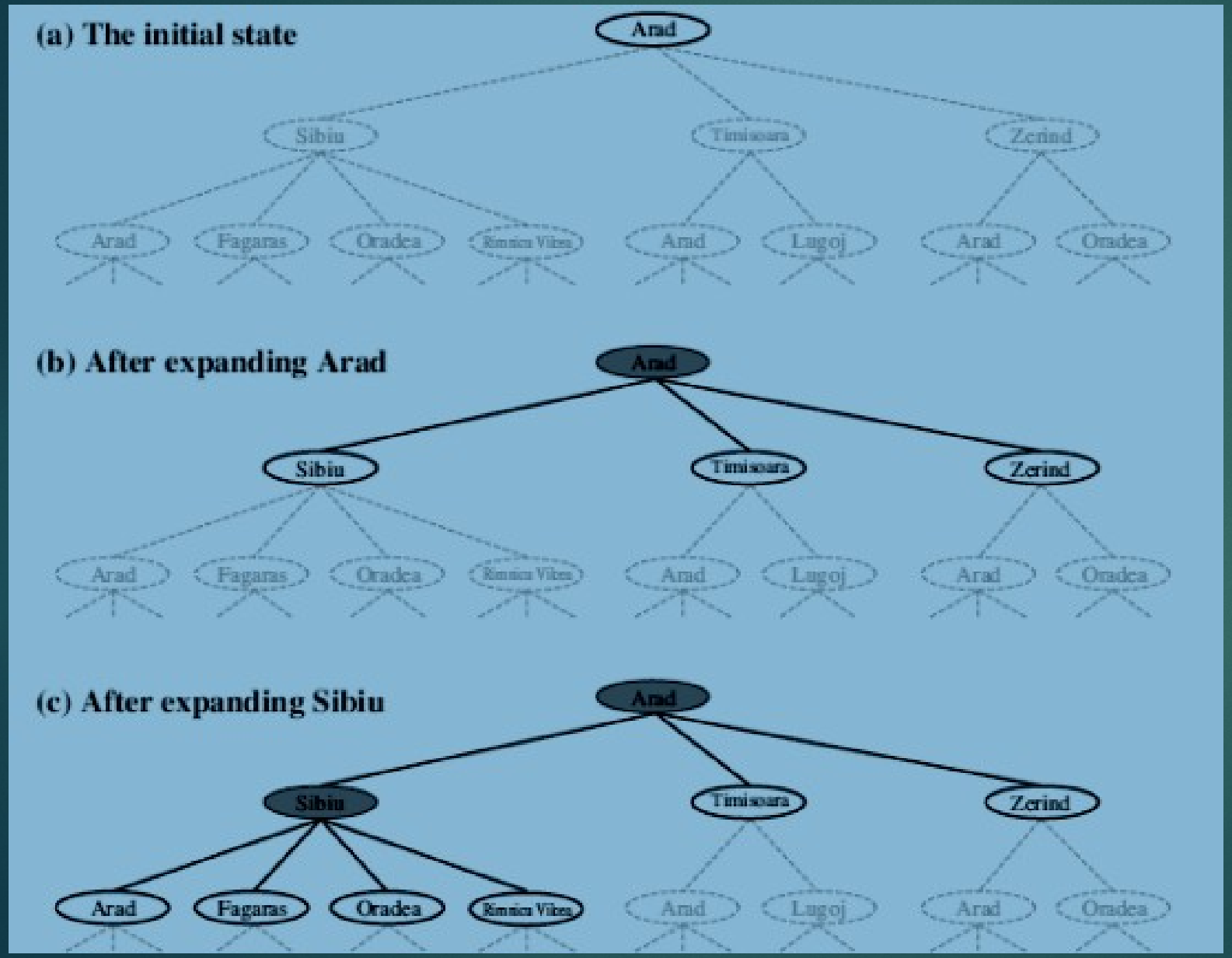


Perform BFS and DFS for the graph given below




0
1
2
3
4
5
6

Search Tree for finding Route



- Partial search trees for finding a route from Arad to Bucharest.
- Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

An informal description of the general tree-search and graph-search algorithms.

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

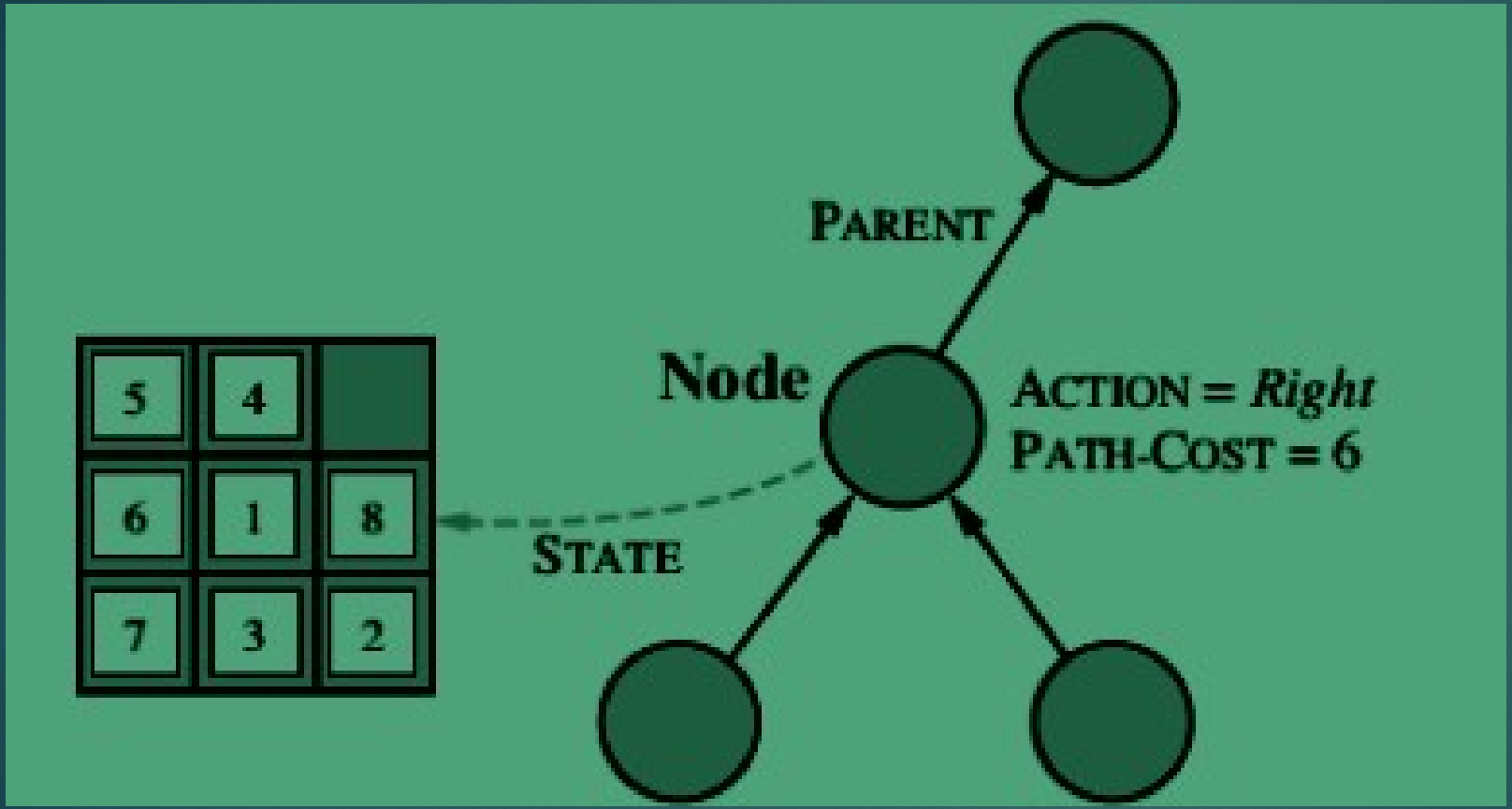
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Infrastructure for search algorithms

- Search algorithms require a data structure to keep track of the search tree that is being constructed.
- For each node *n* of the tree, we have a structure that contains four components:

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

Infrastructure for search algorithms (Cont'd)



- Nodes are the data structures from which the search tree is constructed.
- Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

The function CHILD-NODE takes a parent node and an action and returns the resulting child node.

```
function CHILD-NODE(problem, parent, action) returns a node
    return a node with
        STATE = problem.RESULT(parent.STATE, action),
        PARENT = parent, ACTION = action,
        PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

QUEUE

- The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy.
- The appropriate data structure for this is a queue.

The operations on a queue are as follows:

- **EMPTY?(queue):** returns true only if there are no more elements in the queue.
- **POP(queue) :** removes the first element of the queue and returns it.
- **INSERT(element, queue):** inserts an element and returns the resulting queue.

Measuring problem-solving performance

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

COMPLETENESS: Is the algorithm guaranteed to find a solution when there is one?

OPTIMALITY: Does the strategy find the optimal solution?

TIME COMPLEXITY: How long does it take to find a solution?

SPACE COMPLEXITY: How much memory is needed to perform the search?

Search Strategies

1. Uninformed search (Blind search)
2. Informed search (Heuristic search)

1. Uninformed search (Blind search)

- **The strategies have no additional information about states beyond that provided in the problem definition.**
- **All they can do is generate successors and distinguish a goal state from a non-goal state.**
- **All search strategies are distinguished by the *order* in which nodes are expanded.**

E.g. 1: Breadth-first search

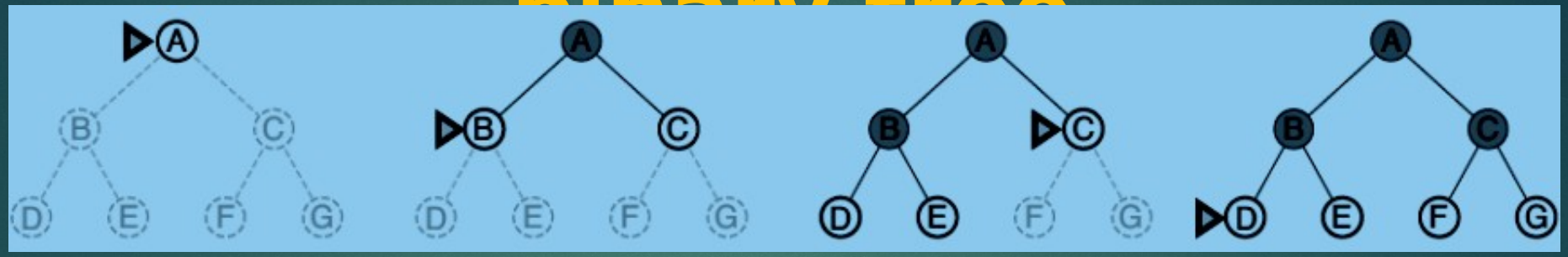
Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on.

All the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

Breadth-first search on a graph

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

Breadth-first search on a simple binary tree



- **At each stage, the node to be expanded next is indicated by a marker.**

Uniform-cost search

- When all step costs are equal, breadth-first search is optimal because it always expands the *shallowest* unexpanded node.
- By a simple extension, we can find an algorithm that is optimal with any step-cost function.
- Instead of expanding the shallowest node, **uniform-cost search** expands the node n with the *lowest path cost* $g(n)$.
- This is done by storing the frontier as a priority queue ordered by g.

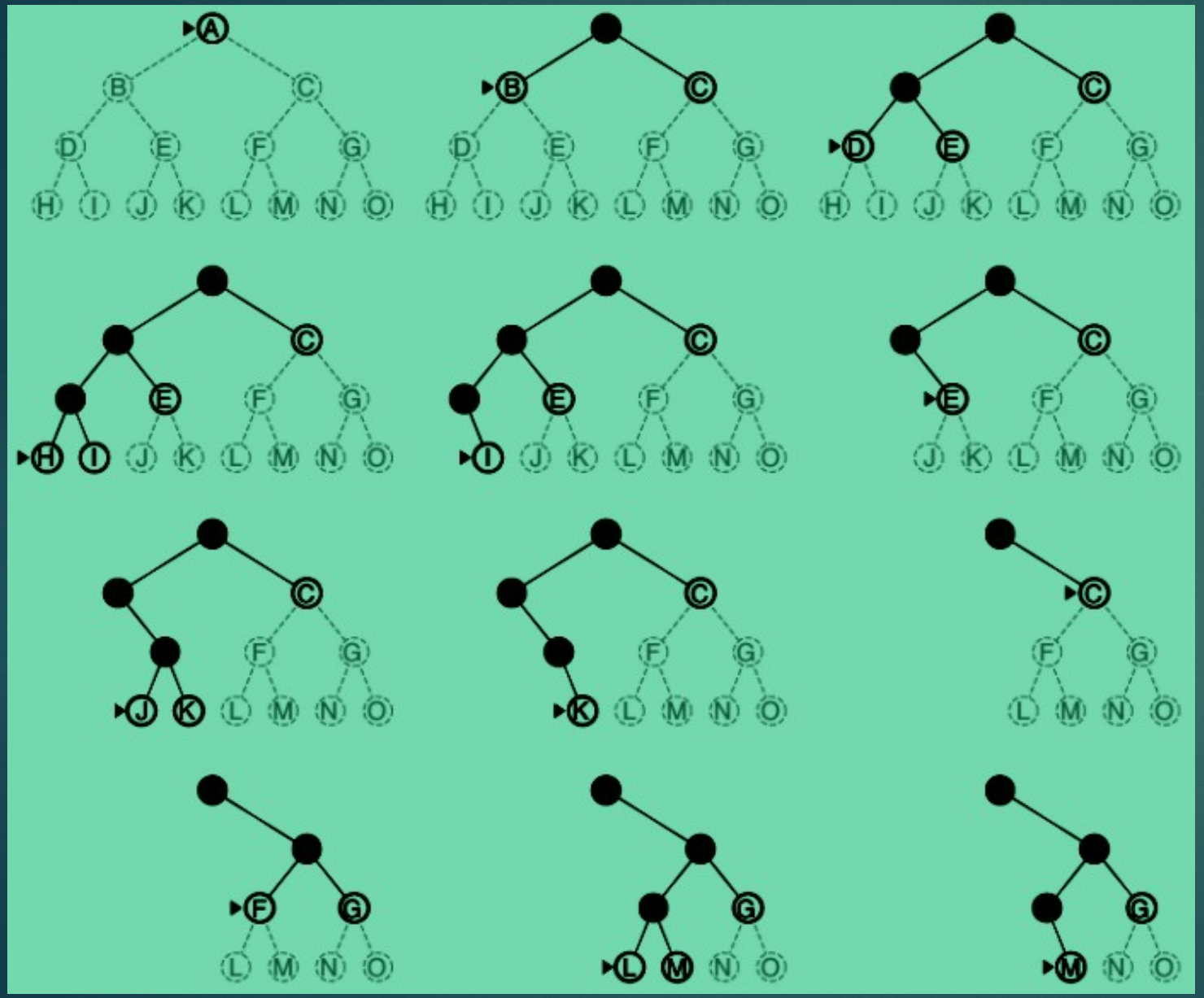
Uniform-cost search (Cont'd)

- Uniform-cost search does not care about the *number* of steps a path has, but only about their total cost.
- Uniform-cost search is guided by path costs rather than depths

E.g. 2: Depth-first search

- **Depth-first search** always expands the *deepest* node in the current frontier of the search tree.
- The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
- As those nodes are expanded, they are dropped from the frontier, so then the search "backs up" to the next deepest node that still has unexplored successors.
- Breadth-first-search uses a FIFO queue, depth-first search uses a LIFO queue.
- A LIFO queue means that the most recently generated node is chosen for expansion.
- This must be the deepest unexpanded node because it is one deeper than its parent —which, in turn, was the deepest unexpanded node when it was selected.

Depth-first search on a binary tree



- The unexplored region is shown in light Gray
- Explored nodes with no descendants in the frontier are removed from memory.
- Nodes at depth 3 have no successors and M is the only goal

Depth-limited search

- The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit.
- Nodes at depth are treated as if they have no successors.
- The depth limit solves the infinite-path problem.
- Depth-limited search will also be nonoptimal if we choose $l > d$.

Informed search (Heuristic search)

- Strategies that know whether one non-goal state is “more promising” than another are called **informed search** or **heuristic search** strategies.
- one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy.
- Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.

Best-first Search

- The general approach we consider is called **best-first search**.
- Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm
- A node is selected for expansion based on an **evaluation function**, *f(n)*.
- The evaluation function is construed as a cost estimate, so the node with the *lowest* evaluation is expanded first.
- The implementation of best-first graph search is identical to that for uniform-cost search, except for the use of *f* instead of *g* to order the priority queue.
- Most best-first algorithms include as a component of *f* a **heuristic function**, denoted *h(n)*

  h(n) = estimated cost of the cheapest path from the state at node *n* to a goal state.

Greedy best-first search

Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$.

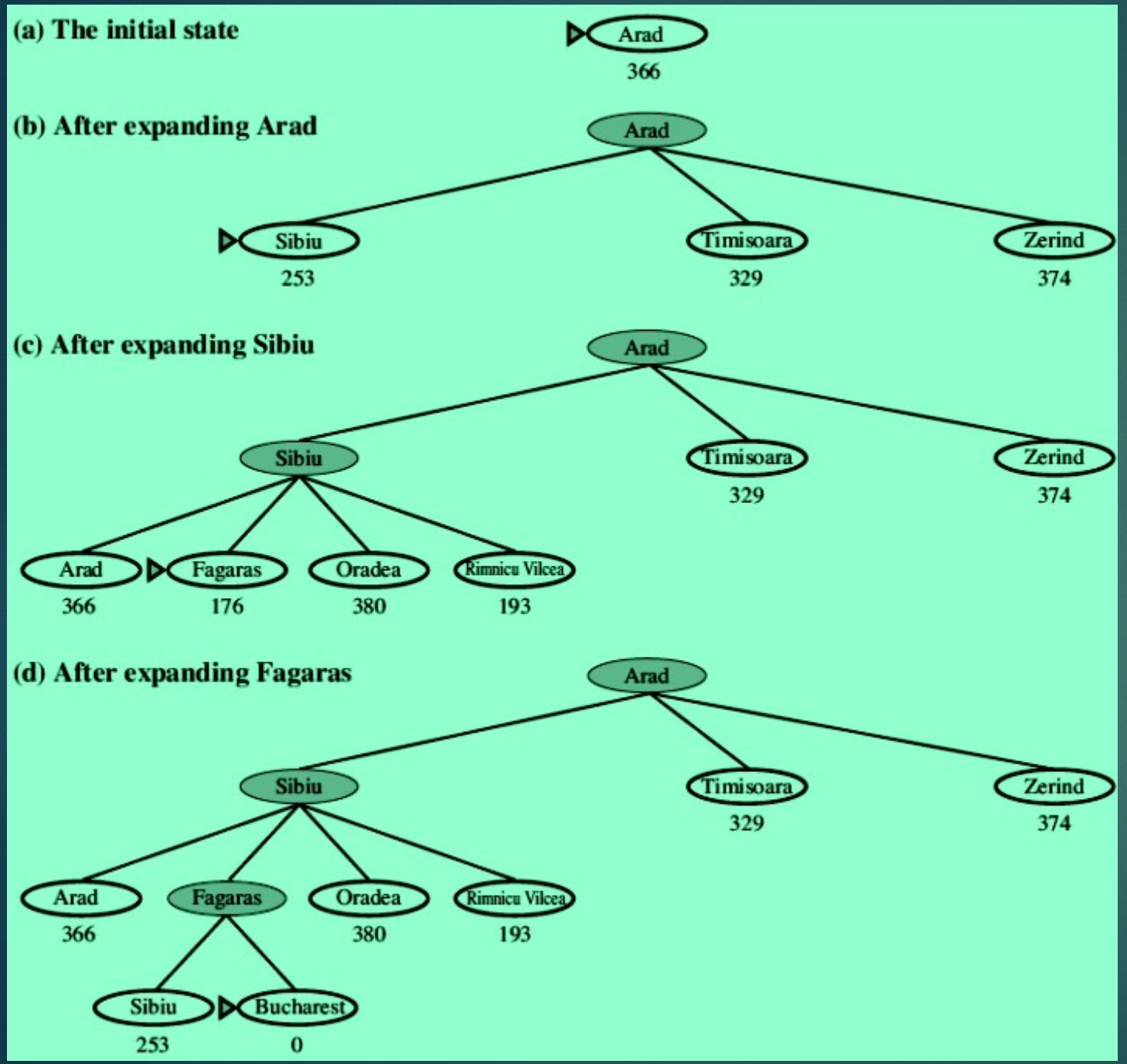
STRAIGHT- LINE DISTANCE HEURISTIC

- Route-finding problems in Romania - can use the straight line distance heuristic, h_{SLD}.
- If the goal is Bucharest, we need to know the straight-line distances to Bucharest. For example, h_{SLD}(In(Arad)) = 366.

Greedy best-first search using h_{SLD} to find a path from Arad to Bucharest

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Values of h_{SLD}—straight-line distances to Bucharest



Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic hSLD.

- Nodes are labelled with their h-values.
- Figure shows the progress of a greedy best-first search using h_{SLD} to find a path from Arad to Bucharest.
- The first node to be expanded from Arad will be Sibiu because it is closer to Bucharest than either Zerind or Timisoara.
- The next node to be expanded will be Fagaras because it is closest.
- Fagaras in turn generates Bucharest, which is the goal.
- For this particular problem, greedy best-first search using h_{SLD} finds a solution without ever expanding a node that is not on

A* search: Minimizing the total estimated solution cost

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

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

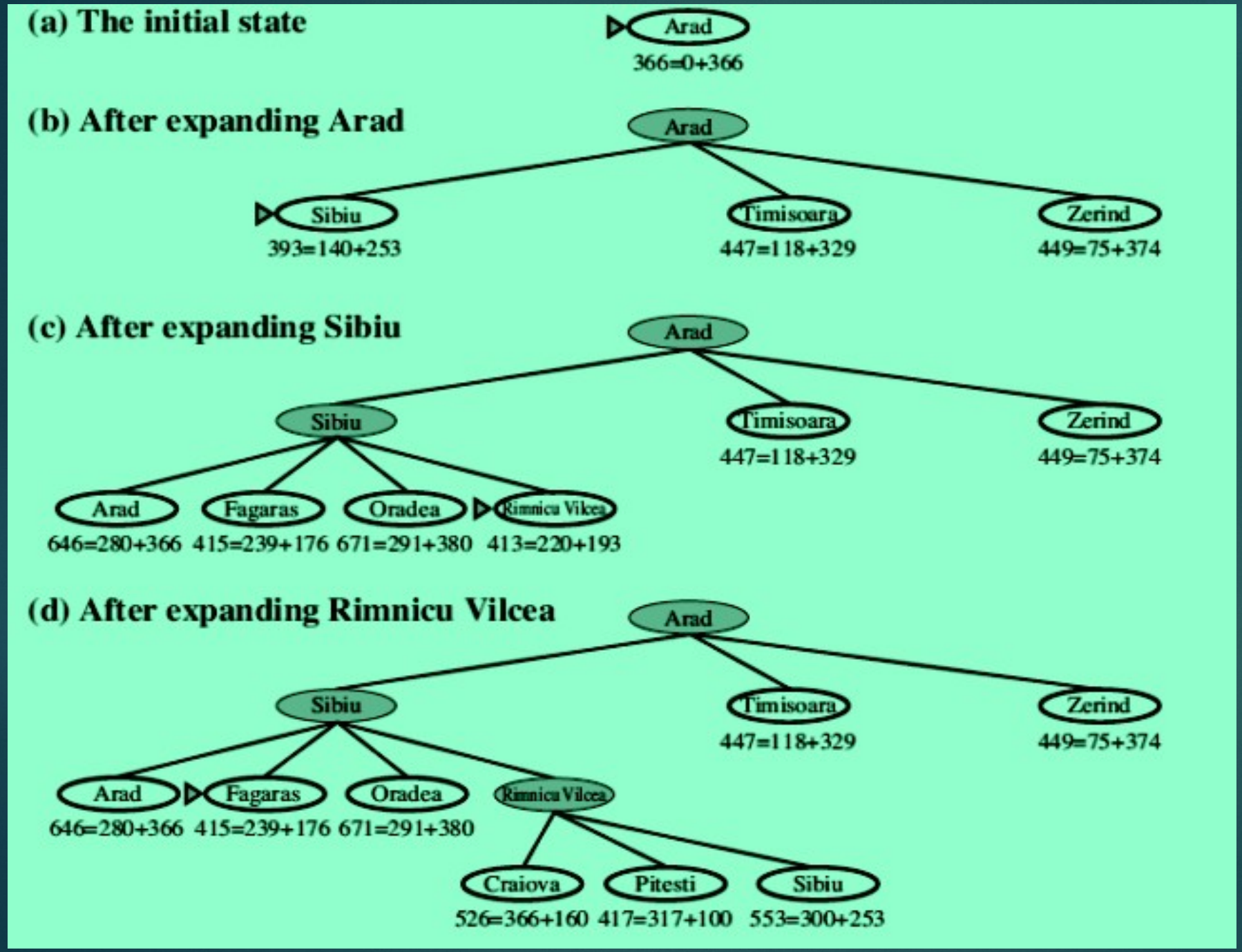
g(n) gives the path cost from the start node to node *n*,

h(n) is the estimated cost of the cheapest path from *n* to the goal,

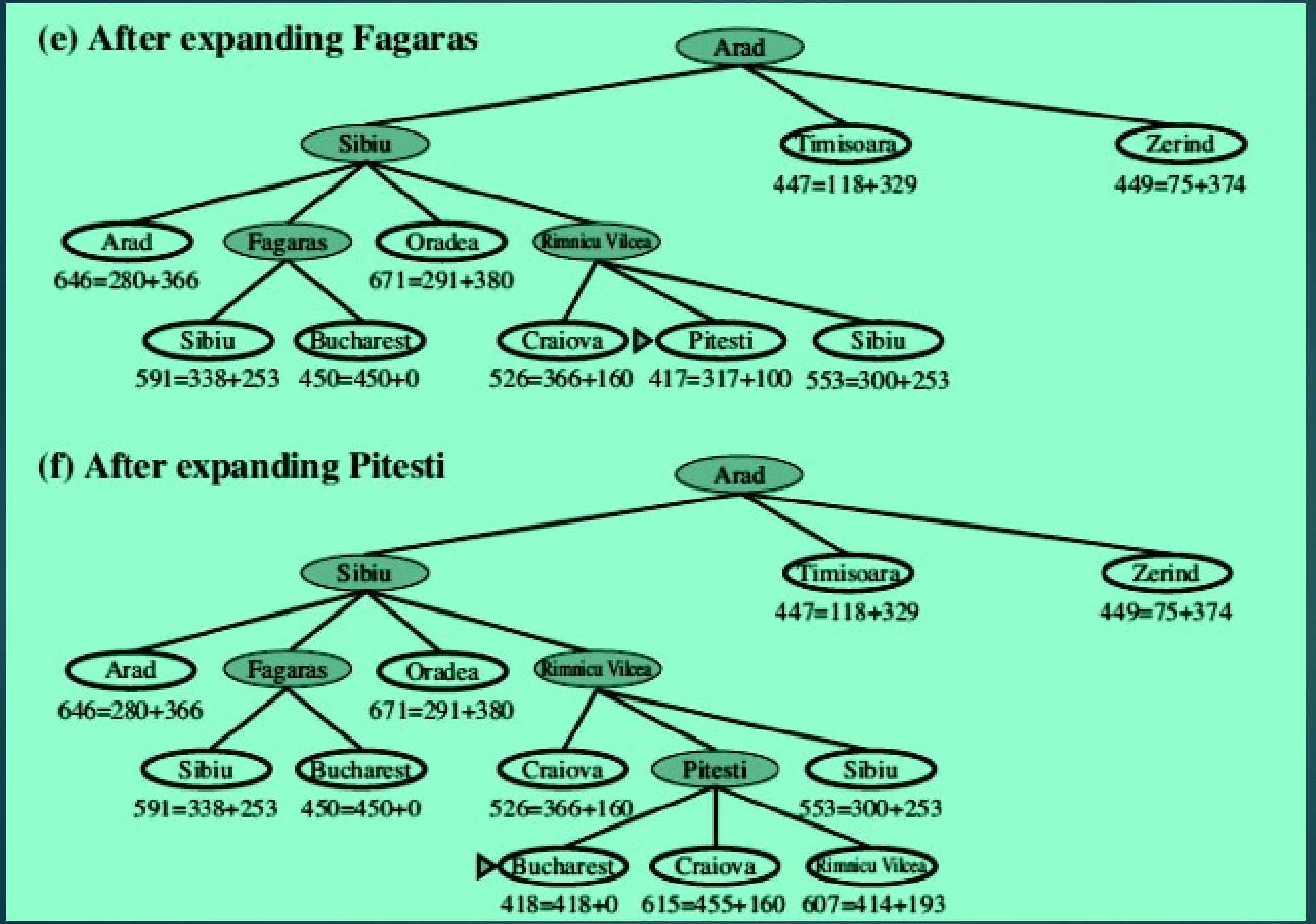
f(n) = estimated cost of the cheapest solution through *n*

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

A * search is **both complete and optimal.**

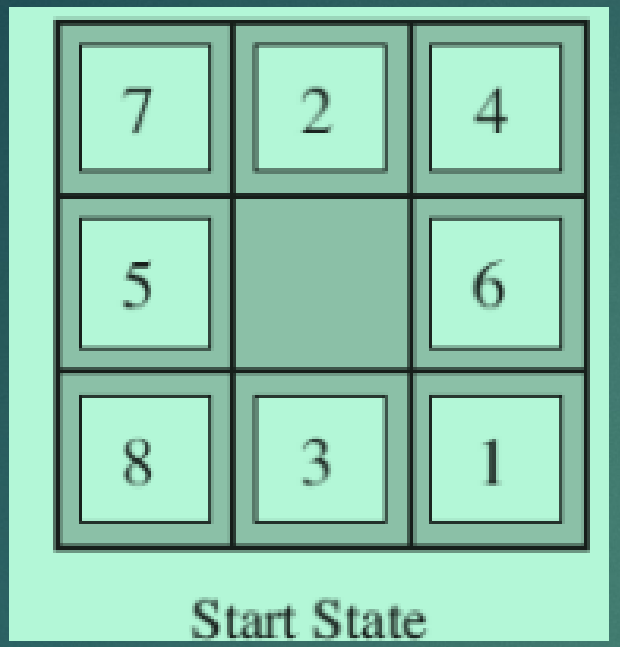


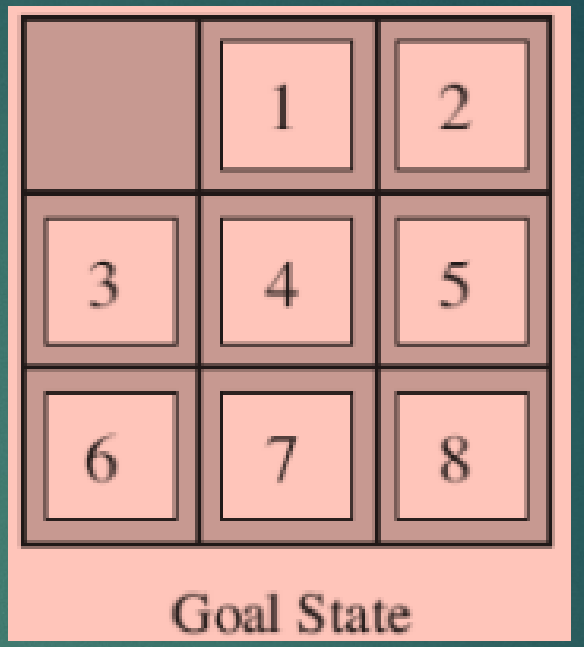
- Stages in an A* search for Bucharest.
- Nodes are labeled with $f = g + h$.
- The h values are the straight-line distances to Bucharest



- Stages in an A* search for Bucharest.
- Nodes are labeled with $f = g + h$.
- The h values are the straight-line distances to Bucharest

Heuristic functions





A typical instance of the 8-puzzle. The solution is 26 steps long.

Heuristics for the 8-puzzle

- Object of the puzzle - slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration.
- The average solution cost for a randomly generated 8-puzzle instance is about 22 steps. The branching factor is about 3.

Heuristic functions

- Two commonly used candidates (heuristics) for the 8-puzzle
  - **h_1 = the number of misplaced tiles.**

For Figure given in above all of the eight tiles are out of position, so the start state would have $h_1 = 8$. h_1 is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once.

- **h_2 = the sum of the distances of the tiles from their goal positions.**

Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances. This is sometimes called the **city block distance** or **Manhattan distance**.

h_2 is also admissible because all any move can do is move one tile one step closer to the goal.

Tiles 1 to 8 in the start state give a Manhattan distance of

$h_2 = 3+1 + 2 + 2+ 2 + 3+ 3 + 2 = 18.$

As expected, neither of these overestimates the true solution cost, which is 26.

References:

Stuart Russell and Peter Norvig, "Artificial intelligence-a modern approach 3rd ed." (2016).

ARTIFICIAL INTELLIGENCE

INTERIM SEMESTER 2021-22 BPL

CSE3007-LT-AB306

FACULTY: SIMI V.R.

Adversarial search

- **Competitive** environments
- which the agents' goals are in conflict
- giving rise to **adversarial search** problems—often known as **games**.
- Games, are interesting *because* they are too hard to solve.
- Optimal move and an algorithm for finding it
- **Pruning** allows us to ignore portions of the search tree that make no difference to the final choice
- **Heuristic evaluation functions** allow us to approximate the true utility of a state without doing a complete search.

Games

A game can be formally defined as a kind of search problem with the following elements:

- S_0: The initial state, which specifies how the game is set up at the start.
- PLAYER(*s*): Defines which player has the move in a state.
- ACTIONS(*s*): Returns the set of legal moves in a state.
- RESULT(*s*, *a*): The transition model, which defines the result of a move.
- TERMINAL-TEST(*s*): A terminal test, which is true when the game is over and false otherwise.
- States where the game has ended are called terminal states.
- UTILITY(*s*, *p*): A utility function (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p.

In chess, the outcome is a win, loss, or draw, with values +1, 0, or . Some games have a wider variety of possible outcomes; the payoffs in backgammon range from 0 to +192.

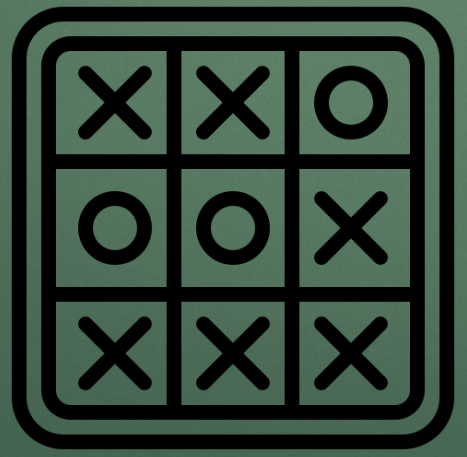
Game tree

- A **zero-sum game** is defined as one where the total payoff to all players is the same for every instance of the game.
- Chess is zero-sum because every game has payoff of either 0 + 1, 1 + 0 or + .

Game Tree

- The initial state, **ACTIONS** function, and **RESULT** function define the **game tree** for the game.
- The nodes are game states and the edges are moves.

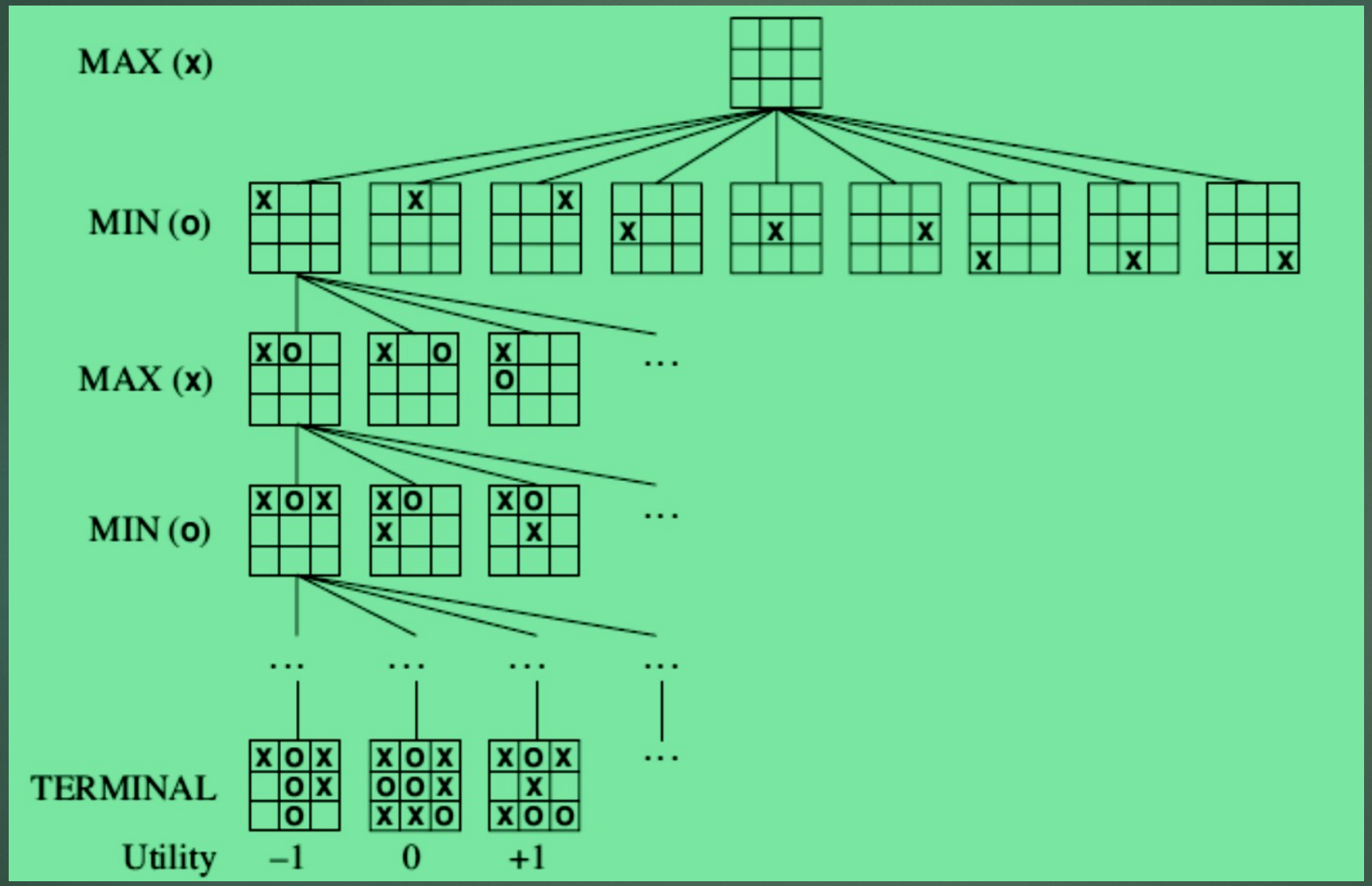
Game tree for tic-tac-toe (noughts and crosses)



Game tree for tic-tac-toe (Cont'd)

- From the initial state, MAX has nine possible moves.
- Play alternates between MAX's placing an X and MIN's placing an O until we reach leaf nodes corresponding to terminal states such that one player has three in a row or all the squares are filled.
- The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX; high values are assumed to be good for MAX and bad for MIN (which is how the players get their names).
- For tic-tac-toe the game tree is relatively small—fewer than 9! = 362, 880 terminal nodes.

A (partial) game tree for the game of tic-tac-toe.



- The top node is the initial state, and MAX moves first, placing an X in an empty square.
- We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

Optimal decisions in games

Two-ply game tree

- An optimal strategy leads to outcomes at least as good as any other strategy when one is playing an infallible opponent.
- The possible moves for MAX at the root node are labelled *a1*, *a2*, and *a3*. The possible replies to *a1* for MIN are *b1*, *b2*, *b3*, and so on.
- This particular game ends after one move each by MAX and MIN.
- The utilities of PLY the terminal states in this game range from 2 to 14.

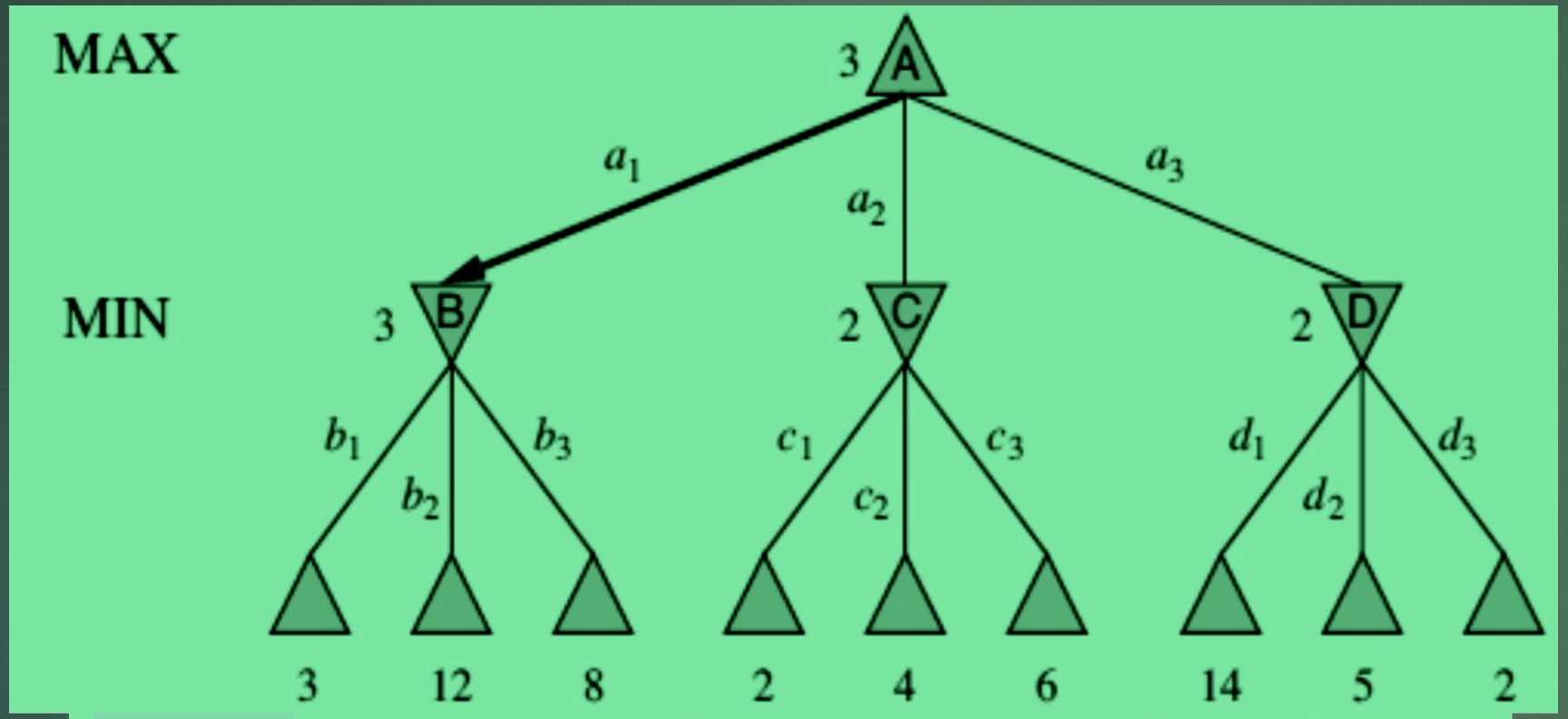
PLY MINIMAX VALUE

- Given a game tree, the optimal strategy can be determined from the **minimax value** of each node, which we write as MINIMAX(*n*).
- The minimax value of a node is the utility (for MAX) of being in the corresponding state, *assuming that both players play optimally* from there to the end of the game.
- The minimax value of a terminal state is just its utility.
- MAX prefers to move to a state of maximum value, whereas MIN prefers a state of minimum value.

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

- MINIMAX algorithm – Back tracking algorithm
- Best move strategy

A two-ply game tree



A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the state with the highest minimax value, and MIN's best reply is b_1, because it leads to the state with the lowest minimax value.

- The terminal nodes on the bottom level get their utility values from the game's UTILITY function.
- The first MIN node, labelled B, has three successor states with values 3, 12, and 8, so its minimax value is 3.
- Similarly, the other two MIN nodes have minimax value 2.
- The root node is a MAX node; its successor states have minimax values 3, 2, and 2; so it has a minimax value of 3.
- MINIMAX DECISION at the root: action a_1 is the optimal choice for MAX because it leads to the state with the highest minimax value.

The minimax algorithm

- The **minimax algorithm** computes the minimax decision from the current state.
- It uses a simple recursive computation of the minimax values of each successor state, directly implementing the defining equations.
- The recursion proceeds all the way down to the leaves of the tree, and then the minimax values are **backed up** through the tree as the recursion unwinds.
- For example, in Figure given above, the algorithm first recurses down to the three bottom left nodes and uses the UTILITY function on them to discover that their values are 3, 12, and 8, respectively.
- Then it takes the minimum of these values, 3, and returns it as the backed up value of node B.
- A similar process gives the backed-up values of 2 for C and 2 for D.
- Finally, we take the maximum of 3, 2, and 2 to get the backed-up value of 3 for the root node.
- The minimax algorithm performs a complete depth-first exploration of the game tree.

The minimax algorithm (cont'd)

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

An algorithm for calculating minimax decisions.

- It returns the action corresponding
- to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility.
- The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state.

$\text{argmax}_{a \in S} f(a)$

The notation computes the element a of set S that has the maximum value of *f(a)*.

Pruning

- The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree.
- Unfortunately, we can’t eliminate the exponent, but it turns out we can effectively cut it in half.
- **Pruning** eliminate large parts of the tree from consideration.

ALPHA–BETA

- When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

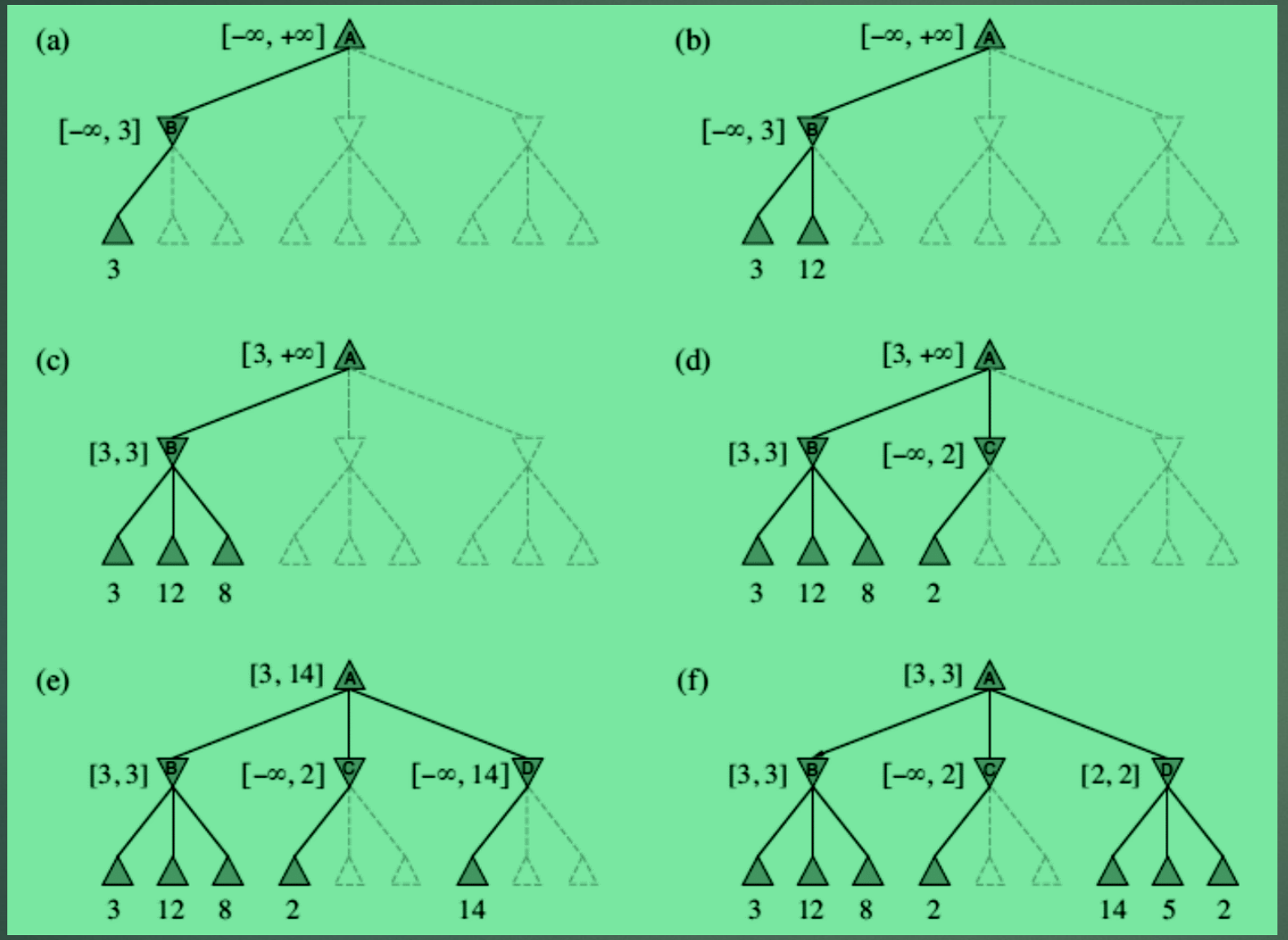


Figure 1: Stages in the calculation of the optimal decision for the game tree

At each point, we show the range of possible values for each node.

(a)The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3.

(b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3.

(c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3.

(d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning.

(e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14.

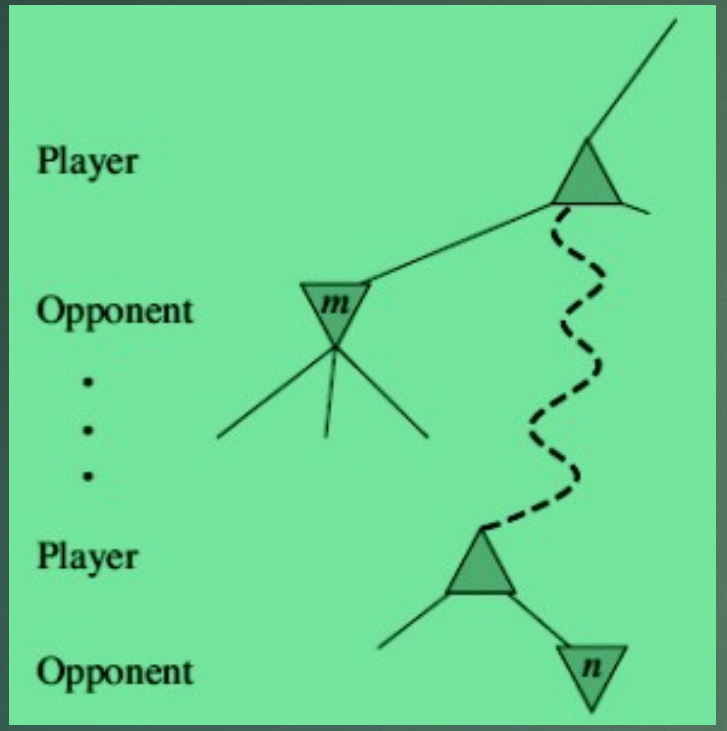
(f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

The value of the root node is given by

$$\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8),\min(2,x,y),\min(14,5,2)) \\
&= \max(3,\min(2,x,y),2) \\
&= \max(3,z,2) \qquad \text{where } z = \min(2,x,y) \le 2 \\
&= 3.
\end{aligned}$$

The value of the root and hence the minimax decision are *independent* of the values of the pruned leaves x and y.

General Case for alpha-beta pruning



If m is better than n for Player, we will never get to n in play.

Alpha–beta pruning gets its name from the following two parameters that describe bounds on the backed-up values that appear anywhere along the path:

$\alpha =$ the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

$\beta =$ the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

Alpha–beta search updates the values of α and β as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

The alpha–beta search algorithm.

Notice that these routines are the same as the MINIMAX functions except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Knowledge Representation

KNOWLEDGE BASE

- The central component of a knowledge-based agent is its **knowledge base**, or KB.
- A knowledge base is a set of **sentences**. (Here "sentence" is used as a technical term. It is related but not identical to the sentences of English and other natural languages.)

KNOWLEDGE REPRESENTATION LANGUAGE

- Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world.
- Sometimes we dignify a sentence with the name **axiom**, when the sentence is taken as given without being derived from other sentences.

INFERENCE

- There must be a way to add new sentences to the knowledge base and a way to query what is known.
- The standard names for these operations are TELL and ASK, respectively.
- Both operations may involve **inference**—that is, deriving new sentences from old.
- Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told to the knowledge base previously.

A generic knowledge-based agent

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

The details of the representation language are hidden inside **three functions** that implement the interface between the sensors and actuators on one side and the core representation and reasoning system on the other.

- **MAKE-PERCEPT-SENTENCE** constructs a sentence asserting that the agent perceived the given percept at the given time.
- **MAKE-ACTION-QUERY** constructs a sentence that asks what action should be done at the current time.
- **MAKE-ACTION-SENTENCE** constructs a sentence asserting that the chosen action was executed.

Logic

- Logic is used to represent simple facts.
- Logic defines the ways of putting symbols together to form sentences that represent facts.
- Sentences are either true or false but not both are called propositions.

Examples :

Sentence	Truth value	Is it a Proposition ?
"Grass is green"	"true"	Yes
"2 + 5 = 5"	"false"	Yes
"Close the door"	-	No
"Is it hot out side?"	-	No
"x > 2"	-	No (since x is not defined)
"x = x"	-	No (don't know what is "x" and "=" mean; "3 = 3" or say "air is equal to air" or "Water is equal to water" has no meaning)

Propositional Logic (PL)

$$
\begin{aligned}
\mathit{Sentence} &\rightarrow \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} &\rightarrow \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} &\rightarrow (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
&\mid \neg\, \mathit{Sentence} \\
&\mid \mathit{Sentence} \wedge \mathit{Sentence} \\
&\mid \mathit{Sentence} \vee \mathit{Sentence} \\
&\mid \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
&\mid \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
\text{OPERATOR PRECEDENCE} &: \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Propositional Logic (PL) (Cont'd)

- **A proposition is a statement - which in English is a declarative sentence and Logic defines the ways of putting symbols together to form sentences that represent facts.**
- **Every proposition is either true or false.**
- **Propositional logic is also called Boolean algebra.**

Examples: (a) The sky is blue., (b) Snow is cold. , (c) 12 * 12=144

Propositional logic : It is fundamental to all logic.

- Propositions are "Sentences"; either true or false but not both.
- A sentence is smallest unit in propositional logic
- f proposition is true, then truth value is "true"; else "false"

Example: Sentence "Grass is green";

Truth value " true";

Proposition "yes"

Statement, Variables and Symbols

Statement : A simple statement is one that does not contain any other statement as a part. A compound statement is one that has two or more simple statements as parts called components.

Operator or connective : Joins simple statements into compounds, and joins compounds into larger compounds.

Symbols for connectives

assertion	P					"p is true"
nagation	¬p	~	!		NOT	"p is false"
conjunction	p ∧ q	·	&&	&	AND	"both p and q are true"
disjunction	P v q	||	|		OR	"either p is true, or q is true, or both "
implication	p → q	⊃	⇒		if . . then	"if p is true, then q is true" " p implies q "
equivalence	↔	≡	⇔		if and only if	"p and q are either both true or both false"

Truth table

- **Is a convenient way of showing relationship between several propositions.**

The truth table for negation, conjunction, disjunction, implication and equivalence are shown below.

p	q	¬p	¬q	p ∧ q	p v q	p→q	p ↔ q	q→p
T	T	F	F	T	T	T	T	T
T	F	F	T	F	T	F	F	T
F	T	T	F	F	T	T	F	F
F	F	T	T	F	F	T	T	T

Tautology

A Tautology is proposition formed by combining other propositions (p, q, r, . . .) which is true regardless of truth or falsehood of p, q, r, ...

The important tautologies are :

$(p \rightarrow q) \leftrightarrow \neg\, [p \wedge (\neg q)]$ and $(p \rightarrow q) \leftrightarrow (\neg p) \vee q$

A proof of these tautologies, using the truth tables are given below.

Tautologies $(p \rightarrow q) \leftrightarrow \neg\, [p \wedge (\neg q)]$ and $(p \rightarrow q) \leftrightarrow (\neg p) \vee q$

Table Proof of Tautologies

p	q	p→q	¬q	p ∧ (¬q)	¬ [p ∧ (¬q)]	¬p	(¬p) ∨ q
T	T	T	F	F	T	F	T
T	F	F	T	T	F	F	F
F	T	T	F	F	T	T	T
F	F	T	T	F	T	T	T

Predicate Logic

- **Predicate Logic/ Calculus extends the syntax of propositional calculus with predicates and quantifiers:**
- **P(X) – P is a predicate.**
- **First Order Predicate Logic allows predicates to apply to objects or terms, but not functions or predicates.**

- ∀ - For all:
  - ∀xP(x) is read "For all x'es, P (x) is true".
- ∃ - There Exists:
  - ∃x P(x) is read "there exists an x such that P(x) is true".
- Relationship between the quantifiers:
  - ∃x P(x) ≡ ¬(∀x)¬P(x)
  - "If There exists an x for which P holds, then it is not true that for all x P does not hold".

References:

Stuart Russell and Peter Norvig, "Artificial intelligence-a modern approach 3rd ed." (2016).

S. Rajasekaran and G.A. Vijayalaksmi Pai ,Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications, (2005), Prentice Hall.

Artificial Intelligence

INTERIM SEMESTER 2021-22 BPL

CSE3007-LT-AB306

FACULTY: SIMI V.R.

Machine Learning

Machine Learning Problems

- **Spam Detection**: Given email in an inbox, identify those email messages that are spam and those that are not. Having a model of this problem would allow a program to leave non-spam emails in the inbox and move spam emails to a spam folder.
- **Credit Card Fraud Detection**: Given credit card transactions for a customer in a month, identify those transactions that were made by the customer and those that were not. A program with a model of this decision could refund those transactions that were fraudulent.
- **Digit Recognition:** Given a zip codes hand written on envelops, identify the digit for each hand written character. A model of this problem would allow a computer program to read and understand handwritten zip codes and sort envelops by geographic region.
- **Speech Understanding:** Given an utterance from a user, identify the specific request made by the user. A model of this problem would allow a program to understand and make an attempt to fulfil that request. The iPhone with Siri has this capability.
- **Face Detection:** Given a digital photo album of many hundreds of digital photographs, identify those photos that include a given person. A model of this decision process would allow a program to organize photos by person. Some cameras and software like iPhoto has this capability.

Machine Learning Problems

- **Product Recommendation**: Given a purchase history for a customer and a large inventory of products, identify those products in which that customer will be interested and likely to purchase. A model of this decision process would allow a program to make recommendations to a customer and motivate product purchases. Amazon has this capability. Also think of Facebook, Google Plus and LinkedIn that recommend users to connect with you after you sign-up.
- **Medical Diagnosis**: Given the symptoms exhibited in a patient and a database of anonymized patient records, predict whether the patient is likely to have an illness. A model of this decision problem could be used by a program to provide decision support to medical professionals.
- **Stock Trading**: Given the current and past price movements for a stock, determine whether the stock should be bought, held or sold. A model of this decision problem could provide decision support to financial analysts.
- **Customer Segmentation**: Given the pattern of behaviour by a user during a trial period and the past behaviours of all users, identify those users that will convert to the paid version of the product and those that will not. A model of this decision problem would allow a program to trigger customer interventions to persuade the customer to covert early or better engage in the trial.
- **Shape Detection**: Given a user hand drawing a shape on a touch screen and a database of known shapes, determine which shape the user was trying to draw. A model of this decision would allow a program to show the platonic version of that shape the user drew to make crisp diagrams.

Types of Machine Learning Problems

There are common classes of problem in Machine Learning. The problem classes below are archetypes for most of the problems we refer to when we are *doing* Machine Learning.

- **Classification**: Data is labelled meaning it is assigned a class, for example spam/non-spam or fraud/non-fraud. The decision being modelled is to assign labels to new unlabelled pieces of data. This can be thought of as a discrimination problem, modelling the differences or similarities between groups.
- **Regression**: Data is labelled with a real value (think floating point) rather then a label. Examples that are easy to understand are time series data like the price of a stock over time, The decision being modelled is what value to predict for new unpredicted data.
- **Clustering**: Data is not labelled, but can be divided into groups based on similarity and other measures of natural structure in the data. An example from the above list would be organising pictures by faces without names, where the human user has to assign names to groups, like iPhoto on the Mac.
- **Rule Extraction**: Data is used as the basis for the extraction of propositional rules (antecedent/consequent like *if-then*). Such rules may, but are typically not directed, meaning that the methods discover statistically supportable relationships between attributes in the data, not necessarily involving something that is being predicted.

Handwritten Digit Recognition

Handwriting recognition (HWR), also known as Handwritten Text Recognition (HTR), is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices.

Important steps in the Algorithm

1. Preparation of dataset

-Ask different persons for writing digits from 0 to 9 in a paper.

- From each person 100 data can be collected.

- If 100 people is involving 10000 data can be prepared.

2. Scanning and digitizing of data

3. Pre-processing of data

- Improving the quality of images
- That may include denoising, contrast improvement and sharpening

5. Training phase CNN with input images and Target values

- 80% of total data will be used for training

6. Testing phase

- 20% of data can be used for testing.

Artificial Intelligence-C11

Fuzzy Reasoning

Interim Semester 2021-22 BPL

CSE3007-LT-AB306

Faculty: SIMI V.R.

Fuzzy Sets: Introduction

- Proposed by Lotfi A. Zadeh
- Generalization of classical set theory
- Uncertainty (Incomplete knowledge, generality, vagueness, ambiguity)
- Effective solving of uncertainty in the problem.
- A classical set is a set with a crisp boundary.
- An element : Belongs to / not belongs to a set (0 or 1)

  Eg.: C classical set A of real numbers greater than 6

  $$A = \{x \mid x>6\}$$

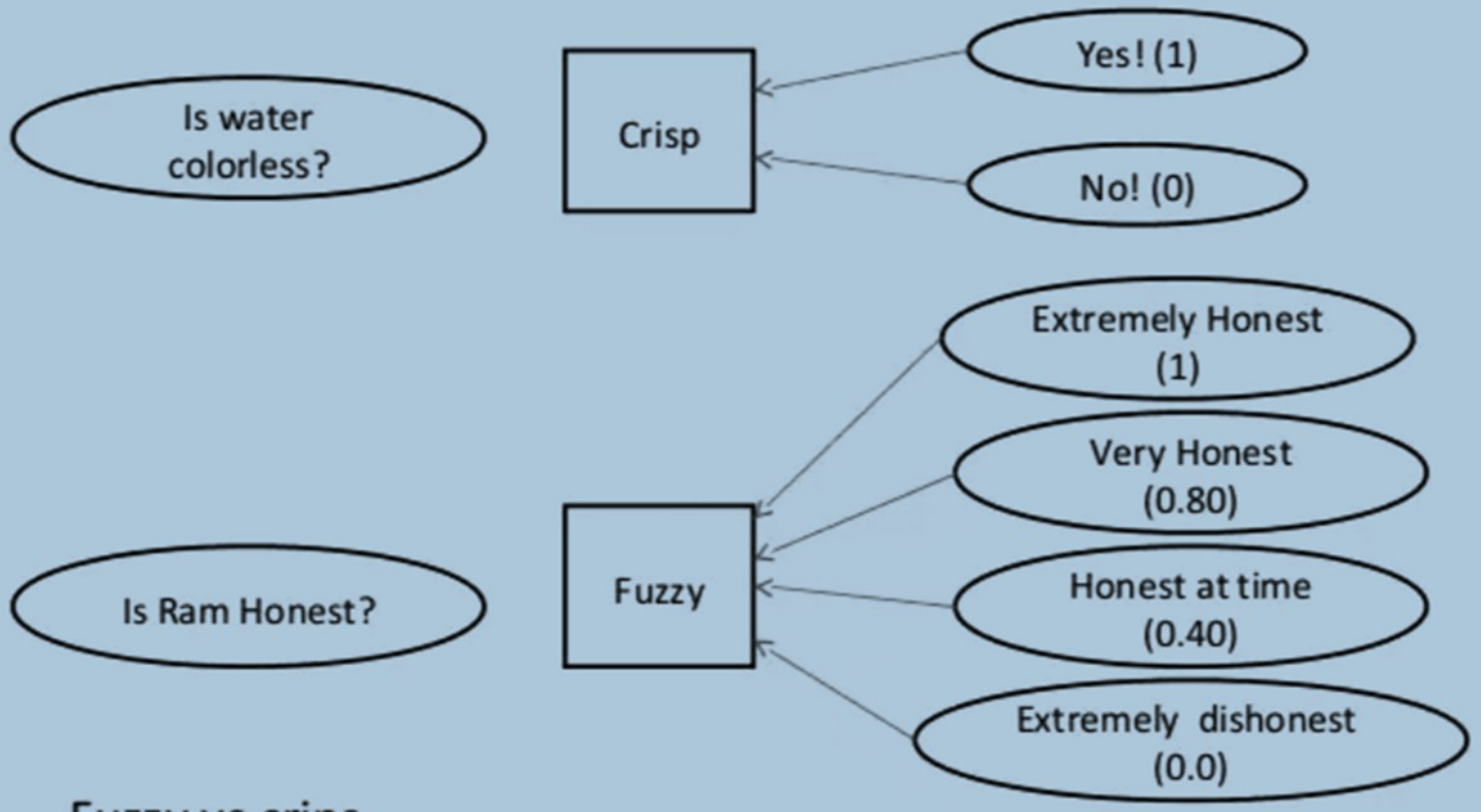
- Fuzzy set many degrees of membership (0 & 1)
- Membership function $\mu_A(x)$ – associated with a fuzzy set A such that the function maps every element of the universe of discourse X to the interval [0,1]

Classical set theory	Fuzzy set theory
• Classes of objects with sharp boundaries.	• Classes of objects with un-sharp boundaries.
• A classical set is defined by crisp(exact) boundaries, i.e., there is no uncertainty about the location of the set boundaries.	• A fuzzy set is defined by its ambiguous boundaries, i.e., there exists uncertainty about the location of the set boundaries.
• Widely used in digital system design	• Used in fuzzy controllers.

The areas of potential fuzzy implementation are numerous and not just for control:

- **Speech recognition**
- **fault analysis**
- **decision making**
- **image analysis**
- **scheduling**

Example

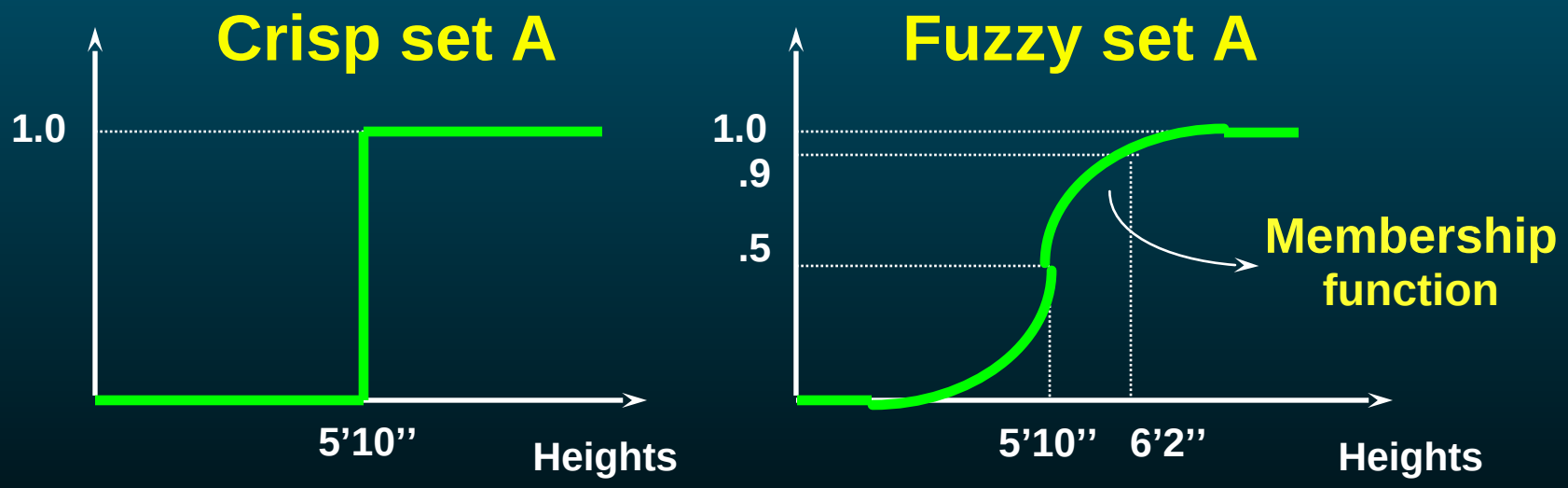


Fuzzy vs crips

Fuzzy Sets

Sets with fuzzy boundaries

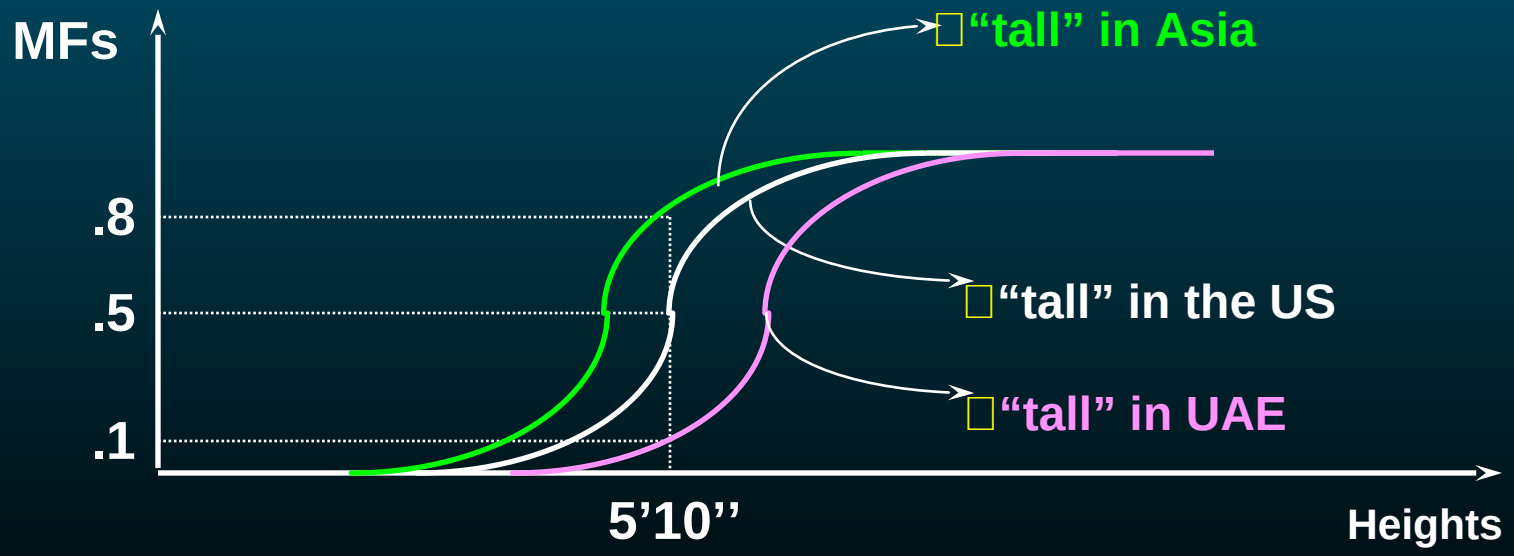
A = Set of tall people



Membership Functions (MFs)

Characteristics of MFs:

- Subjective measures
- Not probability functions



Fuzzy Sets

Formal definition:

A fuzzy set *A* in *X* is expressed as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\}$$

Fuzzy set

Membership function (MF)

Universe or universe of discourse

A fuzzy set is totally characterized by a membership function (MF).

Alternative Notation

A fuzzy set A can be alternatively denoted as follows:

X is discrete

$$A = \sum_{x_i \in X} \mu_A(x_i) / x_i$$

X is continuous

$$A = \int_X \mu_A(x) / x$$

- **Note that Σ and integral signs stand for the union of membership grades; "/" stands for a marker and does not imply division.**
- **Membership Function (MF) – Maps each element of X to a membership grade (membership value) between 0 and 1**

Fuzzy Sets with Discrete Universes

Fuzzy set C = "desirable city to live in"

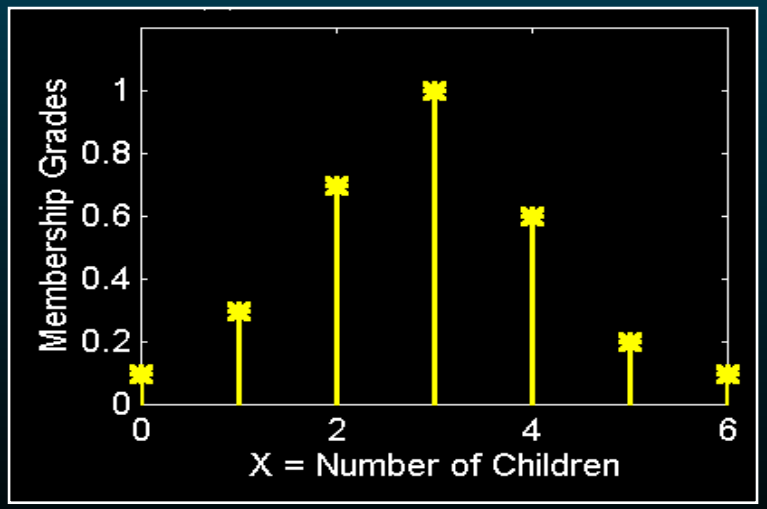
X = {SF, Boston, LA} (discrete and nonordered)

C = {(SF, 0.9), (Boston, 0.8), (LA, 0.6)}

Fuzzy set A = "sensible number of children"

X = {0, 1, 2, 3, 4, 5, 6} (discrete universe)

A = {(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)}



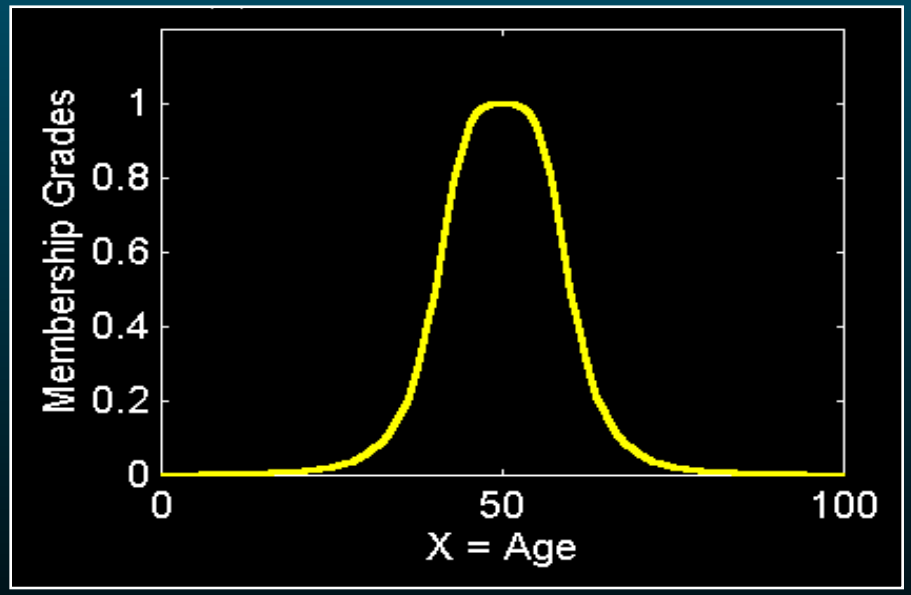
Fuzzy Sets with Cont. Universes

Fuzzy set B = "about 50 years old"

X = Set of positive real numbers (continuous)

B = {(x, μ_B(x)) | x in X}

$$\mu_B(x) = \frac{1}{1 + \left(\frac{x - 50}{10} \right)^2}$$

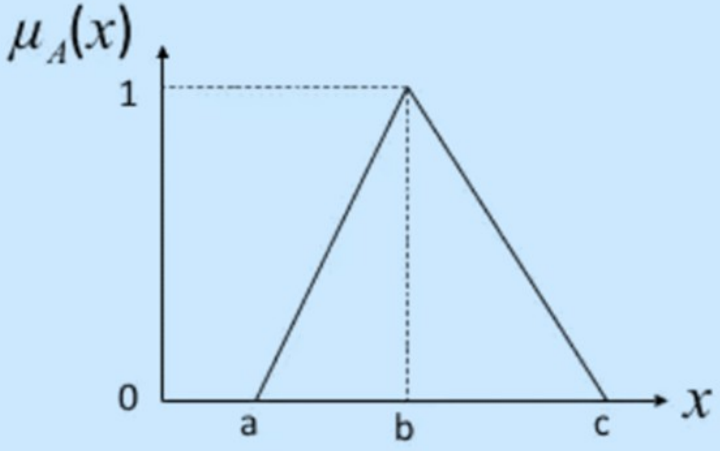


- **Triangular membership function**

  A *triangular* membership function is specified by three parameters {a, b, c}

  a, b and c represent the *x* coordinates of the three vertices of $\mu_A(x)$ in a fuzzy set A (a: lower boundary and c: upper boundary where membership degree is zero, b: the centre where membership degree is 1)

$$\mu_A(x) = \begin{cases} 0 & if\ x \le a \\ \dfrac{x-a}{b-a} & if\ a \le x \le b \\ \dfrac{c-x}{c-b} & if\ b \le x \le c \\ 0 & if\ x \ge c \end{cases}$$



- **Trapezoid membership function**
- A *trapezoidal* membership function is specified by four parameters {a, b, c, d} as follows:

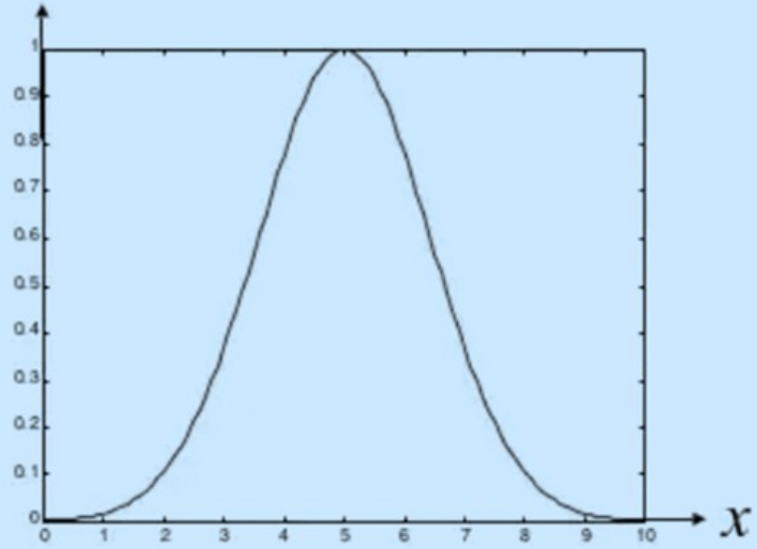
$$\mu_A(x) = \begin{cases} 0 & \text{if } x \le a \\ \dfrac{x-a}{b-a} & \text{if } a \le x \le b \\ 1 & \text{if } b \le x \le c \\ \dfrac{d-x}{d-c} & \text{if } c \le x \le d \\ 0 & \text{if } d \le x \end{cases}$$

- **Gaussian membership function**

$$\mu_A(x,c,s,m) = \exp\left[-\frac{1}{2}\left|\frac{x-c}{s}\right|^m\right]$$

  - c: centre
  - s: width
  - m: fuzzification factor (e.g., m=2)

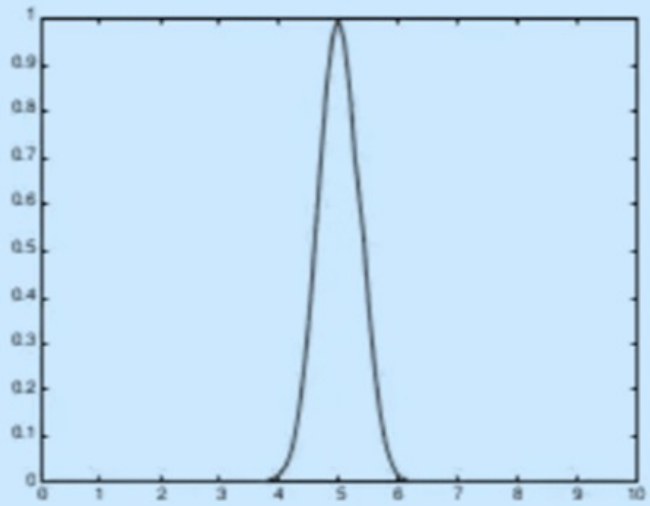
$\mu_A(x)$



c=5

s=2

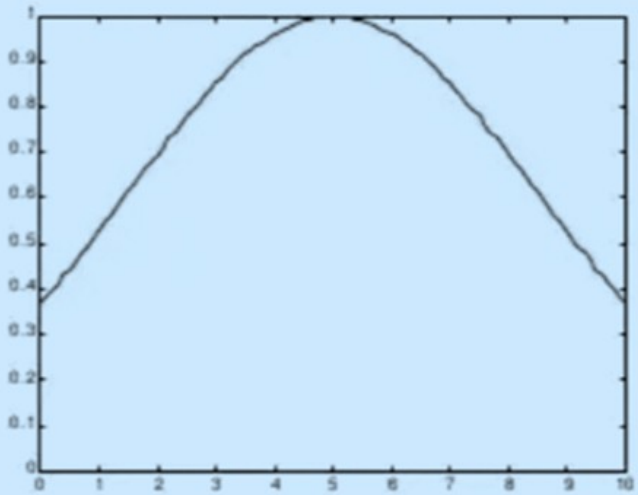
m=2



$c=5$

$s=0.5$

$m=2$



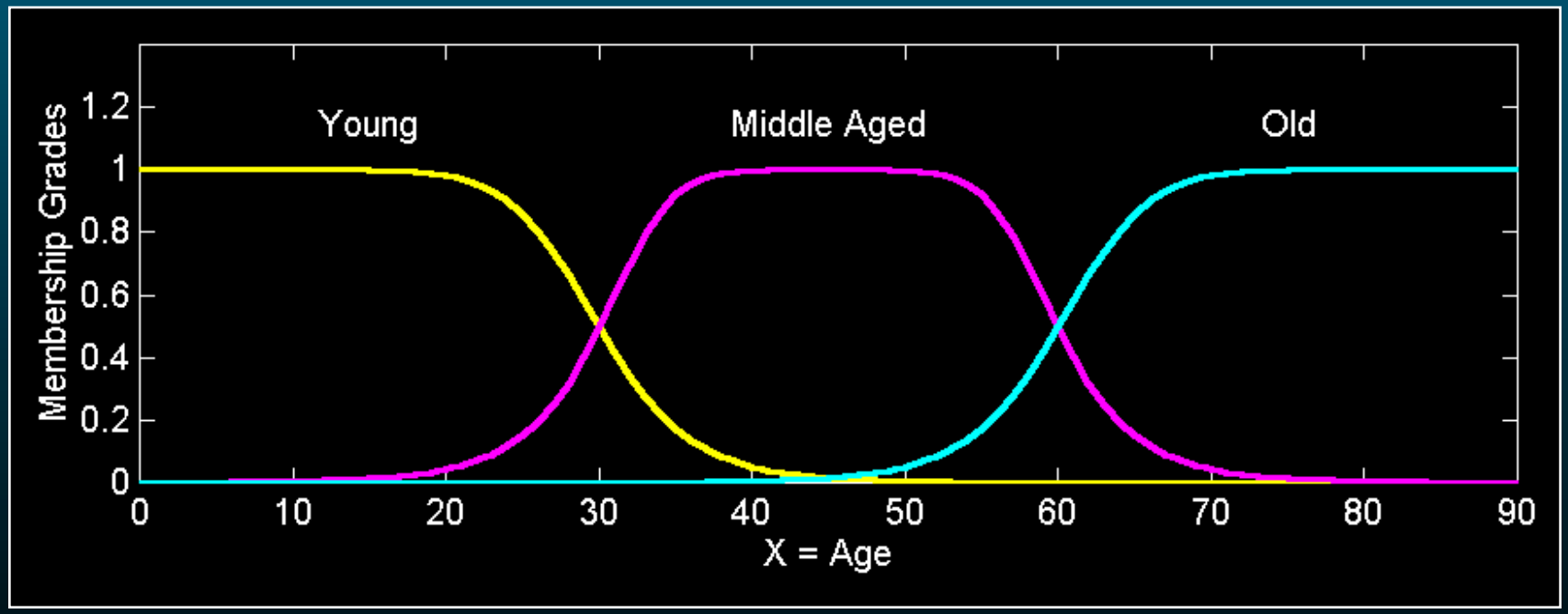
$c=5$

$s=5$

$m=2$

Fuzzy Partition

Fuzzy partitions formed by the linguistic values "young", "middle aged", and "old":



More Definitions

- Fuzzy set is uniquely specified by its MF
- To define membership functions more specifically

Support

Core

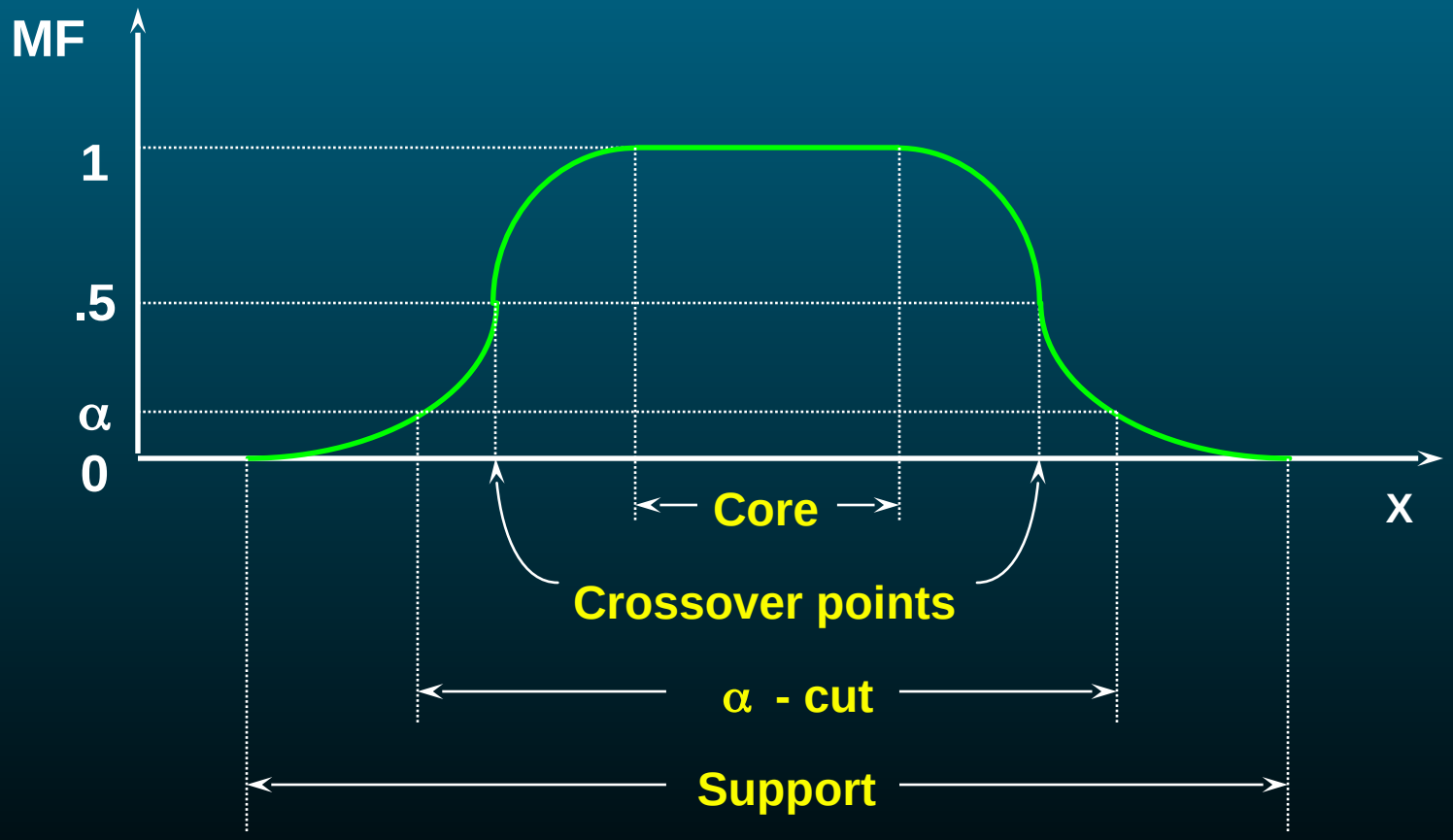
Normality

Crossover points

Fuzzy singleton

α-cut, strong α-cut

MF Terminology



MF Terminology (Cont'd)

The **support** of a fuzzy set A is the set of all points x in X such that $\mu_A(x) > 0$:

$$\text{support}(A) = \{x | \mu_A(x) > 0\}.$$

The **core** of a fuzzy set A is the set of all points x in X such that $\mu_A(x) = 1$:

$$\text{core}(A) = \{x | \mu_A(x) = 1\}.$$

A fuzzy set A is **normal** if its core is nonempty. In other words, we can always find a point $x \in X$ such that $\mu_A(x) = 1$.

A **crossover point** of a fuzzy set A is a point $x \in X$ at which $\mu_A(x) = 0.5$:

$$\text{crossover}(A) = \{x | \mu_A(x) = 0.5\}.$$

A fuzzy set whose support is a single point in X with $\mu_A(x) = 1$ is called a **fuzzy singleton**.

MF Terminology (Cont'd)

The **α-cut** or **α-level set** of a fuzzy set A is a crisp set defined by

$$A_\alpha = \{x | \mu_A(x) \geq \alpha\}.$$

Strong α-cut or **strong α-level set** are defined similarly:

$$A'_\alpha = \{x | \mu_A(x) > \alpha\}.$$

Let A be a fuzzy set.

$A = \{(x_1, 0.1), (x_2, 0.5), (x_3, 0.8), (x_4, 1.0), (x_5, 0.7), (x_6, 0.2)\}$

$support(A) = \{x_1, x_2, x_3, x_4, x_5, x_6\}$

$core = \{x_4\}$

The α-cuts of the fuzzy set A are:

$A_{0.1} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$

$A_{0.2} = \{x_2, x_3, x_4, x_5, x_6\}$

$A_{0.5} = \{x_2, x_3, x_4, x_5\}$

$A_{0.7} = \{x_3, x_4, x_5\}$

$A_{0.8} = \{x_3, x_4\}$

$A_{1.0} = \{x_4\}$

The Strong α-cuts of the fuzzy set A are:

$A_{0.1} = \{ x_2, x_3, x_4, x_5, x_6\}$

$A_{0.2} = \{x_2, x_3, x_4, x_5\}$

$A_{0.5} = \{ x_3, x_4, x_5\}$

$A_{0.7} = \{x_3, x_4\}$

$A_{0.8} = \{x_4\}$

Set-Theoretic Operations

Subset:

$$A \subseteq B \Leftrightarrow \mu_A \leq \mu_B$$

Complement:

$$\overline{A} = X - A \Leftrightarrow \mu_{\overline{A}}(x) = 1 - \mu_A(x)$$

Union:

$$C = A \cup B \Leftrightarrow \mu_c(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

Intersection:

$$C = A \cap B \Leftrightarrow \mu_c(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

Let A and B be two fuzzy sets.

A = {(1, 0.1), (2, 0.5), (3, 0.8), (4, 1)} and

B = {(1, 1), (2, 0.8), (3, 0.4), (4, 0.1)}

Perform union and intersection operations on those fuzzy sets.

Answer:

$$A \cup B = \{(1,\max(0.1, 1)), (2,\max(0.5, 0.8)), (3,\max(0.8, 0.4)), (4,\max(1, 0.1))\}$$

$$= \{(1, 1), (2, 0.8), (3, 0.8), (4, 1)\}$$

$$A \cap B = \{(1,\min(0.1, 1)), (2,\min(0.5, 0.8)), (3,\min(0.8, 0.4)), (4,\min(1, 0.1))\}$$

$$= \{(1, 0.1), (2, 0.5), (3, 0.4), (4, 0.1)\}$$

Fuzzy System

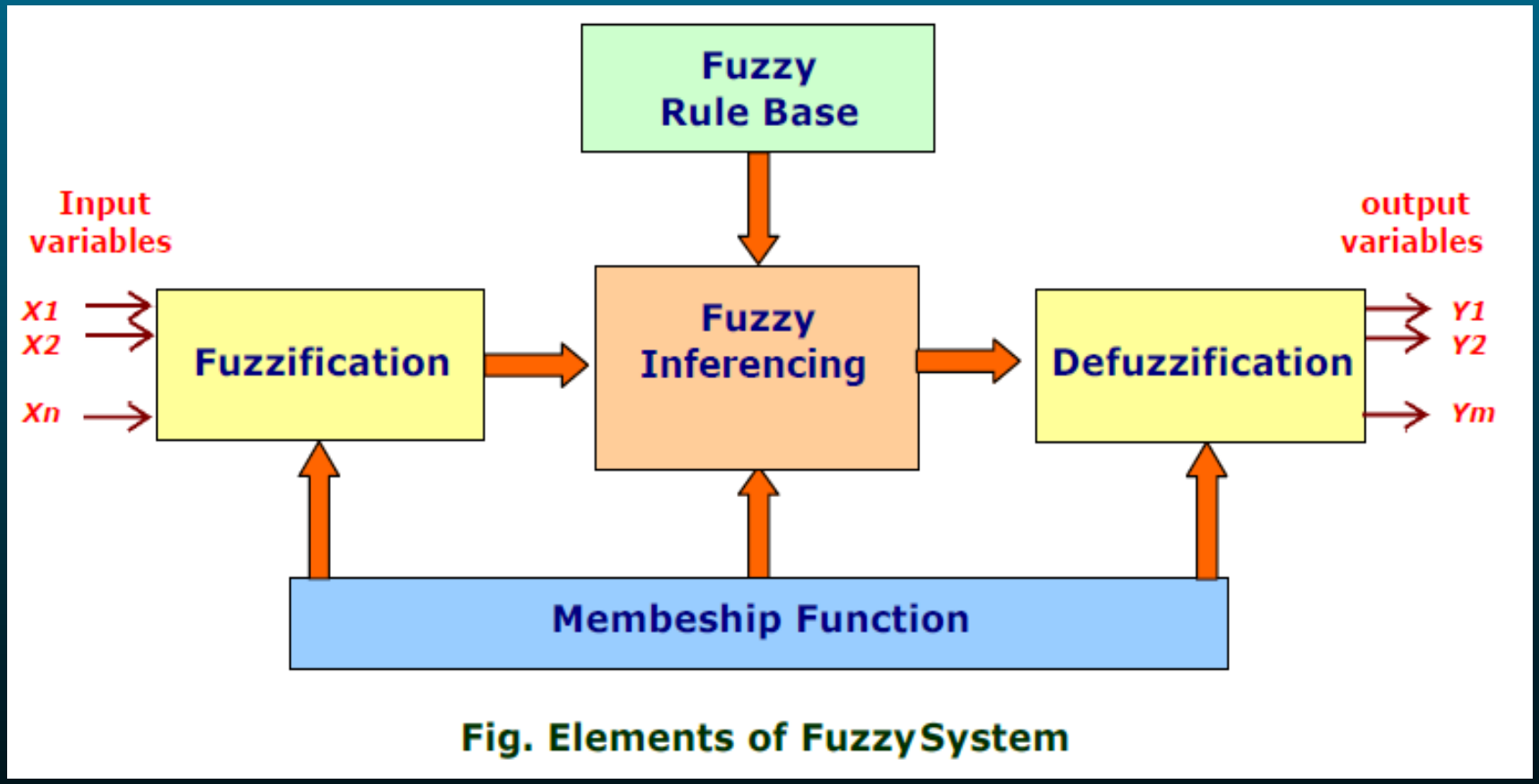


Fig. Elements of FuzzySystem

Fuzzy System elements

- **Input Vector** : $X = [x_1 , x_2, \ldots x_n]^T$ are crisp values, which are transformed into fuzzy sets in the fuzzification block.
- **Output Vector** : $Y = [y_1 , y_2, \ldots y_m]^T$ comes out from the defuzzification block, which transforms an output fuzzy set back to a crisp value.
- **Fuzzification** : a process of transforming crisp values into grades of membership for linguistic terms, "far", "near", "small" of fuzzy sets.
- **Fuzzy Rule base** : a collection of propositions containing linguistic variables; the rules are expressed in the form:

  If (x is A) AND (y is B) . . . . . THEN (z is C)

  where **x, y** and **z** represent variables (e.g. distance, size) and **A, B** and **Z** are linguistic variables (e.g. `far', `near', `small').
- **Membership function** : provides a measure of the degree of similarity of elements in the universe of discourse **U** to fuzzy set.
- **Fuzzy Inferencing** : combines the facts obtained from the Fuzzification with the rule base and conducts the Fuzzy reasoning process.
- **Defuzzyfication:** Translate results back to the real world values.

References:

J-S R Jang and C-T Sun, Neuro-Fuzzy and Soft Computing, Prentice Hall, 1997

S. Rajasekaran and G.A. Vijayalaksmi Pai ,Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications, (2005), Prentice Hall.

Artificial Intelligence

Chapter 3

INTERIM SEMESTER 2021-22
BPL
CSE3007-LT-AB306
FACULTY: SIMI V.R.

Machine Learning

What is Learning

- Learning denotes changes in a system that enable the system to do the same task more efficiently next time.
- Learning is an important feature of “Intelligence”.

Definition

A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**. (Mitchell 1997)
This means :

Given : A task **T,** A performance measure P, Some experience **E** with the task

Goal : Generalize the experience in a way that allows to improve your performance on the task.

Why do you require Machine Learning ?

- Understand and improve efficiency of human learning.
- Discover new things or structure that is unknown to humans.
- Fill in skeletal or incomplete specifications about a domain.

LEARNING

An agent is **learning** if it improves its performance on future tasks after making observations about the world.

FORMS OF LEARNING

Any component of an agent can be improved by learning from data. The improvements, and the techniques used to make them, 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.

Components to be learned

1. A direct mapping from conditions on the current state to actions.
2. A means to infer relevant properties of the world from the percept sequence.
3. Information about the way the world evolves and about the results of possible actions the agent can take.
4. Utility information indicating the desirability of world states.
5. Action-value information indicating the desirability of actions.
6. Goals that describe classes of states whose achievement maximizes the agent’s utility.

Well-Posed Learning Problems : Examples

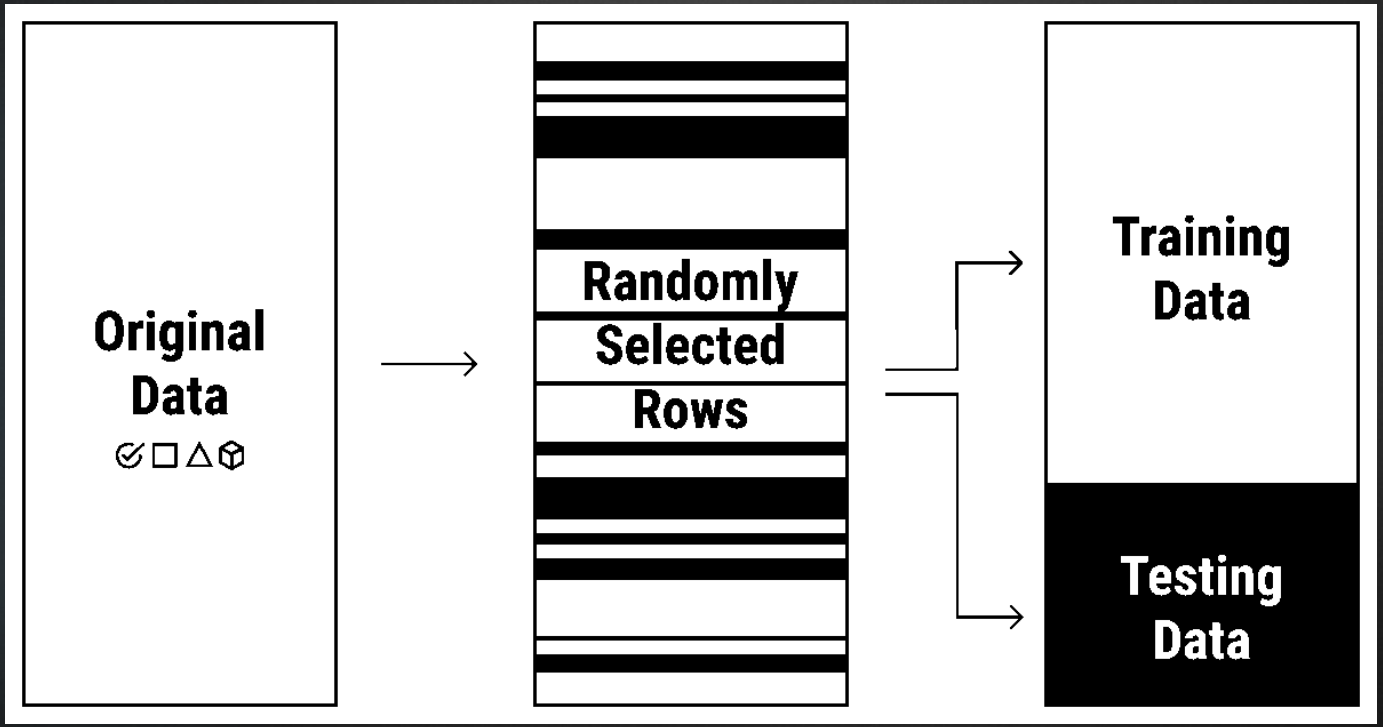
- A checkers learning problem
  - Task T : playing checkers
  - Performance measure P : percent of games won against opponents
  - Training experience E : playing practice games against itself

- A handwriting recognition learning problem
  - Task T : recognizing and classifying handwritten words within images
  - Performance measure P : percent of words correctly classified
  - Training experience E : a database of handwritten words with given classifications

- A robot driving learning problem
  - Task T : driving on public four-lane highways using vision sensors
  - Performance measure P : average distance traveled before an error (as judged by human overseer)
  - Training experience E : a sequence of images and steering commands recorded while observing a human driver

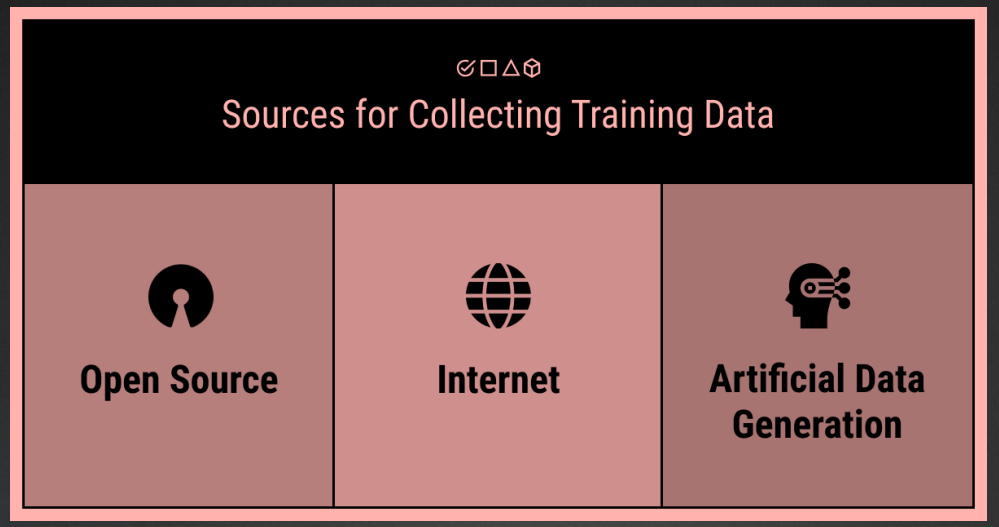
What Is Training Data in Machine Learning?

Training data is a set of samples (such as a collection of photos or videos, a set of texts or audio files, etc.) with assigned relevant and comprehensive labels (classes or tags) used to fit the parameters (weights) of a machine learning model with the goal of training it by example.



80%-20%
70%-20%

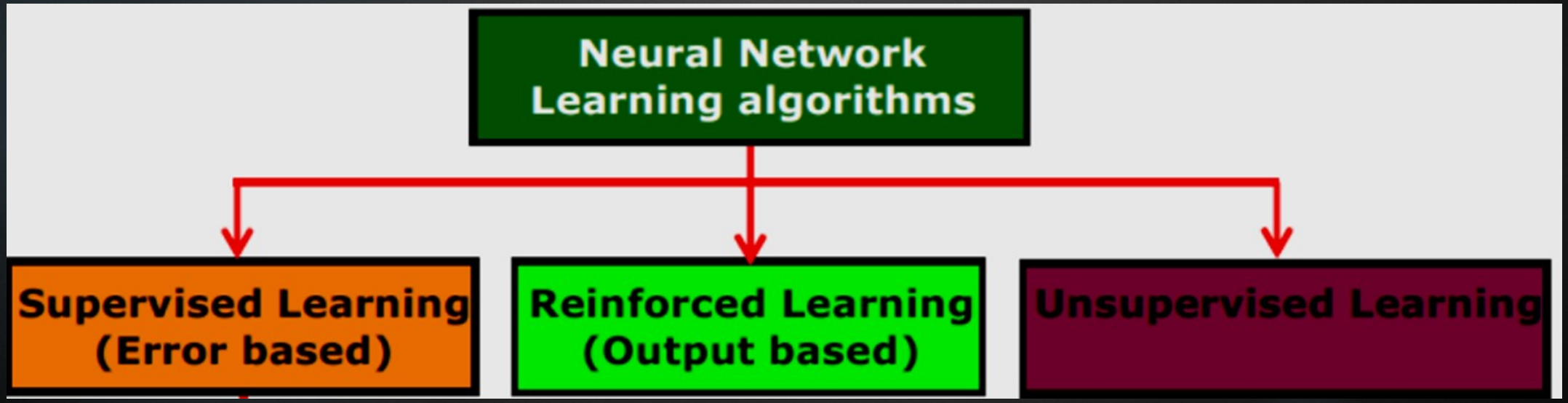
Where to get training data?



Open-source training data sets: The great benefit of this option is that it's free and it's already collected.
Web and IoT: This means that you use the Internet to collect the pieces of data. Alternatively, sensors, cameras, and other smart devices may provide you with the raw data that you will later need to annotate by hand.
Artificial training data sets: You have to create an ML model that will generate your data. This is a great way if you need large volumes of unique data to train your algorithm.

Machine Learning Areas

- **Supervised Learning:** Data and corresponding labels are given
- **Unsupervised Learning:** Only data is given, no labels provided
- **Semi-supervised Learning:** Some (if not all) labels are present
- **Reinforcement Learning**: An agent interacting with the world makes observations, takes actions, and is rewarded or punished; it should learn to choose actions in such a way as to obtain a lot of reward



Representation and prior knowledge

Supervised Learning

- A teacher is present during learning process and presents expected output.
- Every input pattern is used to train the network.
- Learning process is based on comparison, between network's computed output and the correct expected output, generating "error".
- The "error" generated is used to change network parameters that result improved performance.

Unsupervised Learning

- No teacher is present.
- The expected or desired output is not presented to the network.
- The system learns of it own by discovering and adapting to the structural features in the input patterns.

Reinforced learning

- A teacher is present but does not present the expected or desired output but only indicated if the computed output is correct or incorrect.
- The information provided helps the network in its learning process.
- A reward is given for correct answer computed and a penalty for a wrong answer.

- **In semi-supervised learning we are given a few labelled examples and must make what we can of a large collection of unlabelled examples. Even the labels themselves may not be the oracular truths that we hope for. Imagine that you are trying to build a system to guess a person's age from a photo. You gather some labeled examples by snapping pictures of people and asking their age. That's supervised learning. But in reality some of the people lied about their age.**

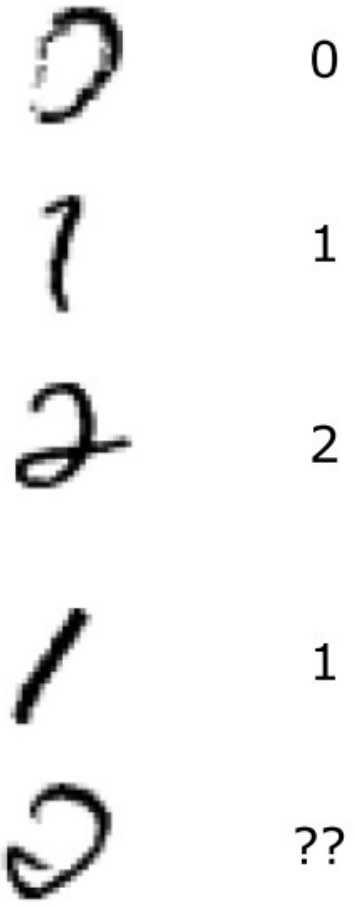
Example: Digit Recognition

Input: images / pixel grids

Output: a digit 0-9

Setup:

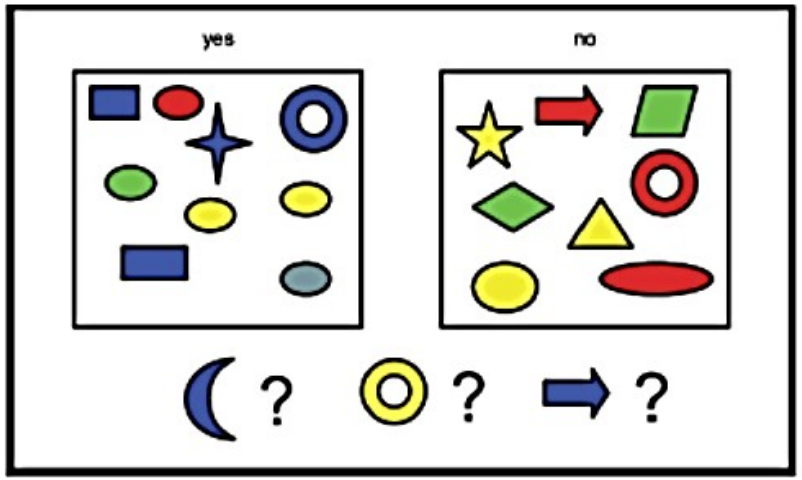
- Get a large collection of example images, each labeled with a digit
- Note: someone has to hand label all this data!
- Want to learn to predict labels of new, future digit images



Features: The attributes used to make the digit decision

- Pixels: (6,8)=ON
- Shape Patterns: NumComponents, AspectRatio, NumLoops
- …

Illustrating Classification Task



(a)

D features (attributes)

N cases

Color	Shape	Size (cm)	Label
Blue	Square	10	1
Red	Ellipse	2.4	1
Red	Ellipse	20.7	0

(b)

(a) Some labeled training examples of colored shapes, along with 3 unlabeled test cases. (b): Representing the training data as an N × D design matrix. Row i represents the feature vector x_i. The last column is the label, $y_i \in \{0, 1\}$.

References

http://www.myreaders.info/html/soft_computing.html

J-S R Jang and C-T Sun, Neuro-Fuzzy and Soft Computing, Prentice Hall, 1997

S. Rajasekaran and G.A. Vijayalaksmi Pai ,Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications, (2005), Prentice Hall.

Artificial Intelligence

Chapter 4

INTERIM SEMESTER 2021-22
BPL
CSE3007-LT-AB306
FACULTY: SIMI V.R.

Machine learning algorithms

Regression

- Technique used for the modeling and analysis of numerical data
- Exploits the relationship between two or more variables so that we can gain information about one of them through knowing values of the other
- Regression can be used for prediction, estimation, hypothesis testing, and modeling causal relationships.
- Linear functions provide the basis for many learning algorithms.
- Can solve the problem of predicting a real-valued function from training examples.
- Regression is a set of techniques for estimating relationships

Simple Linear Regression

We're going to fit a line $y = b_0 + b_1x$ to our data. Here, x is called the independent variable or predictor variable, and y is called the dependent variable or response variable.

Before we talk about how to do the fit, let's take a closer look at the important quantities from the fit:

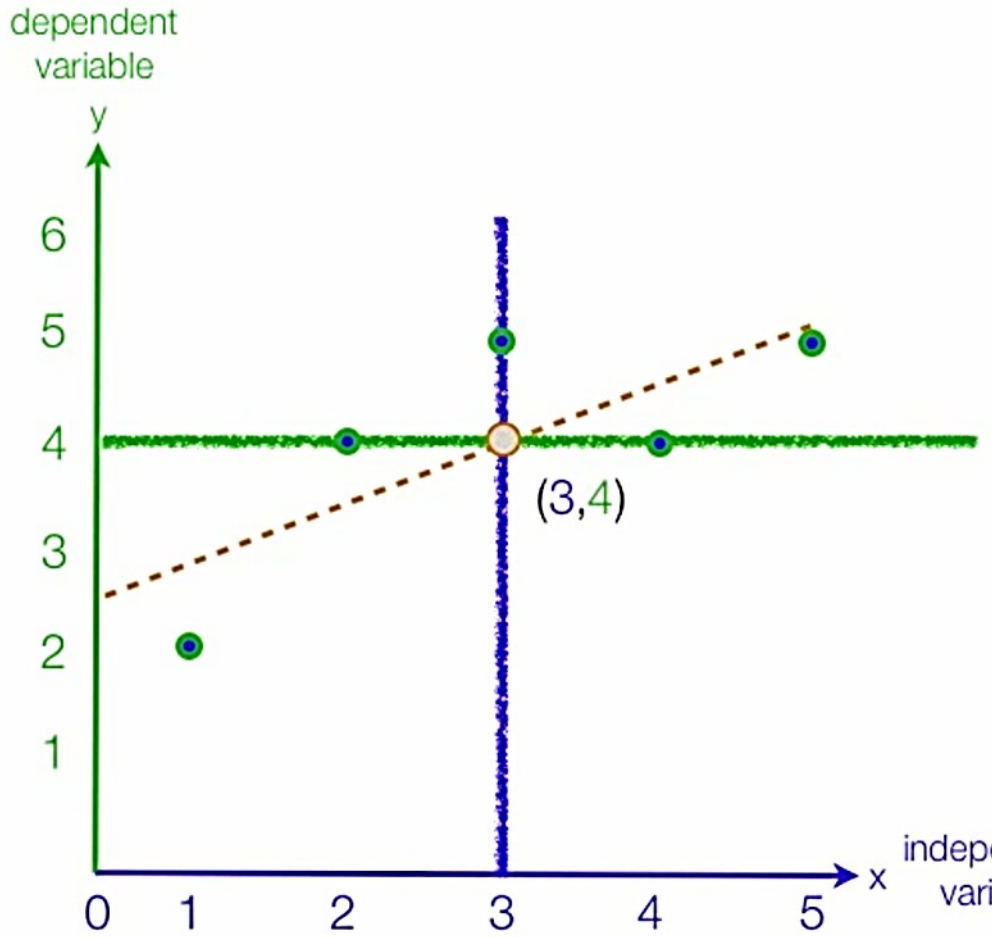
- b_1 is the slope of the line: this is one of the most important quantities in any linear regression analysis.

  A value very close to 0 indicates little to no relationship; large positive or negative values indicate large positive or negative relationships, respectively.

- b_0 is the intercept of the line.

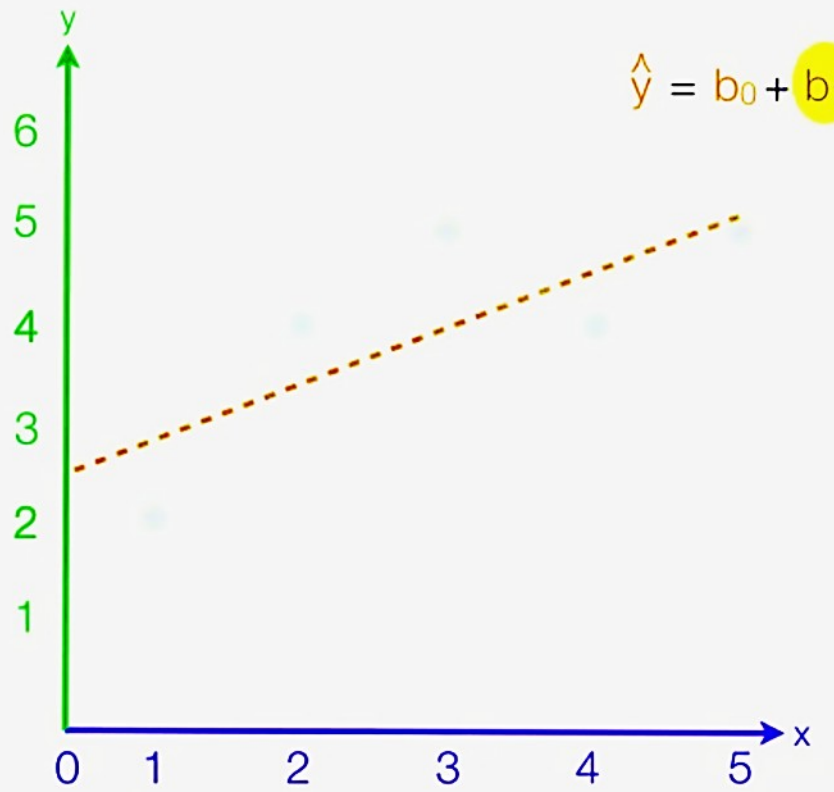
In order to actually fit a line, we'll start with a way to quantify how good a line is. We'll then use this to fit the "best" line we can.

Linear Regression



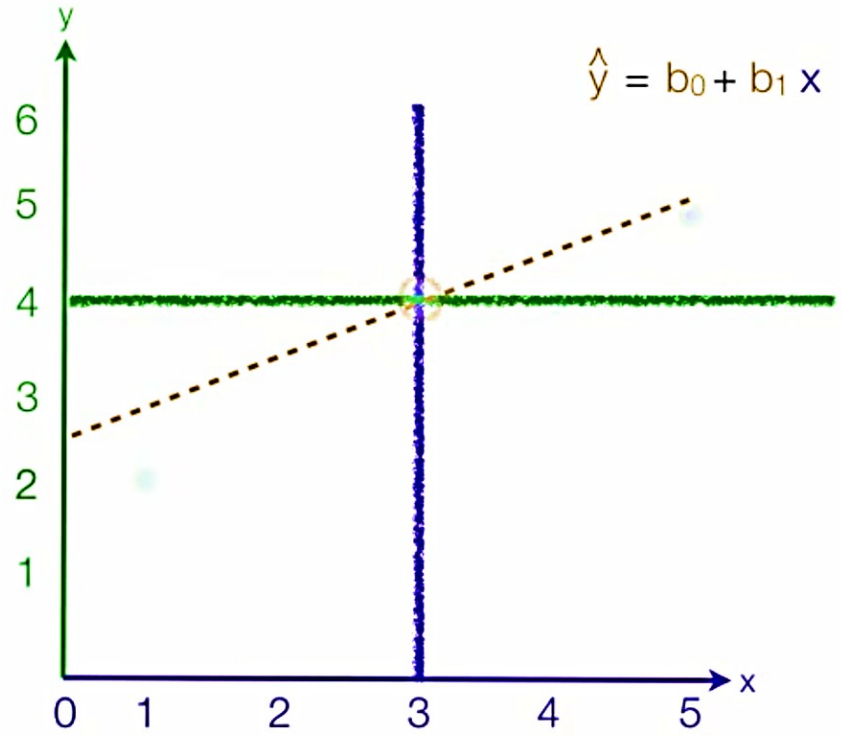
independent variable	dependent variable
x	y
1	2
2	4
3	5
4	4
5	5

mean



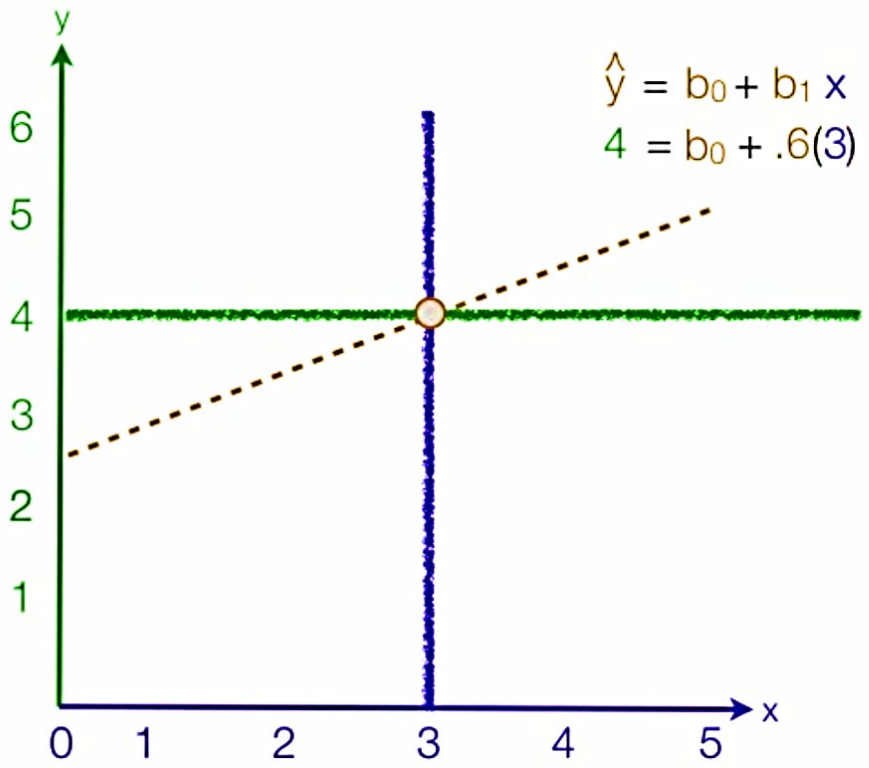
$$\hat{y} = b_0 + b_1 x$$

x	y	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})^2$	$(x - \bar{x})(y - \bar{y})$
1	2	-2	-2	4	4
2	4	-1	0	1	0
3	5	0	1	0	0
4	4	1	0	1	0
5	5	2	1	4	2



x	y	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})^2$	$(x - \bar{x})(y - \bar{y})$
1	2	-2	-2	4	4
2	4	-1	0	1	0
3	5	0	1	0	0
4	4	1	0	1	0
5	5	2	1	4	2
				10	6
mean	3	4			

$$b_1 = \frac{6}{10} = .6 = \frac{\Sigma (x - \bar{x})(y - \bar{y})}{\Sigma (x - \bar{x})^2}$$



$\hat{y} = b_0 + b_1 x$

$4 = b_0 + .6(3)$

$b_0 = 2.2$

$b_1 = .6$

$\hat{y} = 2.2 + .6x$

x	y	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})^2$	$(x - \bar{x})(y - \bar{y})$
1	2	-2	-2	4	4
2	4	-1	0	1	0
3	5	0	1	0	0
4	4	1	0	1	0
5	5	2	1	4	2
				10	6
mean	3	4			

$4 = b_0 + .6(3)$

$4 = b_0 + 1.8$

$-1.8 \qquad -1.8$

$2.2 = b_0$

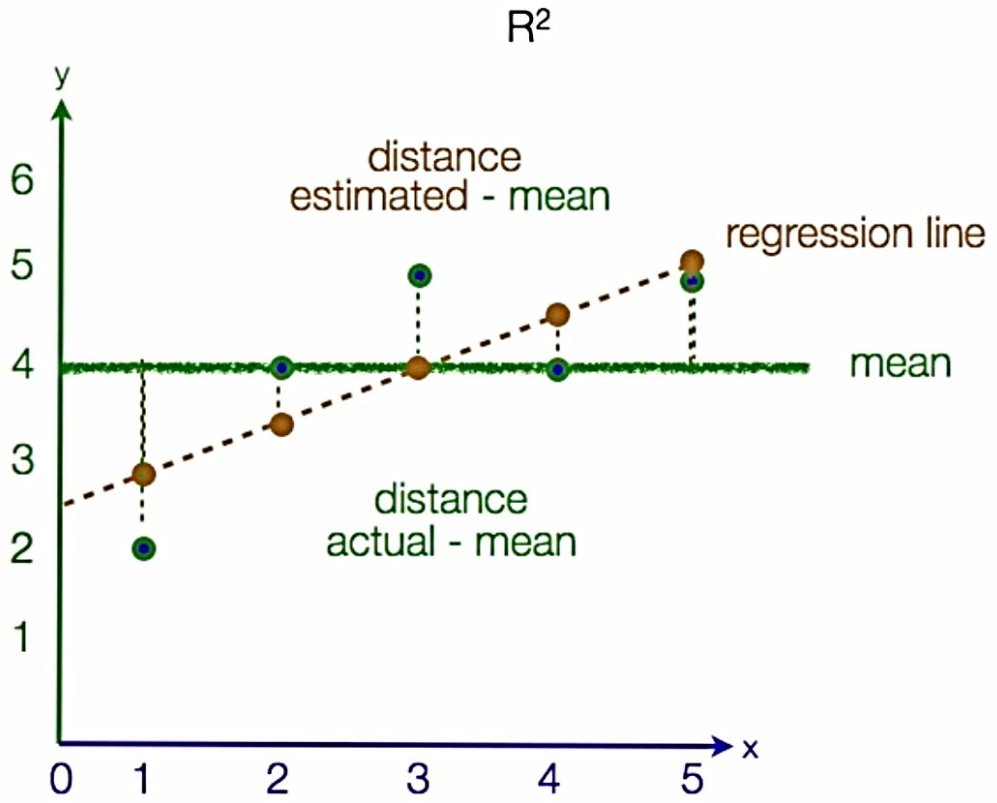
$$b_1 = \frac{6}{10} = .6 = \frac{\Sigma(x - \bar{x})(y - \bar{y})}{\Sigma(x - \bar{x})^2}$$

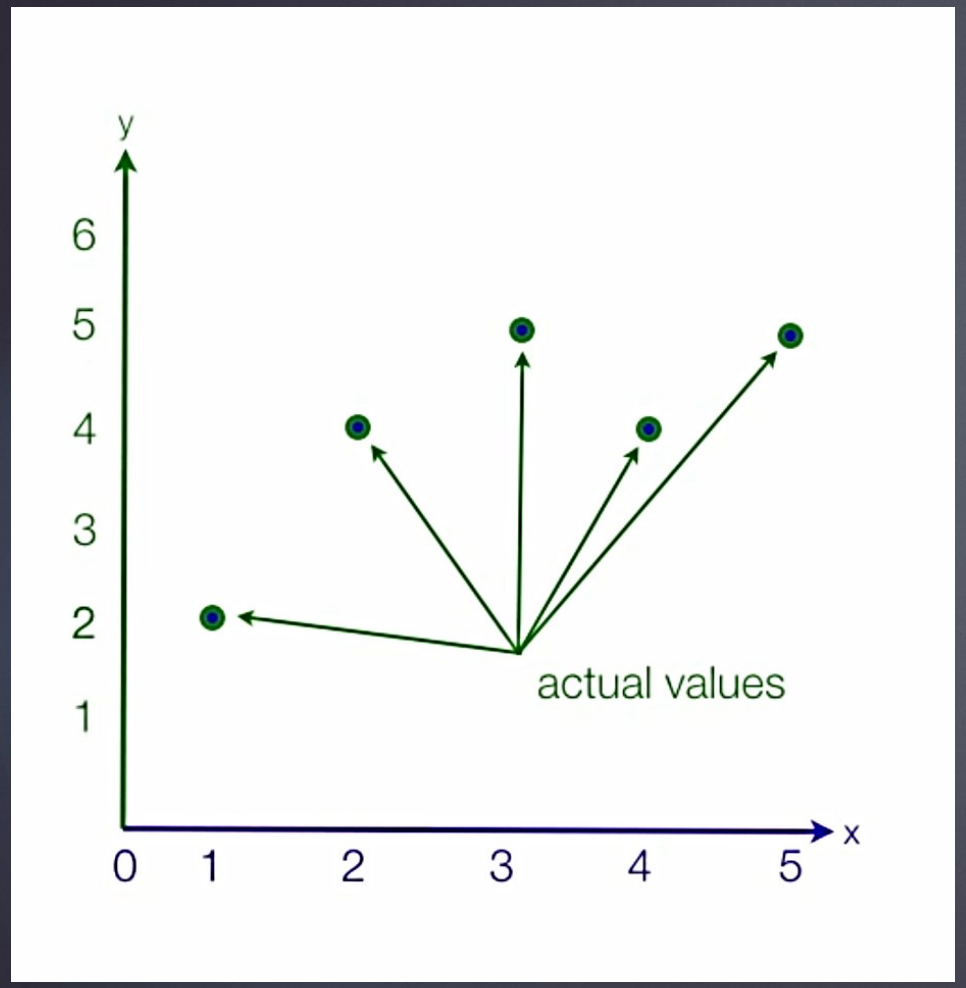
Goodness of fit – R^2

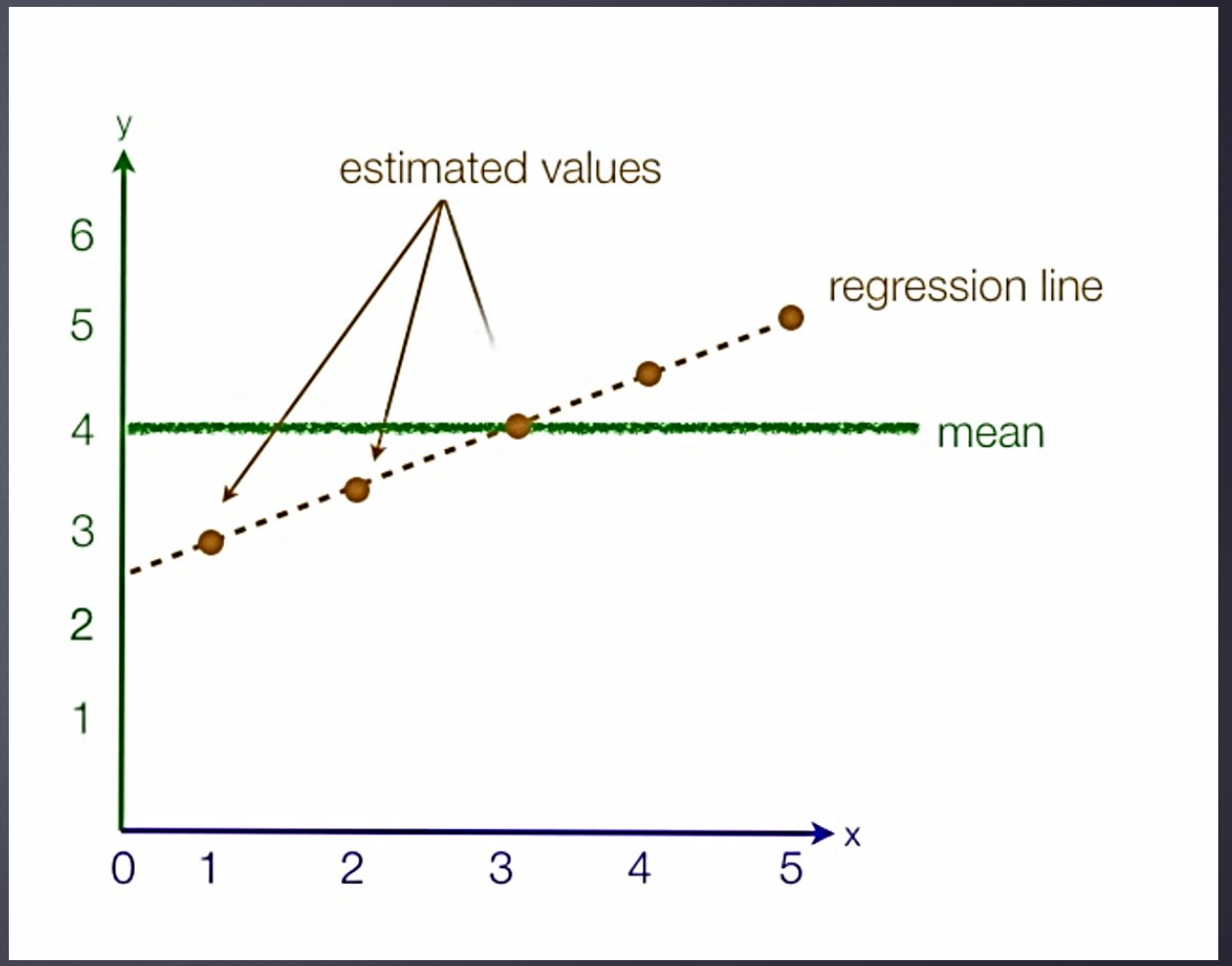
WHAT IS R-SQUARED?

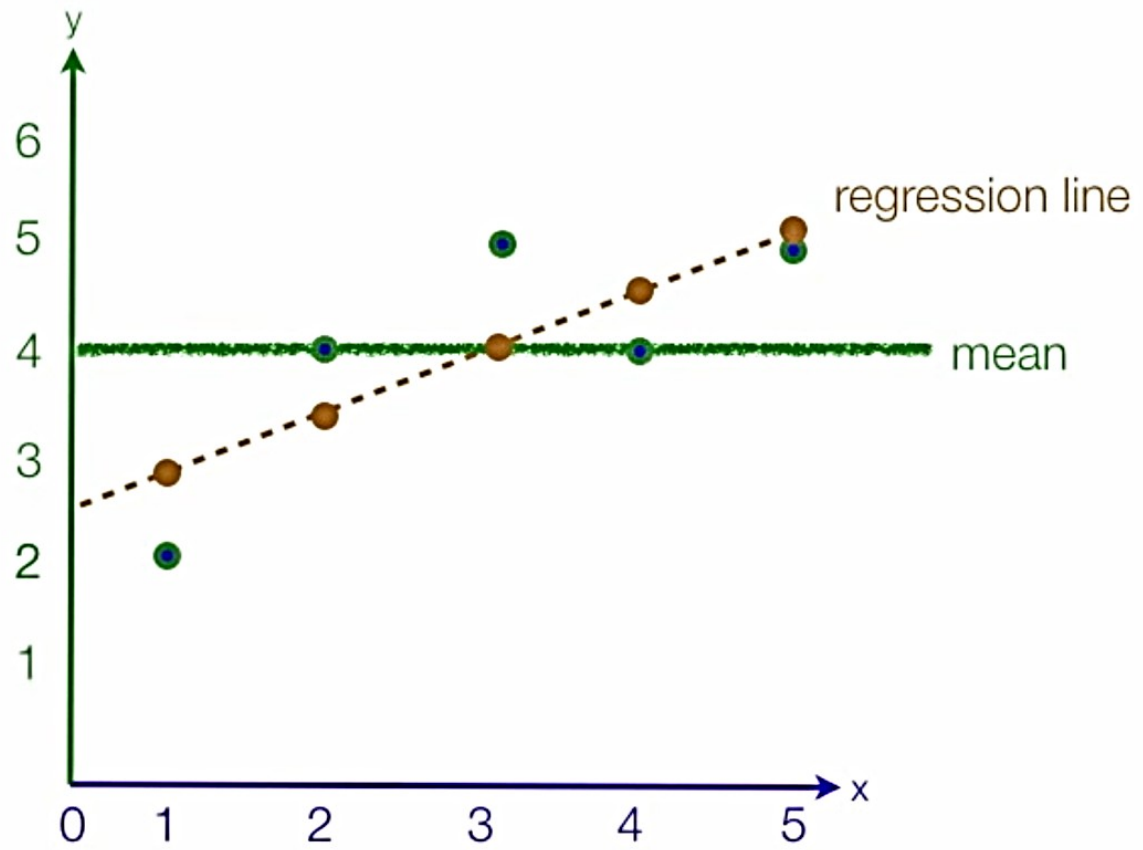
- R-squared is a statistical measure of how close the data are to the fitted regression line.
- It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.
- The definition of R-squared is fairly straight-forward; it is the percentage of the response variable variation that is explained by a linear model.
- R-squared = Explained variation / Total variation

Goodness of fit – R^2



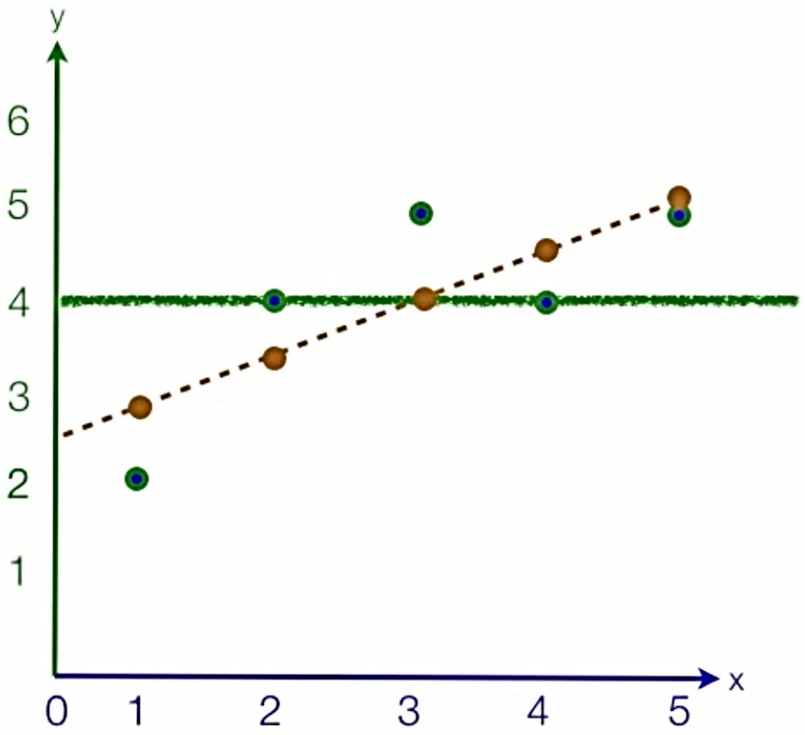

y
6
5
4
3
2
1
actual values
0
1
2
3
4
5
x


y
estimated values
regression line
mean
6
5
4
3
2
1
0
1
2
3
4
5
x



distance
actual - mean

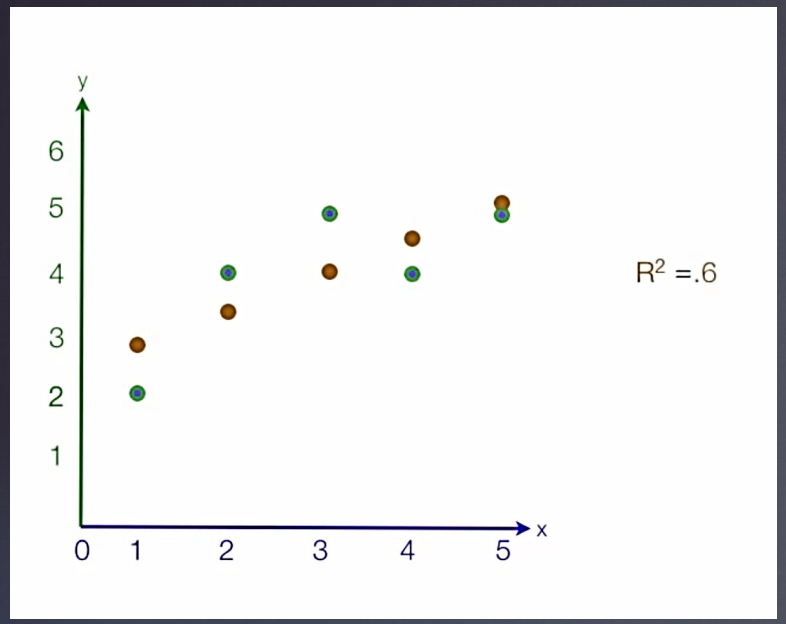
distance
estimated - mean

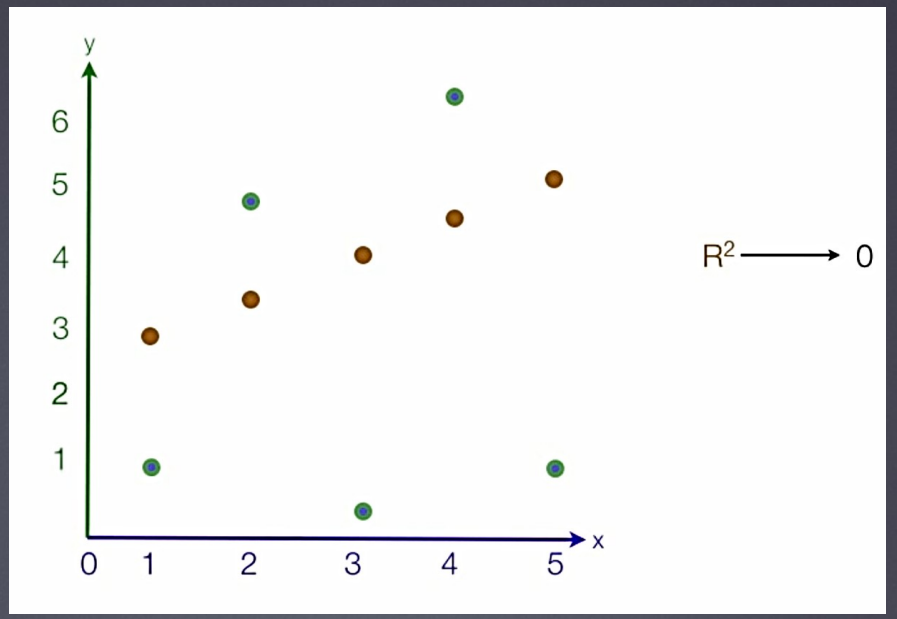


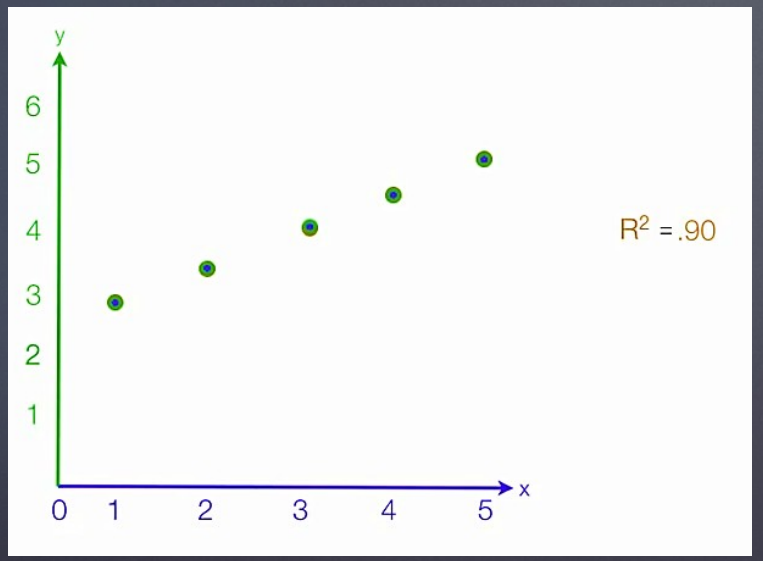
x	y	$y - \bar{y}$	$(y - \bar{y})^2$	$\hat{y}$	$\hat{y} - \bar{y}$	$(\hat{y} - \bar{y})^2$
1	2	-2	4	2.8	-1.2	1.44
2	4	0	0	3.4	-.6	.36
3	5	1	1	4	0	0
4	4	0	0	4.6	.6	.36
5	5	1	1	5.2	1.2	1.44
			6			3.6
mean	4					

$$R^2 = \frac{3.6}{6} = .6 = \frac{\text{sum}\ \Sigma(\hat{y} - \bar{y})^2}{(y - \bar{y})^2}$$

Interpretation of values of R^2







$R^2=1$

Regression line is a Perfect fit on actual values

$R^2=0$

There is larger distance between Actual and predicted values.

Mean Squared Error

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \hat{y} \right)}^{2}$$

The square of the difference between actual and predicted

Mean Squared Error Definition

- The mean squared error (MSE) tells you how close a regression line is to a set of points.
- It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them.
- The squaring is necessary to remove any negative signs.
- It also gives more weight to larger differences.
- It’s called the mean squared error as you’re finding the average of a set of errors.
- The lower the MSE, the better the forecast.

Mean Squared Error

General steps to calculate the MSE from a set of X and Y values:

1. **Find the regression line.**
2. **Insert your X values into the linear regression equation to find the new Y values (Y').**
3. **Subtract the new Y value from the original to get the error.**
4. **Square the errors.**
5. **Add up the errors (the Σ in the formula is summation notation).**
6. **Find the mean.**

Mean Squared Error

Example Problem:

Find the MSE for the following set of values: (43,41), (44,45), (45,49), (46,47), (47,44).

Step 1: Find the regression line.

Regression line y = 9.2 + 0.8x.

Step 2: Find the new Y' values:

9.2 + 0.8(43) = 43.6

9.2 + 0.8(44) = 44.4

9.2 + 0.8(45) = 45.2

9.2 + 0.8(46) = 46

9.2 + 0.8(47) = 46.8

x	y		y-	$(y-)^2$
43	41	43.6	41 – 43.6 = -2.6	6.76
44	45	44.4	45 – 44.4 = 0.6	0.36
45	49	45.2	49 – 45.2 = 3.8	14.44
46	47	46	47 – 46 = 1	1
47	44	46.8	44 – 46.8 = -2.8	7.84

Step 3: Find the error (Y – Y’):

41 – 43.6 = -2.6

45 – 44.4 = 0.6

49 – 45.2 = 3.8

47 – 46 = 1

44 – 46.8 = -2.8

Step 4: Square the Errors:

$-2.6^2 = 6.76$

$0.6^2 = 0.36$

$3.8^2 = 14.44$

$1^2 = 1$

$-2.8^2 = 7.84$

Step 5: Add all of the squared errors up: 6.76 + 0.36 + 14.44 + 1 + 7.84 = 30.4

Step 6: Find the mean squared error: 30.4 / 5 = 6.08.

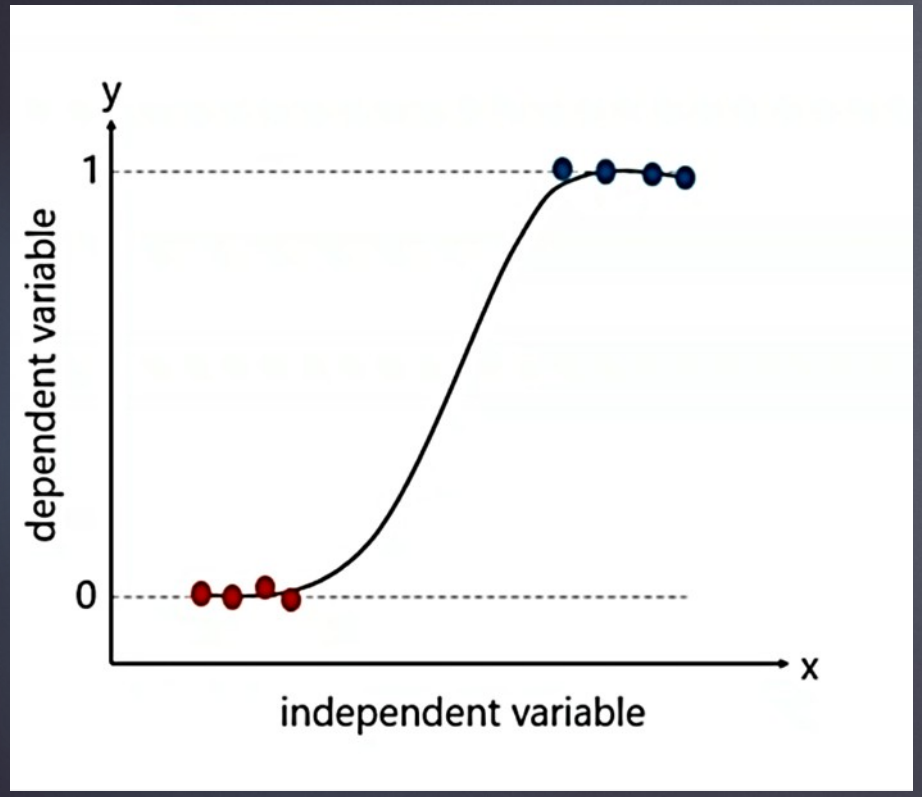
What does the Mean Squared Error Tell You?

- The smaller the mean squared error, the closer you are to finding the line of best fit.
- Depending on your data, it may be impossible to get a very small value for the mean squared error.
- For example, the above data is scattered wildly around the regression line, so 6.08 is as good as it gets (and is in fact, the line of best fit).

The smallest MSE would be the line of best fit.

Logistic regression

Logistic Regression is a method used to predict a dependent variable, given a set of independent variables, such that the dependent variable is categorical.



- *Dependent variable (Y):*

The response binary variable holding values like 0 or 1, Yes or No, A, B or C

- *Independent variable (X):*

The predictor variable used to predict the response variable.

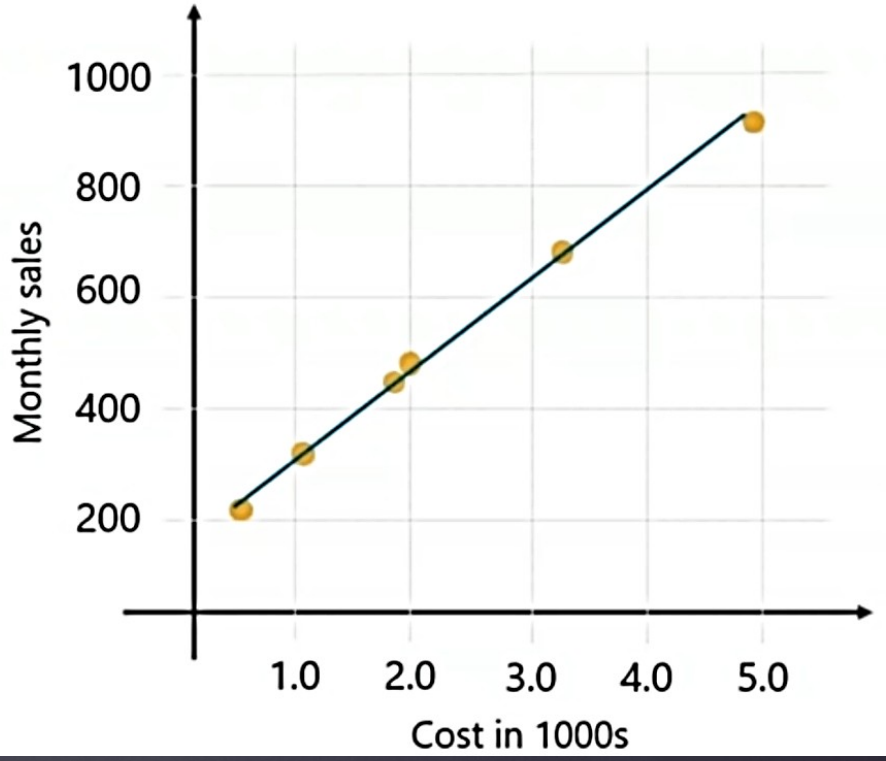
The following equation is used to represent a linear regression model:

$$\text{Log}\left(\frac{P}{1-P}\right)=a_0+a_1x_1+a_2x_2+\ldots+a_nx_n$$

Linear Regression Use Case

To forecast monthly sales by studying the relationship between the monthly e-commerce sales and the online advertising costs.

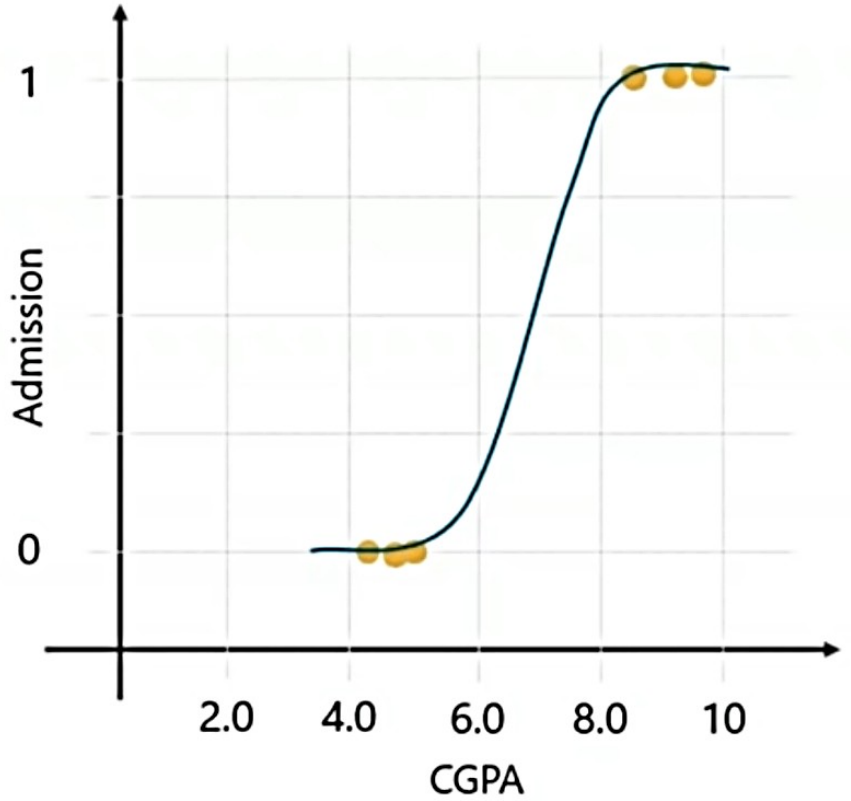
Monthly sales	Advertising cost In 1000s
200	0.5
900	5
450	1.9
680	3.2
490	2.0
300	1.0



Logistic Regression Use Case

To predict if a student will get admitted to a school based on his CGPA.

Admission	CGPA
0	4.2
0	5.1
0	5.5
1	8.2
1	9.0
1	9.1

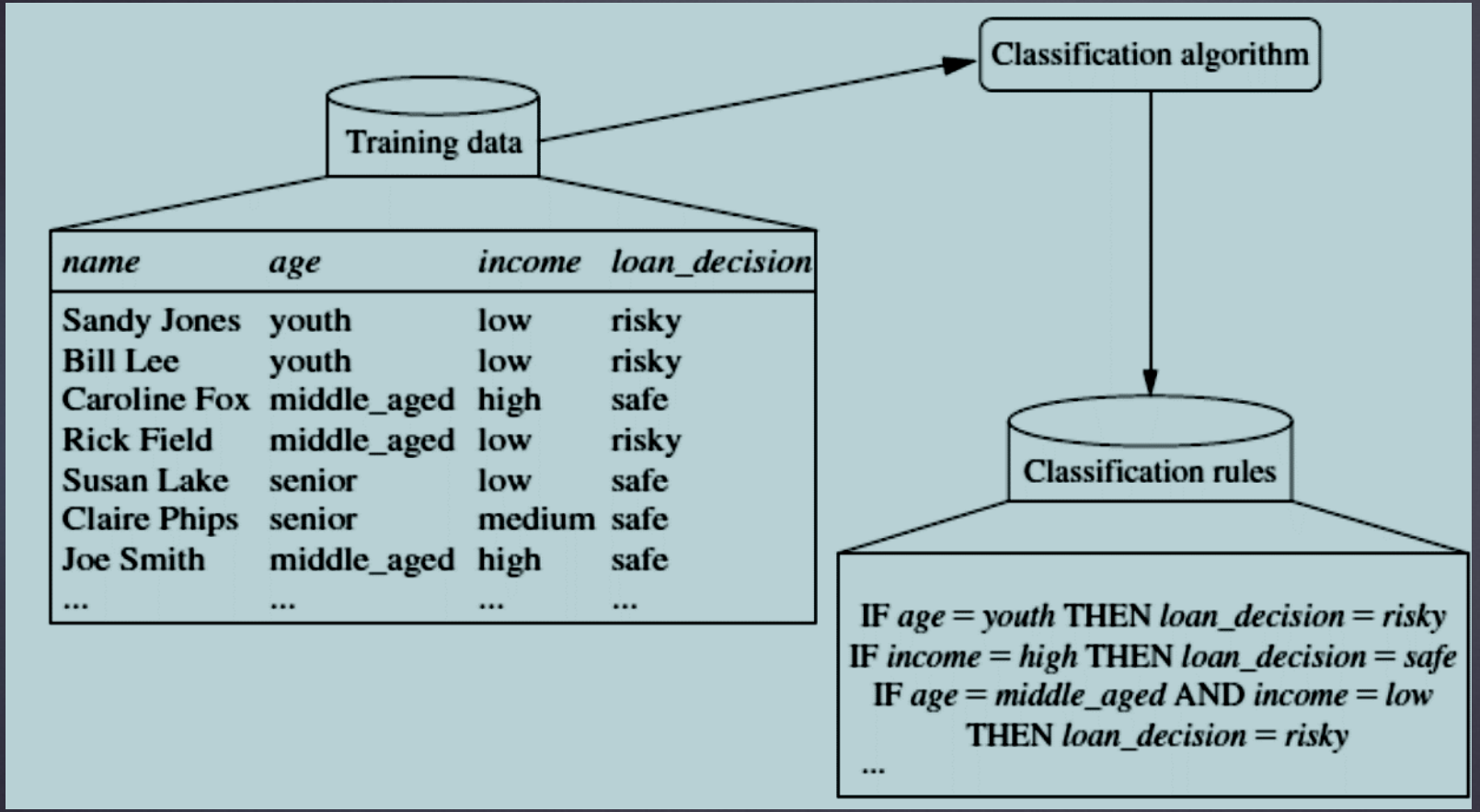


	Linear Regression	Logistic Regression
1. Definition	To predict a continuous dependent variable based on values of independent variable	To predict a categorical dependent variable based on values of independent variables
2. Variable Type	Continuous dependent variable	Categorical dependent variable
3. Estimation Method	Least square estimation	Maximum likelihood estimation
4. Equation	$Y= a_0+a_1x$	$Log()=a_0+a_1x_1+a_2x_2+\ldots+a_nx_n$
5. Best fit line	Straight line	Curve
6. Relationship between dependent and independent variable	Linear	Non linear
7. Output	Predicted Integer value	Predicted binary value (0/1)

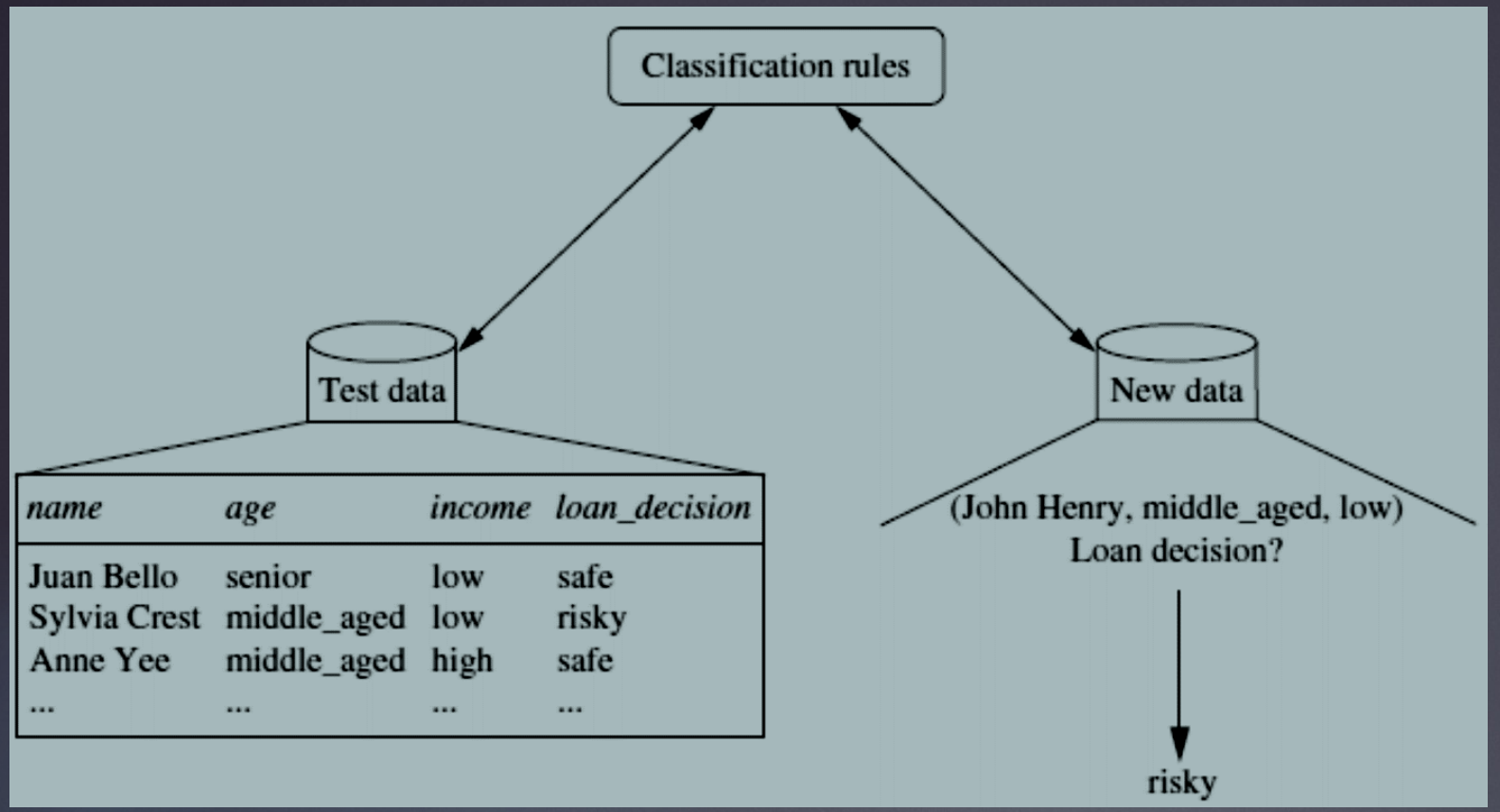
Classification

Data classification is a two-step process

1. **Consisting of a *learning step* (where a classification model is constructed)**
2. **A *classification step* (where the model is used to predict class labels for given data).**



Learning: Training data are analyzed by a classification algorithm. Here, the class label attribute is *loan decision*, and the learned model or classifier is represented in the form of classification rules.



Classification:

Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

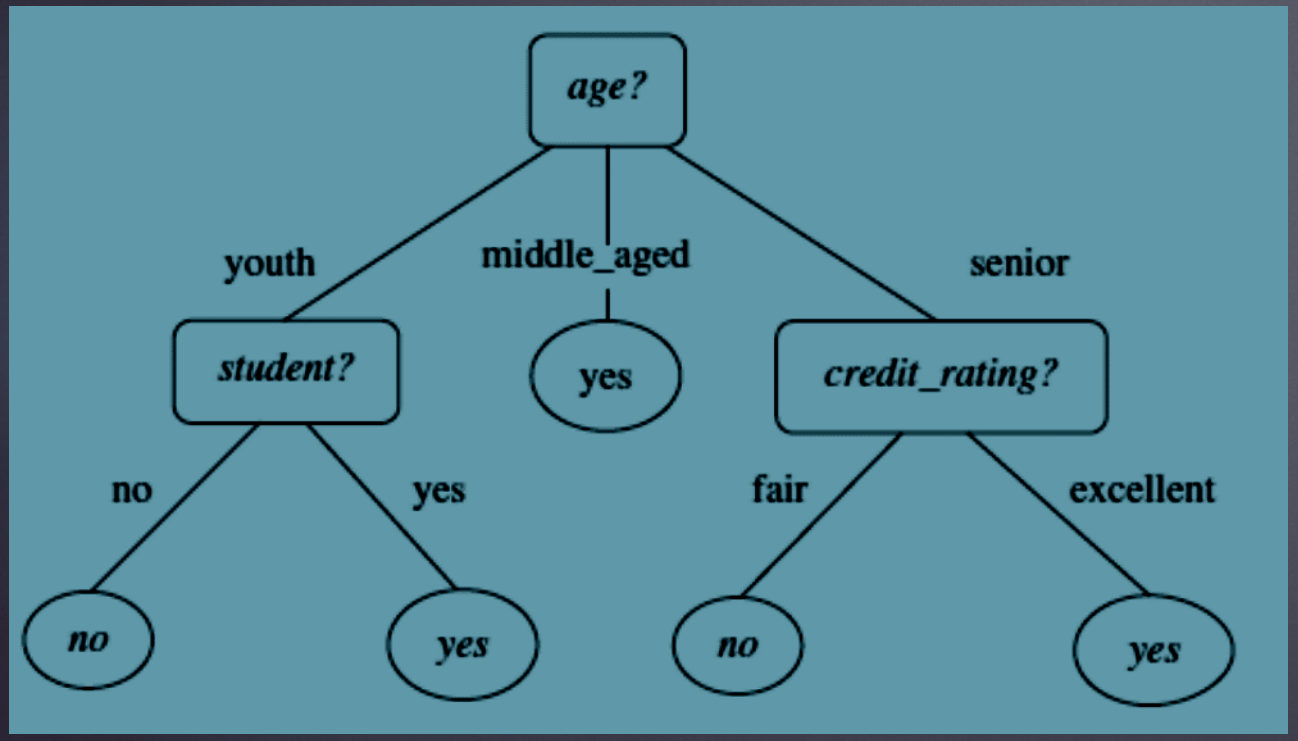
Decision tree induction : learning of decision trees from class-labeled training tuples.

- The **accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier.
- The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple.

Decision Tree

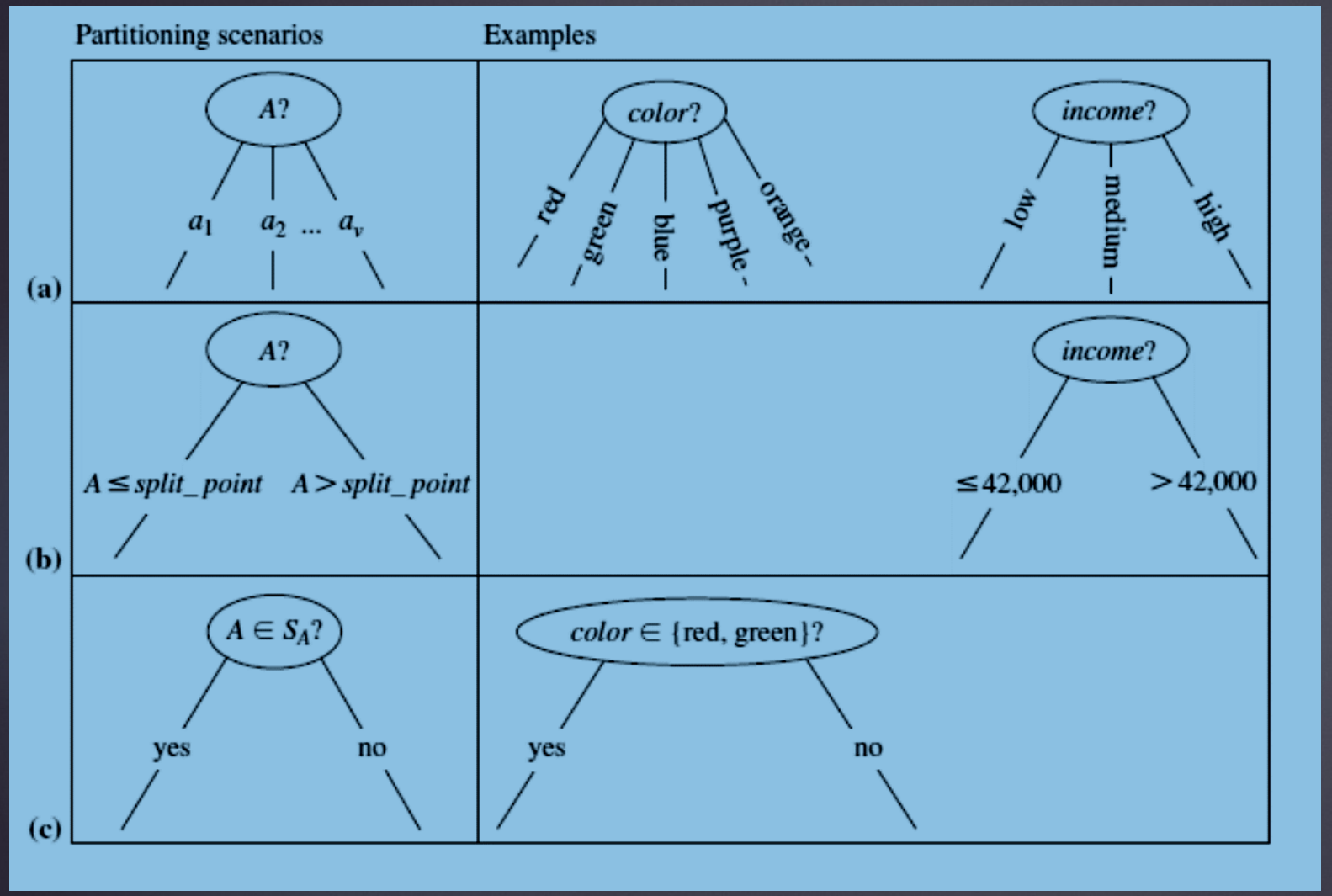
Decision tree : flowchart-like tree structure, where each **internal node** (nonleaf node) denotes a test on an attribute, each **branch** represents an outcome of the test, and each **leaf node** (or *terminal node*) holds a class label.

The topmost node in a tree is the **root** node.



- **A decision tree for the concept *buys computer* that is, it predicts whether a customer at *AllElectronics* is likely to purchase a computer.**
- **Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.**
- **Some decision tree algorithms produce only *binary* trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.**

Partitioning based on the splitting criterion



This figure shows three possibilities for partitioning tuples based on the splitting criterion, each with examples. Let *A* be the splitting attribute.

(a) If *A* is discrete-valued, then one branch is grown for each known value of *A*.

(b) If *A* is continuous-valued, then two branches are grown, corresponding to

$$A \leq split\_point \text{ and } A > split\_point.$$

(c) If *A* is discrete-valued and a binary tree must be produced, then the test is of the form $A \in S_A$, where S_A is the splitting subset for *A*.

Attribute Selection Measures

- An attribute selection measure is a heuristic for selecting the splitting criterion that “best” separates a given data partition, *D*, of class-labeled training tuples into individual classes.
- If we were to split *D* into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class).
- Conceptually, the “best” splitting criterion is the one that most closely results in such a scenario.
- Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.
- **Three popular attribute selection measures**

  Information gain, gain ratio, and Gini index.

Information Gain/ Entropy

- This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or “information content” of messages.
- Let node *N* represent or hold the tuples of partition *D*.
- The attribute with the highest information gain is chosen as the splitting attribute for node *N*.
- This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or “impurity” in these partitions.
- Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.

The expected information needed to classify a tuple in *D* is given by

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

Where *pi* is the nonzero probability that an arbitrary tuple in *D* belongs to class *Ci*.
Note that, at this point, the information we have is based solely on the proportions of tuples of each class.

- ***Info*(*D*) is just the average amount of information needed to identify the class label of a tuple in *D***
- ***Info*(*D*) is also known as the entropy of *D*.**

Now, suppose we were to partition the tuples in D on some attribute A having v distinct values, $\{a_1, a_2, \ldots, a_v\}$, as observed from the training data. If A is discrete-valued, these values correspond directly to the v outcomes of a test on A. Attribute A can be used to split D into v partitions or subsets, $\{D_1, D_2, \ldots, D_v\}$, where D_j contains those tuples in D that have outcome a_j of A. These partitions would correspond to the branches grown from node N. Ideally, we would like this partitioning to produce an exact classification of the tuples. That is, we would like for each partition to be pure. However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class).

How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

The term $\frac{|D_j|}{|D|}$ acts as the weight of the jth partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions.

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is,

$$Gain(A) = Info(D) - Info_A(D).$$

In other words, $Gain(A)$ tells us how much would be gained by branching on A. It is the expected reduction in the information requirement caused by knowing the value of A. The attribute A with the highest information gain, $Gain(A)$, is chosen as the splitting attribute at node N. This is equivalent to saying that we want to partition on the attribute A that would do the "best classification," so that the amount of information still required to finish classifying the tuples is minimal (i.e., minimum $Info_A(D)$).

Induction of a decision tree using information gain

Example

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

- The presents a training set, *D*, of class-labeled tuples randomly selected from the *AllElectronics* customer database.
- In this example, each attribute is discrete valued.
- The class label attribute, *buys computer*, has two distinct values (namely, {*yes, no*}); therefore, there are two distinct classes (i.e., $m = 2$).
- Let class C_1 correspond to *yes* and class C_2 correspond to *no*.

There are nine tuples of class *yes* and five tuples of class *no*. A (root) node *N* is created for the tuples in *D*. To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We first use Equation given below.

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

To compute the expected information needed to classify a tuple in *D*:

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

Next, we need to compute the expected information requirement for each attribute. Let's start with the attribute *age*.

We need to look at the distribution of *yes* and *no* tuples for each category of *age*.

For the *age* category "youth," there are two *yes* tuples and three *no* tuples.

For the category "middle aged," there are four *yes* tuples and zero *no* tuples.

For the category "senior," there are three *yes* tuples and two *no* tuples.

The expected information needed to classify a tuple in *D* if the tuples are partitioned according to *age* is

(Hint: use equation)

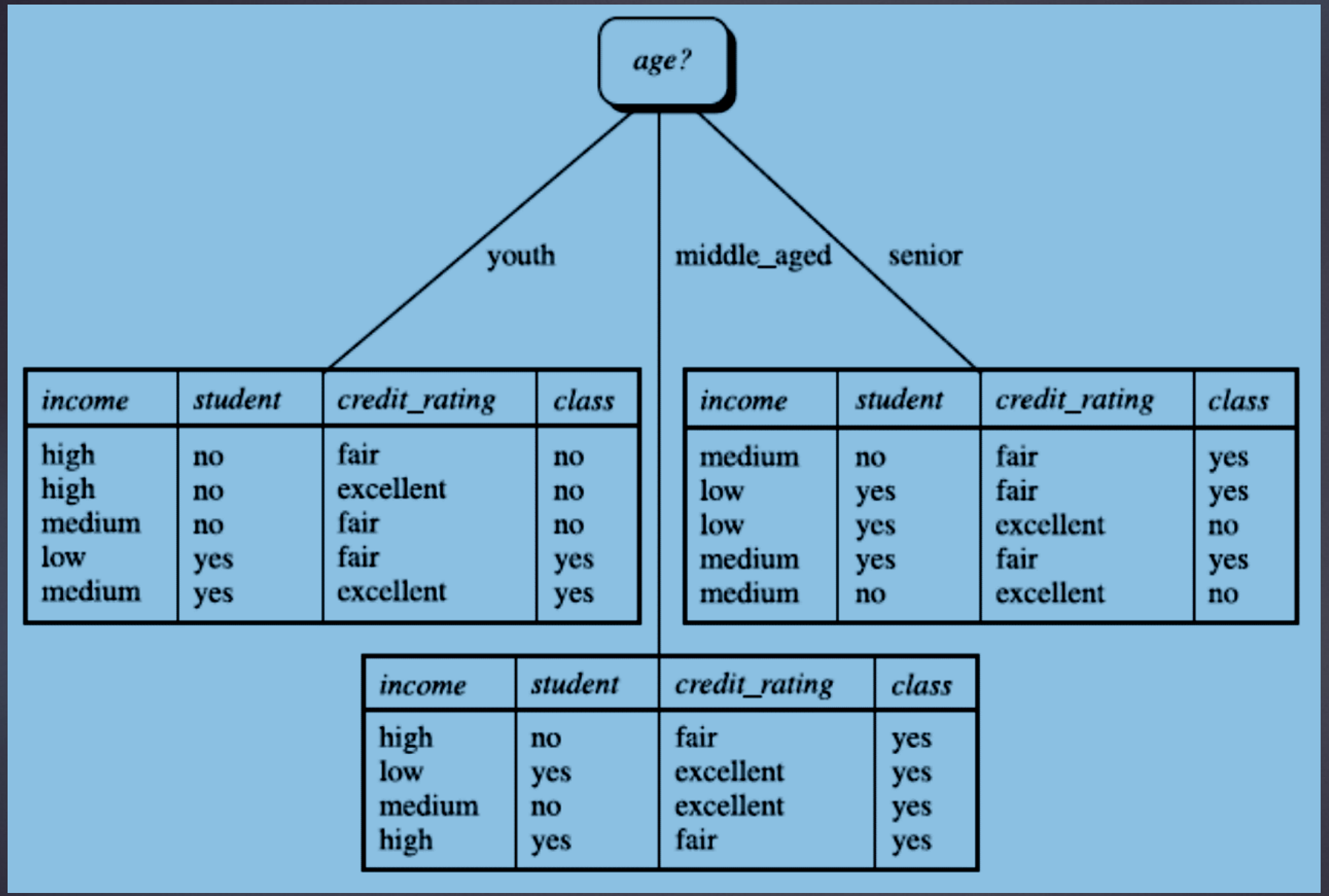
$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) = 0.694 \text{ bits.}$$

Hence, the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits}.$$

Similarly, we can compute
***Gain*(*income*)= 0.029 bits,**
***Gain*(*student*)= 0.151 bits, and**
***Gain*(*credit rating*)=0.048 bits.**
Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute.
Node *N* is labeled with *age*, and branches are grown for each of the attribute's values.
The tuples are then partitioned accordingly, as shown in Figure below.
Notice that the tuples falling into the partition for *age* D *middle aged* all belong to the same class.
Because they all belong to class *"yes"* a leaf should therefore be created at the end of this branch and labeled *"yes"*



income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

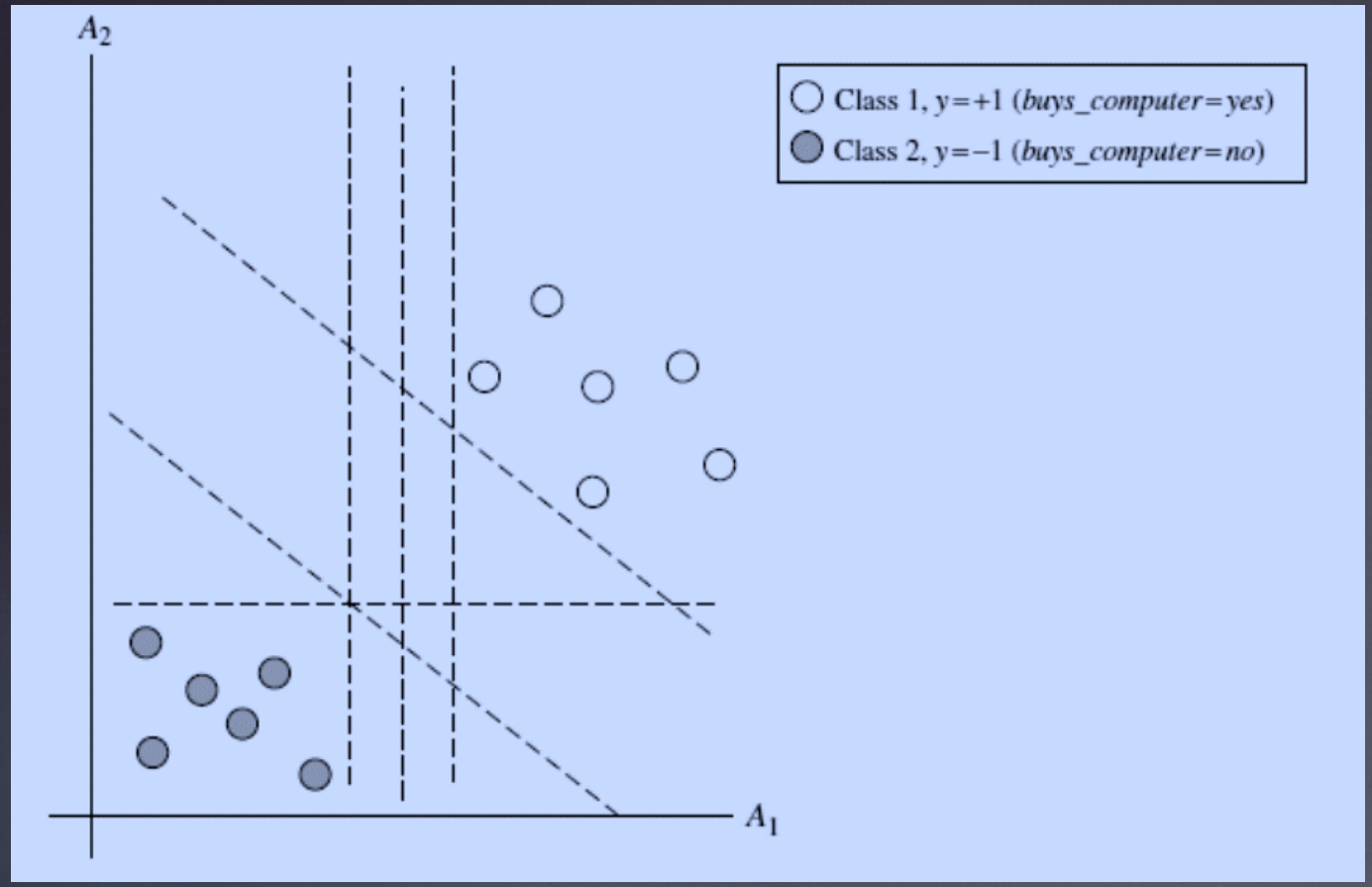
income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

- The attribute *age* has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree.
- Branches are grown for each outcome of *age*.
- The tuples are shown partitioned accordingly.

Support Vector Machines (SVM)

- SVM is a non-probabilistic linear classification approach that chooses an optimal separating hyperplane such that it maximizes the distance between data points of different classes.
- To construct the best solution for separating hyperplane, training data points are used which are considered as vectors or support vectors.
- These support vectors help in determining the width of the hyperplane. SVM can be extended for the cases where data is not separated by a hard margin.
- Hence, a trade-off parameter is used which allows margin to be flexible and separates non linearly separable data. In addition to that, SVM can be used to classify nonlinear data using kernel trick.
- This is the reason behind extensive use to SVM as it achieves to an optimal solution for both linear and nonlinear data.



The 2-D training data are linearly separable.

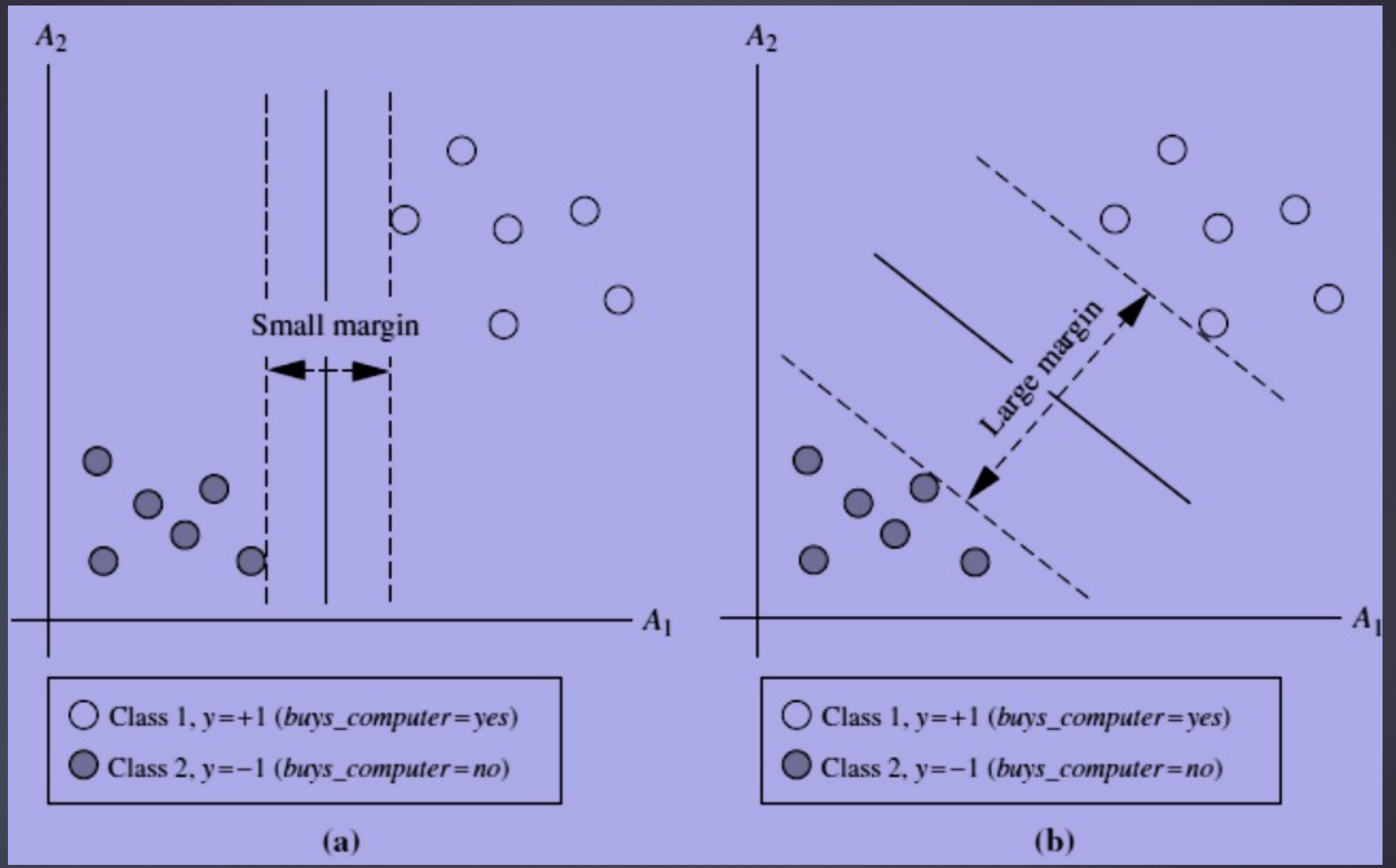
There are an infinite number of possible separating hyperplanes or “decision boundaries,” some of which are shown here as dashed lines.

Which one is best?

Generalizing to *n* dimensions, we want to find the best *hyperplane*.

We will use “hyperplane” to refer to the decision boundary that we are seeking, regardless of the number of input attributes.

- How can we find the best hyperplane?
- An SVM approaches this problem by searching for the maximum marginal hyperplane.


A_2
Small margin
A_1
Class 1, y=+1 (buys_computer=yes)
Class 2, y=−1 (buys_computer=no)
(a)
A_2
Large margin
A_1
Class 1, y=+1 (buys_computer=yes)
Class 2, y=−1 (buys_computer=no)
(b)

A separating hyperplane can be written as $W \cdot X + b = 0,$

Where $\boldsymbol{W}$ is a weight vector, $W = \{w_1, w_2, \ldots, w_n\}$

n is the number of attributes; and b is a scalar, often referred to as a bias.

Training tuples are 2-D, e.g., X = (x_1, x_2), where x_1 and x_2 are the values of attributes A_1 and A_2, respectively, for X. If we think of b as an additional weight, w_0, we can rewrite hyperplane as

$$w_0 + w_1 x_1 + w_2 x_2 = 0.$$

Thus, any point that lies above the separating hyperplane satisfies

$$w_0 + w_1 x_1 + w_2 x_2 > 0.$$

Similarly, any point that lies below the separating hyperplane satisfies

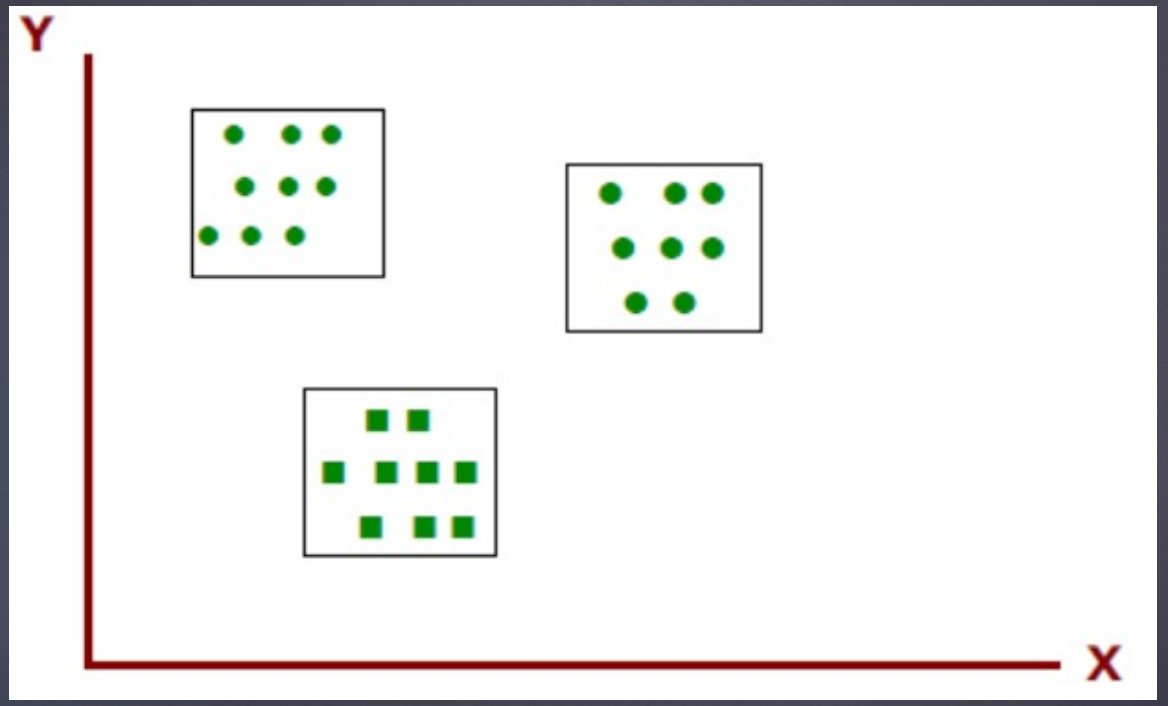
$$w_0 + w_1 x_1 + w_2 x_2 < 0.$$

Clustering

Clustering is a way to form **natural groupings** or clusters of patterns.

Clustering is often called an **unsupervised learning**.

Example : Three natural groups of data points, i.e., 3 natural clusters.



Example : Three natural groups of data points, i.e., 3 natural clusters.

k-Means Clustering

k-Means Clustering

- k-means clustering is an algorithm to classify or to group objects based on attributes/features into K number of group.
- k is positive integer number.
- The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.

Steps involved in k-means clustering

Step 1:Select the number of clusters you want to identify in the data (k)

step 2: Randomly select k distinct data points as initial cluster centre (Centroid).

step 3: Group remaining data points to the cluster centres (Centroids) based on shortest distance criteria.

step 5: Calculate mean of each clusters and rearrange the cluster centres according to new mean.

step 6: Re-cluster the remaining datapoints according to new centroids

step 7: Repeat 5 and 6 since clustering did not change at all during the last iteration.

K-means clustering

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
(5) **until** no change;

Hierarchical Methods

In some situations we may want to partition our data into groups at different levels such as in a hierarchy.

Hierarchical clustering method - works by grouping data objects into a hierarchy or “tree” of clusters.

Representing data objects in the form of a hierarchy is useful for data summarization and visualization.

Example 1:

Organize employees in a Company into major groups such as executives, managers, and staff. You can further partition these groups into smaller subgroups. For instance, the general group of staff can be further divided into subgroups of senior officers, officers, and trainees. All these groups form a hierarchy.

Advantage: We can easily summarize or characterize the data that are organized into a hierarchy, which can be used to find, say, the average salary of managers and of officers.

Hierarchical Agglomerative Clustering

Example 2: Consider handwritten character recognition as another example. A set of handwriting samples may be first partitioned into general groups where each group corresponds to a unique character. Some groups can be further partitioned into subgroups since a character may be written in multiple substantially different ways. If necessary, the hierarchical partitioning can be continued recursively until a desired granularity is reached.

Click to add text

A hierarchical clustering method can be either *agglomerative* or *divisive*, depending on whether the hierarchical decomposition is formed in a **bottom-up (merging)** or top down (splitting) fashion.

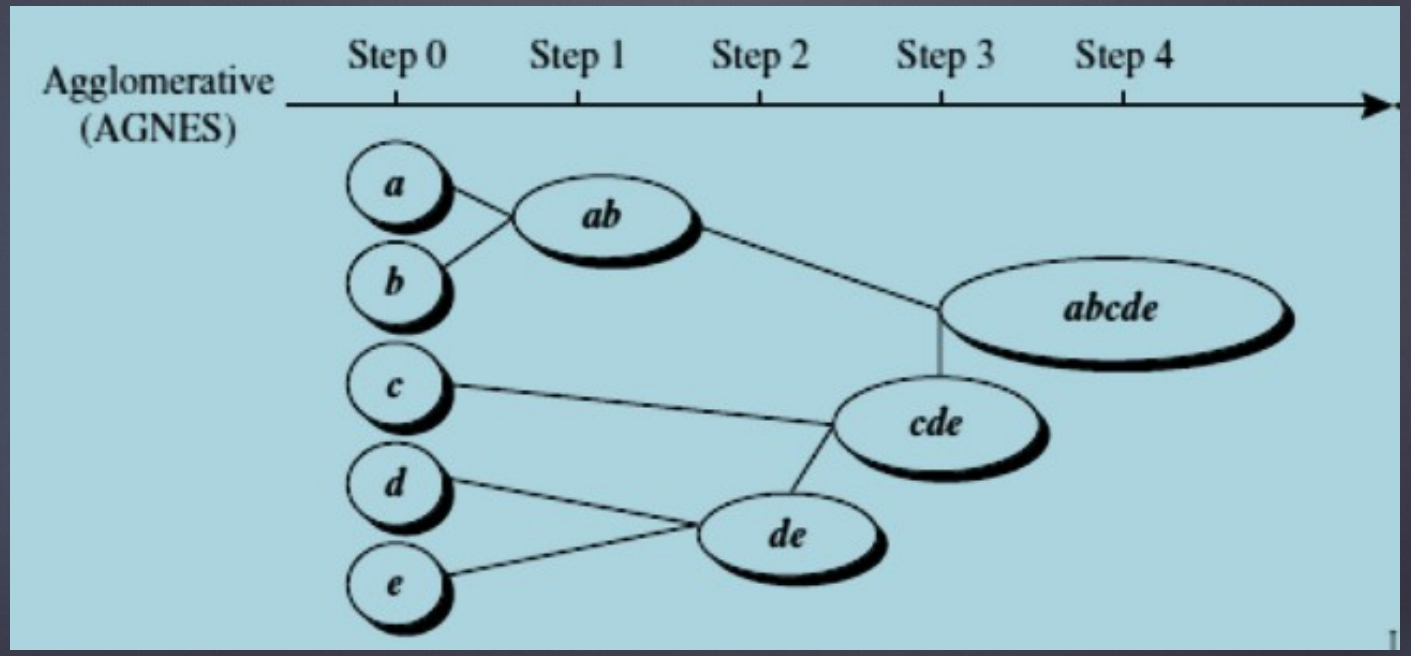
Agglomerative hierarchical clustering:

An agglomerative hierarchical clustering method uses a bottom-up strategy.

It typically starts by letting each object form its own cluster and iteratively merges clusters into larger and larger clusters, until all the objects are in a single cluster or certain termination conditions are satisfied.

Agglomerative hierarchical clustering

- The single cluster becomes the hierarchy's root.
- For the merging step, it finds the two clusters that are closest to each other (according to some similarity measure), and combines the two to form one cluster.
- Because two clusters are merged per iteration, where each cluster contains at least one object, an agglomerative method requires at most n iterations.



- Figure shows the application of AGNES (AGglomerative NESting), an agglomerative hierarchical clustering method, on a data set of five objects, {*a*, *b*, *c*, *d*, *e*}
- Initially, AGNES, places each object into a cluster of its own.
- The clusters are then merged step-by-step according to some criterion.

For example, clusters C_1 and C_2 may be merged if an object in C_1 and an object in C_2 form the minimum Euclidean distance between any two objects from different clusters. This is a single-linkage approach in that each cluster is represented by all the objects in the cluster, and the similarity between two clusters is measured by the similarity of the closest pair of data points belonging to different clusters.

- The cluster-merging process repeats until all the objects are eventually merged to form one cluster.

References:

J. Han, J. Pei, & M. Kamber, (2011). Data mining: concepts and techniques. Elsevier.

http://www.myreaders.info/html/soft_computing.html

J-S R Jang and C-T Sun, Neuro-Fuzzy and Soft Computing, Prentice Hall, 1997

S. Rajasekaran and G.A. Vijayalaksmi Pai ,Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications, (2005), Prentice Hall.

Artificial Intelligence

Chapter 5

INTERIM SEMESTER 2021-22 BPL

CSE3007-LT-AB306

FACULTY: SIMI V.R.

Artificial Neural Networks

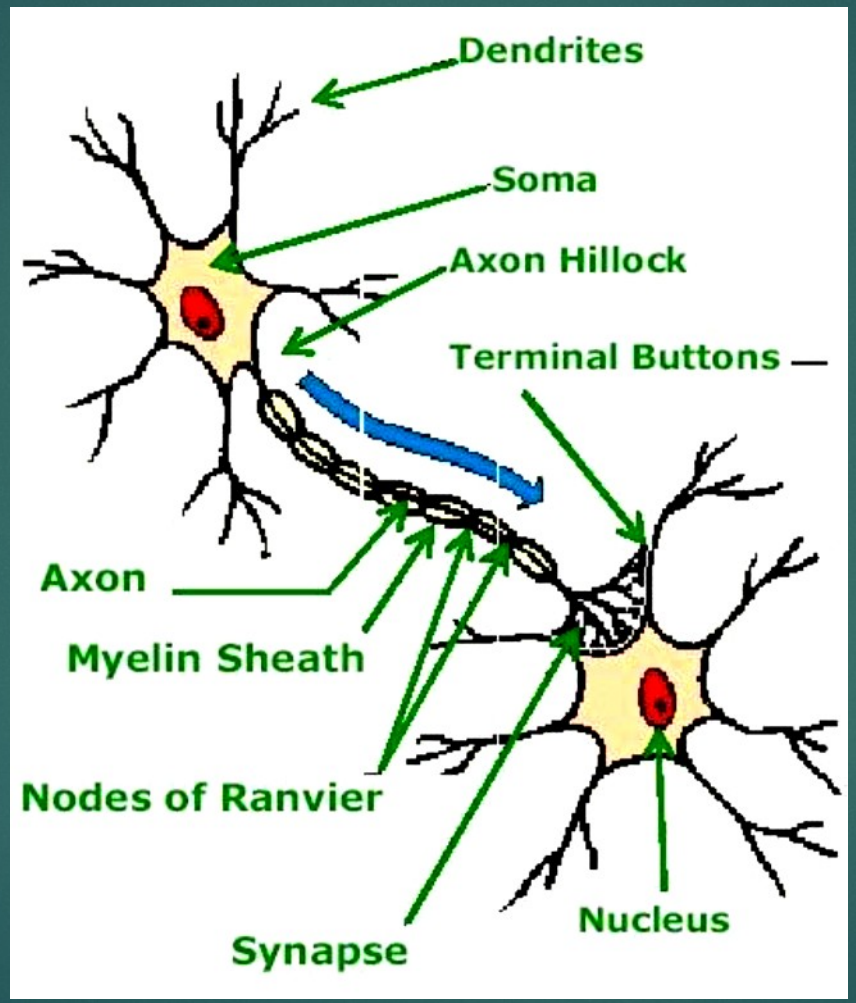
Some Facts

- Human brain contains ≈ 10^{11} neurons
- Each neuron is connected to ≈ 10^{4} others
- Some scientists compared the brain with a ‘complex, nonlinear, parallel computer’.
- The largest modern neural networks achieve the complexity comparable to a nervous system of a fly.

Biological Neuron Model

- The human brain consists of a large number, more than a billion of neural cell/neurons that process information.
- Each cell works like a simple processor.
- The massive interaction between all cells and their parallel processing only makes the brain's abilities possible.

Biological Neuron Model



- Evidence suggests that neurons receive, analyse and transmit information.

Biological Neuron Model

Dendrites are branching fibers that extend from the cell body or soma.

Soma or cell body of a neuron contains the nucleus and other structures, support chemical processing and production of neurotransmitters.

Axon is a singular fiber carries information away from the soma to the synaptic sites of other neurons (dendrites and somas), muscles, or glands.

Axon hillock is the site of summation for incoming information. At any moment, the collective influence of all neurons that conduct impulses to a given neuron will determine whether or not an action potential will be initiated at the axon hillock and propagated along the axon.

Myelin Sheath consists of fat-containing cells that insulate the axon from electrical activity. This insulation acts to increase the rate of transmission of signals. A gap exists between each myelin sheath cell along the axon. Since fat inhibits the propagation of electricity, the signals jump from one gap to the next.

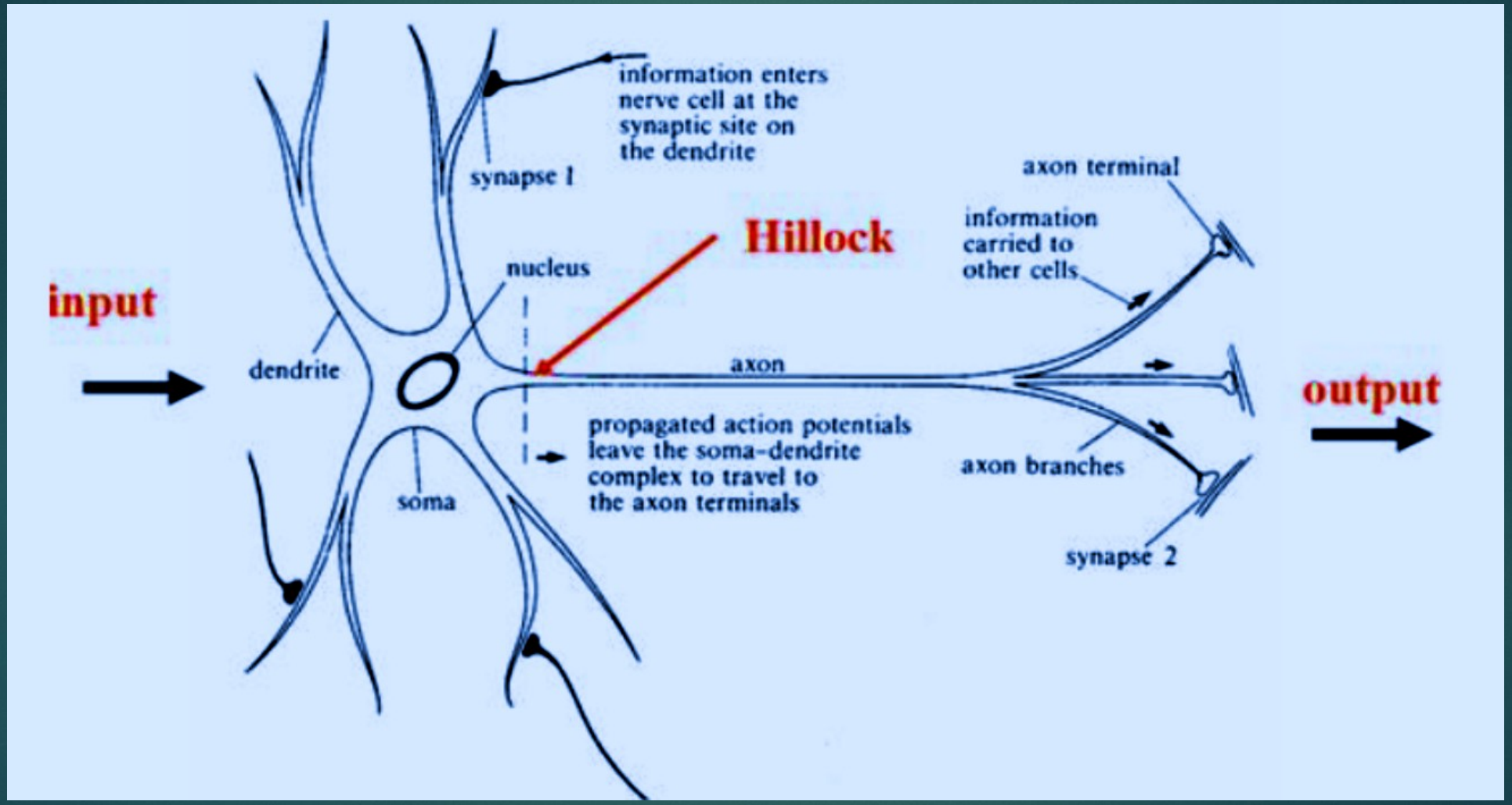
Biological Neuron Model

Nodes of Ranvier are the gaps (about 1 µm) between myelin sheath cells long axons are Since fat serves as a good insulator, the myelin sheaths speed the rate of transmission of an electrical impulse along the axon.

Synapse is the point of connection between two neurons or a neuron and a muscle or a gland. Electrochemical communication between neurons takes place at these junctions.

Terminal Buttons of a neuron are the small knobs at the end of an axon that release chemicals called neurotransmitters.

Information flow in a Neural Cell



Structure of a Neural Cell in the Human Brain

Information flow in a Neural Cell

- **Dendrites** receive activation from other neurons.
- **Soma** processes the incoming activations and converts them into output activations.
- **Axons** act as transmission lines to send activation to other neurons.
- **Synapses** the junctions allow signal transmission between the axons and dendrites.

The process of transmission is by diffusion of chemicals called neuro-transmitters.

Information flow in a Neural Cell

Excitation and Inhibition

- The receptors of a neuron are called synapses, and they are located on many branches, called dendrites.
- There are many types of synapses, but roughly they can be divided into two classes:

  Excitatory a signal: received at this synapse ‘encourages’ the neuron to **fire**.

  Inhibitory a signal: received at this synapse inhibits the neuron (as if asking to **‘shut up’**)

- The neuron analyses all the signals received at its synapses.

  If most of them are ‘**encouraging**’, then the neuron gets ‘**excited**’ and fires its own message along a single wire, called **axon**.

- The axon may have branches to reach many other neurons.

Simplified model of real neurons

- A set of input connections brings in activations from other neurons.
- A processing unit sums the inputs, and then applies a non-linear

  activation function (i.e. squashing / transfer / threshold function).
- An output line transmits the result to other neurons.

In other words ,

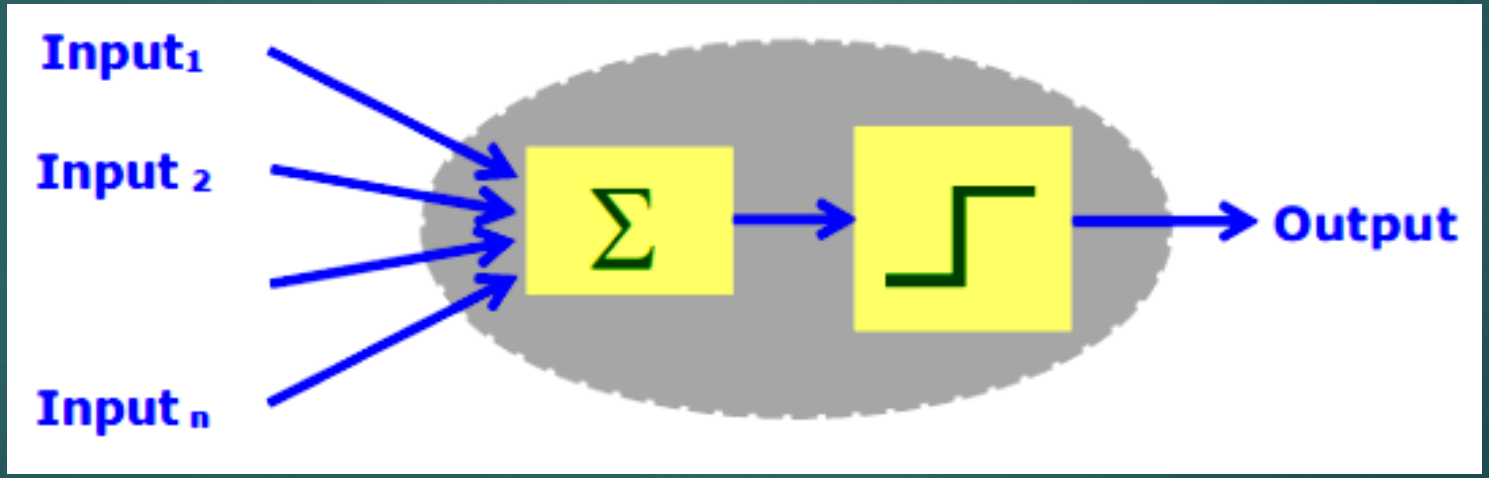
- The input to a neuron arrives in the form of signals.
- The signals build up in the cell.
- Finally the cell discharges (cell fires) through the output .
- The cell can start building up signals again.

Artificial Neuron Model

McCulloch-Pitts (M-P) Neuron Equation

McCulloch-Pitts neuron is a simplified model of real biological neuron.

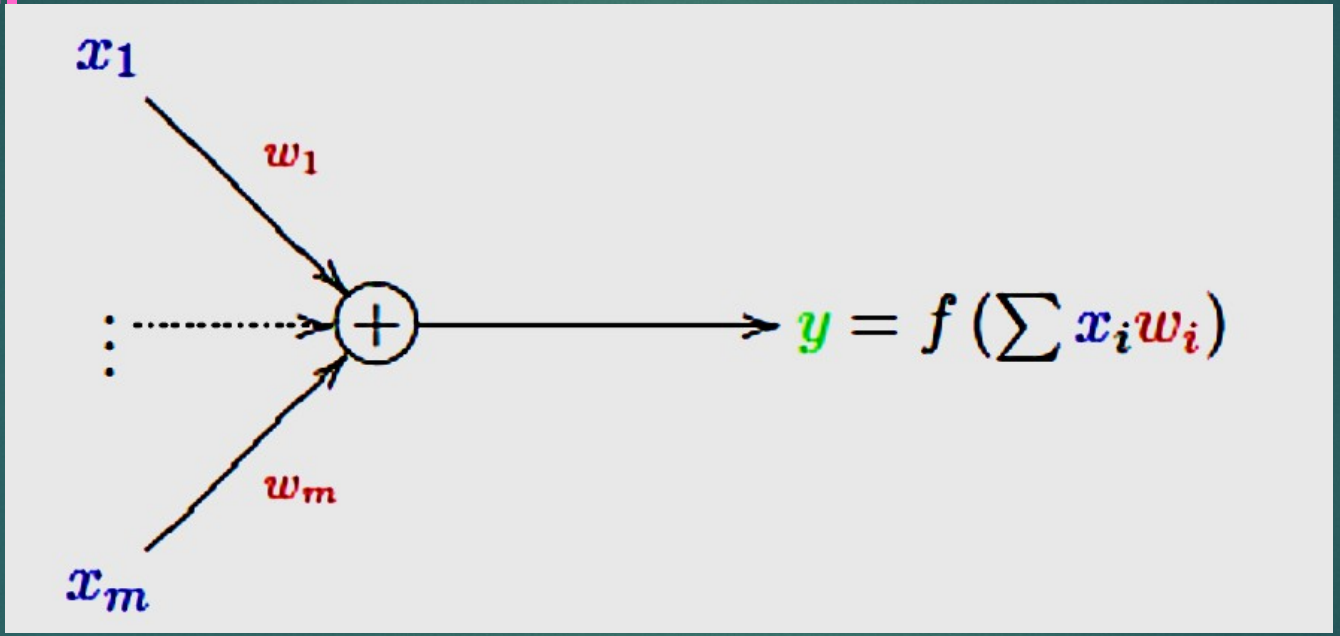
An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.



Simplified model of real neurons, known as a Threshold Logic Unit.

A Model of A Single Neuron

- A Model of a Single Neuron (Unit)
- McCulloch and Pitts (1943) proposed the ‘integrate and fire’ model:



- Denote the m input values by $x_1, x_2, \ldots, x_m$.
- Each of the m inputs (synapses) has a weight $w_1, w_2, \ldots, w_m$.
- The input values are multiplied by their weights and summed

$$v = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m = \sum_{i=1}^{m} w_i x_i$$

The output is some function $y = f(v)$ of the *weighted sum*

Example 1. Let $x = (0, 1, 1)$ and $w = (1, -2, 4)$. Then

$$v = 1 \cdot 0 - 2 \cdot 1 + 4 \cdot 1 = 2$$

Activation Function

- The output of a neuron (y) is a function of the weighted sum

$$y = f(v)$$

- This function is often called the **activation function.**
- What function is it and how is it computed?

Linear function:

$$f(v) = a + v = a + \sum w_i x_i$$

where parameter a is called **bias.**

Noice that in this case, a neuron becomes a linear model with bias a being the *intercept* and the weights, $w_1, \ldots, w_m$, being the *slopes.*

Functions

- The output of a neuron (y) is a function of the weighted sum

$$y = f(v)$$

- This function is often called the **activation function**.
- What function is it and how is it computed?

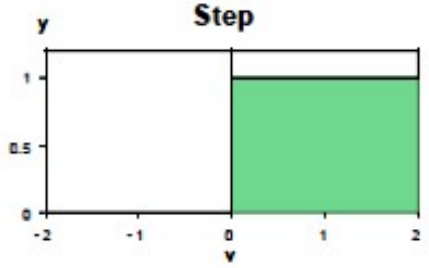
Heviside **step** function:

$$f(v) = \begin{cases} 1 & \text{if } v \geq a \\ 0 & \text{otherwise} \end{cases}$$

Here a is called the **threshold**

Example

If $a = 0$ and $v = 2 > 0$, then $f(2) = 1$, the neuron fires



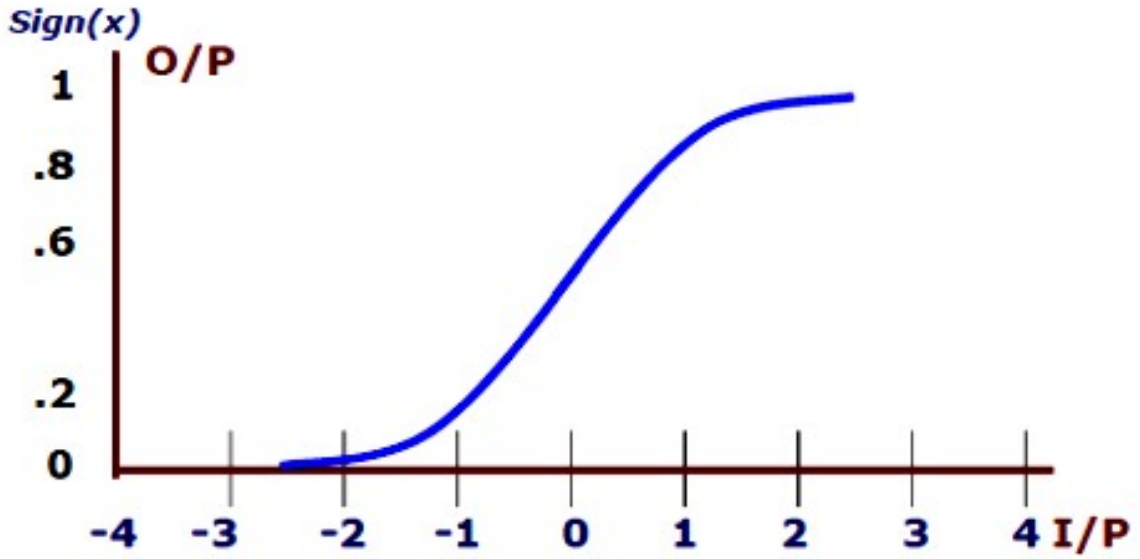
If $\sum_{i=1}^{n}$ Input i $\geq \Phi$ then Output = 1

If $\sum_{i=1}^{n}$ Input i $< \Phi$ then Output = 0

Sigmoid function

sigmoid(x) defined as a smoothed (differentiable) form of the threshold function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



Artificial Neuron - Basic Elements

Neuron consists of three basic components - weights, thresholds, and a single activation function.

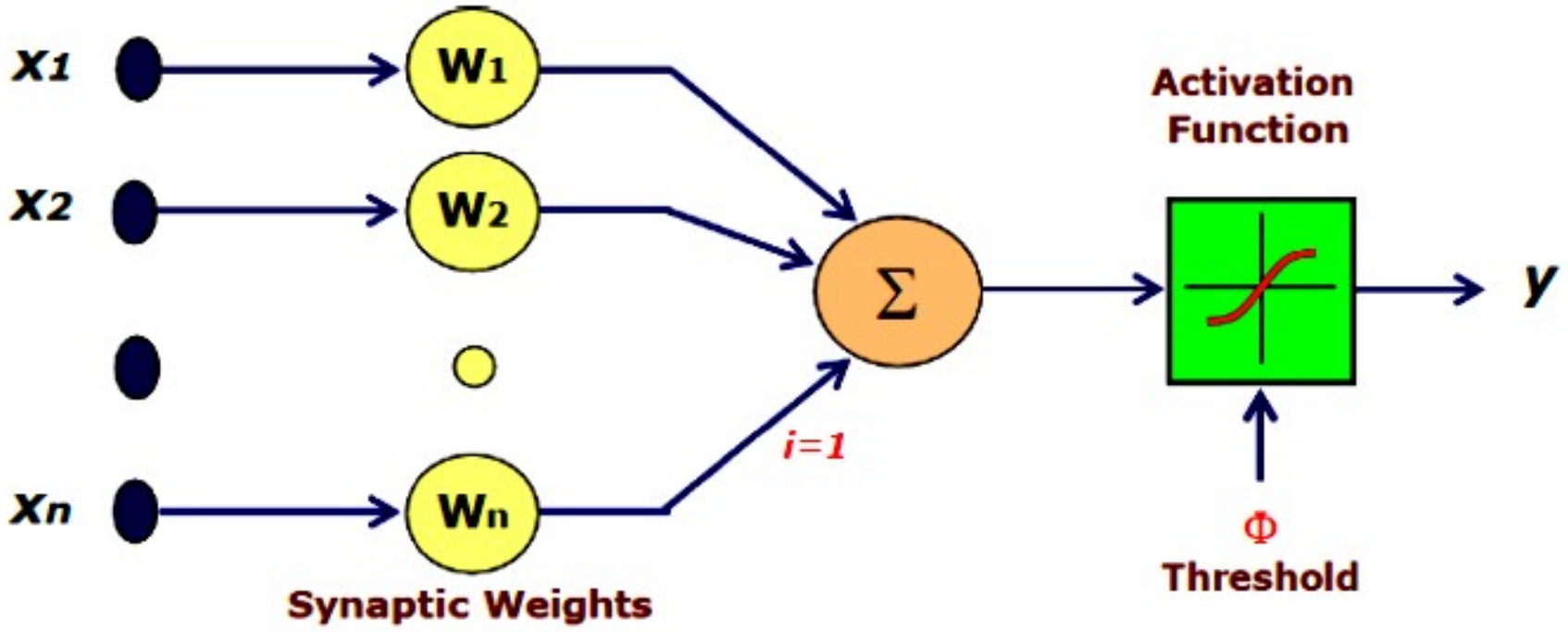


Fig Basic Elements of an Artificial Linear Neuron

- **Weighting Factors w**

The values $w_1, w_2, \ldots w_n$ are weights to determine the strength of input vector $X = [x_1, x_2, \ldots, x_n]^T$. Each input is multiplied by the associated weight of the neuron connection $X^T W$. The +ve weight excites and the -ve weight inhibits the node output.

$$I = X^T.W = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n = \sum_{i=1}^{n} x_i w_i$$

- **Threshold Φ**

The node's internal threshold Φ is the magnitude offset. It affects the activation of the node output **y** as:

$$Y = f(I) = f\{\sum_{i=1}^{n} x_i w_i - \Phi_k\}$$

To generate the final output **Y**, the sum is passed on to a non-linear filter **f** called Activation Function or Transfer function or Squash function which releases the output **Y**.

- **Threshold for a Neuron**

  In practice, neurons generally do not fire (produce an output) unless their total input goes above a threshold value.

  The total input for each neuron is the sum of the weighted inputs to the neuron minus its threshold value. This is then passed through the sigmoid function.

- **Activation Function**

An activation function **f** performs a mathematical operation on the signal output. The most common activation functions are:

- Linear Function,
- Piecewise Linear Function,
- Tangent hyperbolic function
- Threshold Function,
- Sigmoidal (S shaped) function,

The activation functions are chosen depending upon the type of problem to be solved by the network.

Activation Functions f - Types

Over the years, researches tried several functions to convert the input into an outputs. The most commonly used functions are described below.

- **I/P** Horizontal axis shows sum of inputs .
- **O/P** Vertical axis shows the value the function produces ie output.
- All functions **f** are designed to produce values between **0** and **1**.

Role of the Activation functions in Neural Networks

The reason for using activation functions in Neural Networks are as follows:

1. The idea behind the activation function is to introduce nonlinearity into the neuron so that it can learn more complex functions.

2. Without the Activation function, the neural network behaves as a linear classifier, learning the function which is a linear combination of its input data.

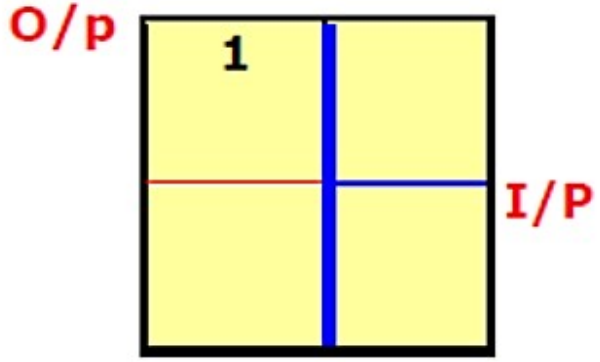
4. The activation function is responsible for deciding whether a neuron should be activated i.e, fired or not.

5. To make the decision, firstly it calculates the weighted sum and further adds bias with it.

Threshold Function

A threshold (hard-limiter) activation function is either a binary type or a bipolar type as shown below.

binary threshold Output of a binary threshold function produces :



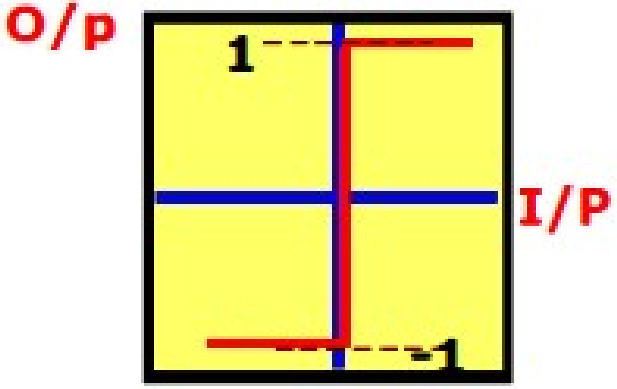
1 if the weighted sum of the inputs is +ve,

0 if the weighted sum of the inputs is -ve.

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

Neuron with hard limiter activation function is called McCulloch-Pitts model.

bipolar threshold Output of a bipolar threshold function produces :



1 if the weighted sum of the inputs is +ve,

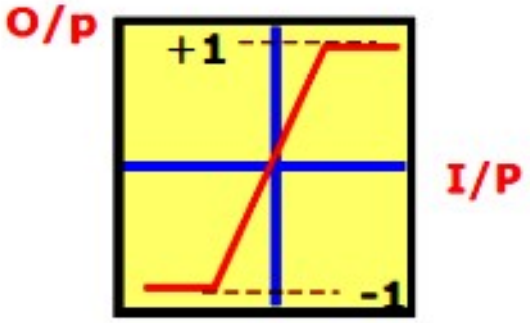
-1 if the weighted sum of the inputs is -ve.

$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ -1 & \text{if } I < 0 \end{cases}$$

Piecewise Linear Function

This activation function is also called saturating linear function and can have either a binary or bipolar range for the saturation limits of the output. The mathematical model for a symmetric saturation function is described below.

Piecewise Linear



This is a sloping function that produces :

-1 for a -ve weighted sum of inputs,

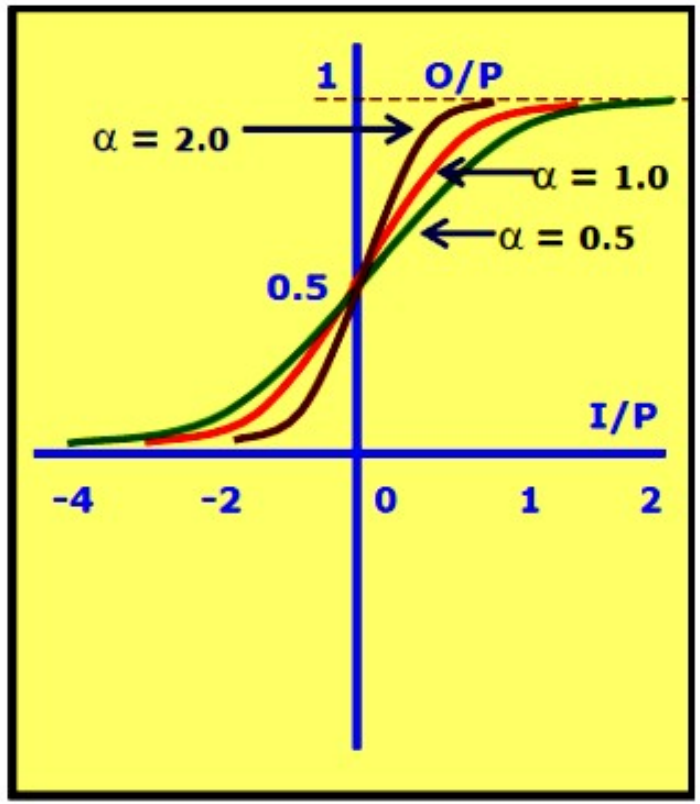
1 for a +ve weighted sum of inputs.

$\propto$ **I** proportional to input for values between **+1** and **-1** weighted sum,

$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ I & \text{if } -1 \geq I \geq 1 \\ -1 & \text{if } I < 0 \end{cases}$$

Sigmoidal Function (S-shape function)

- The nonlinear curved S-shape function is called the sigmoid function.
- This is most common type of activation used to construct the neural networks.
- It is mathematically well behaved, differentiable and strictly increasing function.



A sigmoidal transfer function can be written in the form:

$$Y = f(I) = \frac{1}{1 + e^{-\alpha I}}, \quad 0 \le f(I) \le 1$$

$$= 1/(1 + \exp(-\alpha I)), \quad 0 \le f(I) \le 1$$

This is explained as

≈ **0** for large **-ve** input values,

1 for large **+ve** values, with a smooth transition between the two.

α is slope parameter also called shape parameter; symbol the λ is also used to represented this parameter.

- The sigmoidal function is achieved using exponential equation.
- By varying **α** different shapes of the function can be obtained which adjusts the abruptness of the function as it changes between the two asymptotic values.

Example : 1

The neuron shown consists of four inputs with the weights.

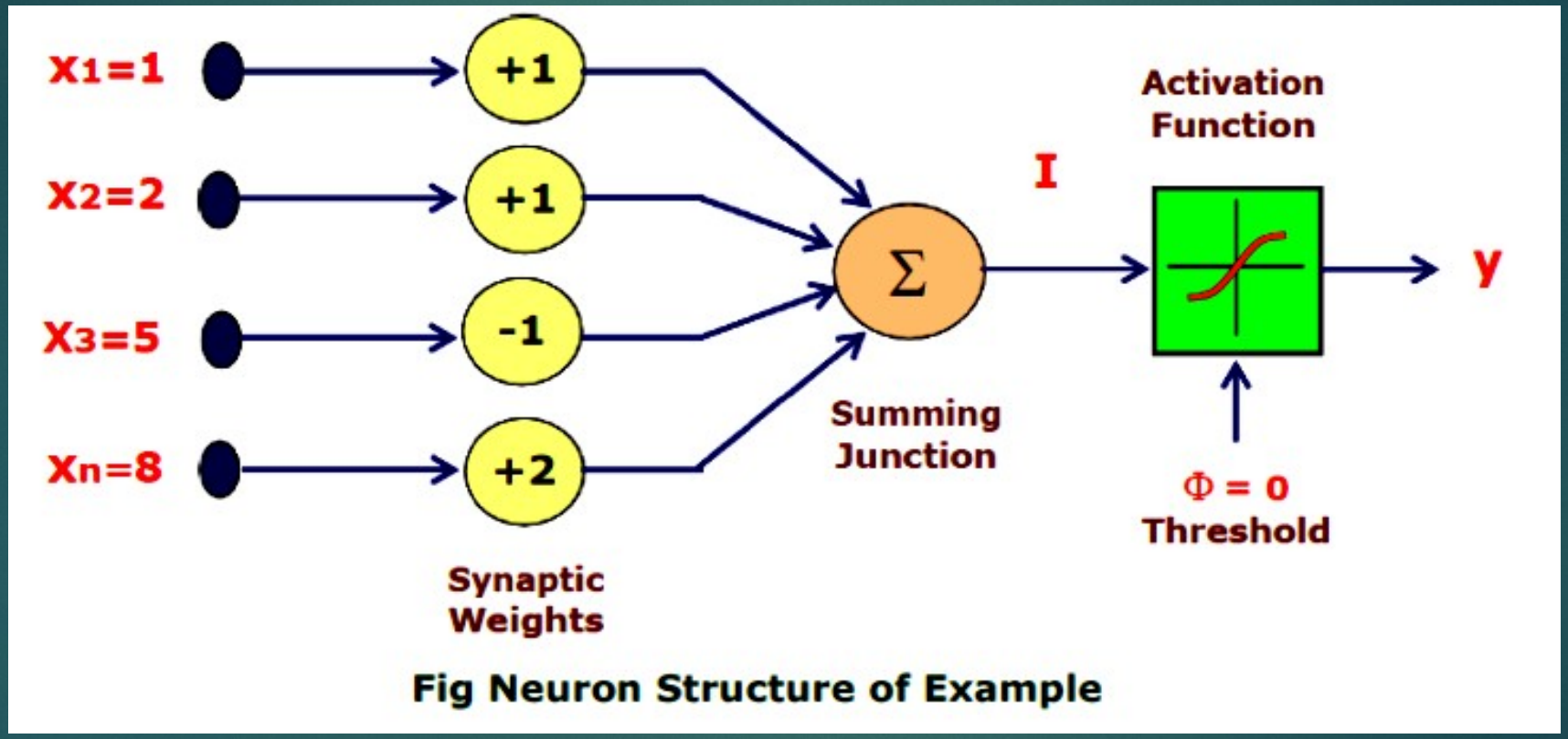


Fig Neuron Structure of Example

The output I of the network, prior to the activation function stage, is

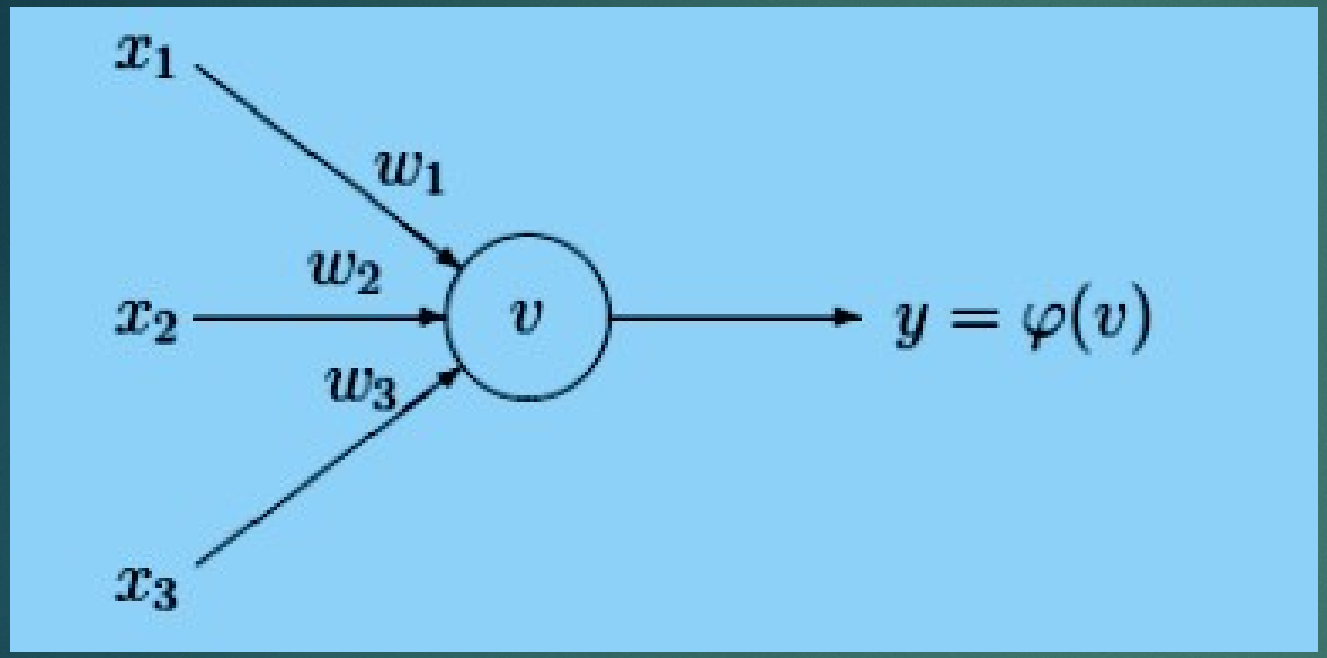
$$I = X^T . W = \begin{bmatrix} 1 & 2 & 5 & 8 \end{bmatrix} \bullet \begin{pmatrix} +1 \\ +1 \\ -1 \\ +2 \end{pmatrix} = 14$$

$$= (1 \times 1) + (2 \times 1) + (5 \times -1) + (8 \times 2) = 14$$

With a binary activation function the outputs of the neuron is:

y (threshold) = 1;

Example 2



Single unit with three inputs

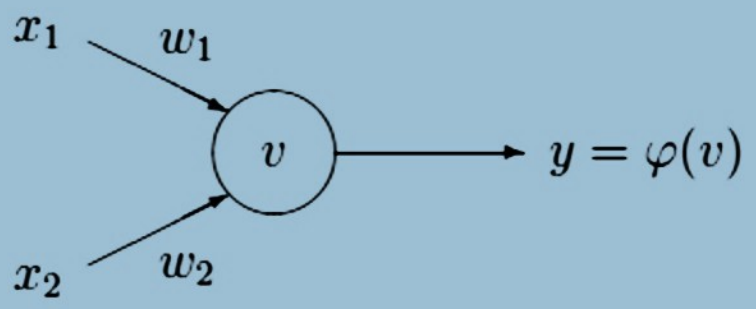
The node has three inputs x = (x_1, x_2, x_3) that receive only binary signals (either 0 or 1). How many different input patterns this node can receive?
Can you give a formula that computes the number of binary input patterns for a given number of inputs?

Example 3

Logical operators (i.e. NOT, AND, OR, XOR, etc) are the building blocks of any computational device. Logical functions return only two possible values, true or false, based on the truth or false values of their arguments. For example, operator AND returns true only when all its arguments are true, otherwise (if any of the arguments is false) it returns false. If we denote truth by 1 and false by 0, then logical function AND can be represented by the following table:

x_1 :	0	1	0	1
x_2 :	0	0	1	1
x_1 AND x_2 :	0	0	0	1

This function can be implemented by a single-unit with two inputs:



if the weights are $w_1 = 1$ and $w_2 = 1$ and the activation function is:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

Note that the threshold level is 2 ($v \geq 2$).

a) Test how the neural AND function works.

Answer:

$$\begin{aligned}
P_1: &\quad v = 1 \cdot 0 + 1 \cdot 0 = 0\,, \quad (0 < 2)\,, \quad y = \varphi(0) = 0 \\
P_2: &\quad v = 1 \cdot 1 + 1 \cdot 0 = 1\,, \quad (1 < 2)\,, \quad y = \varphi(1) = 0 \\
P_3: &\quad v = 1 \cdot 0 + 1 \cdot 1 = 1\,, \quad (1 < 2)\,, \quad y = \varphi(1) = 0 \\
P_4: &\quad v = 1 \cdot 1 + 1 \cdot 1 = 2\,, \quad (2 = 2)\,, \quad y = \varphi(2) = 1
\end{aligned}$$

Neural Networks Architectures

An Artificial Neural Network (ANN) is a data processing system, consisting large number of simple highly interconnected processing elements as artificial neuron in a network structure that can be represented using a directed graph G, an ordered 2-tuple (V, E) , consisting a set V of vertices and a set E of edges.

- The vertices may represent neurons (input/output) and

- The edges may represent synaptic links labeled by the weights attached.

Example

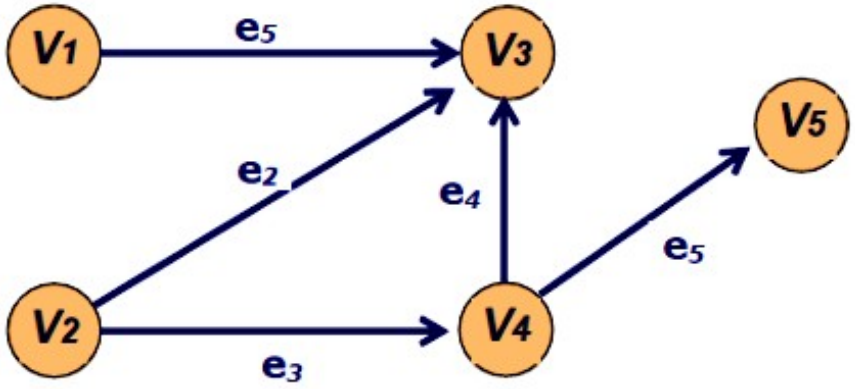


Fig. Directed Graph

Vertices V = { v_1 , v_2 , v_3 , v_4, v_5 }

Edges E = { e_1 , e_2 , e_3 , e_4, e_5 }

Advantages of Neural Networks

- Can be applied to many problems, as long as there is some data.
- Can be applied to problems, for which analytical methods do not yet exist.
- Can be used to model non-linear dependencies.
- If there is a pattern, then neural networks should quickly work it out, even if the data is ‘noisy’.
- Always gives some answer even when the input information is not complete.
- Networks are easy to maintain.

Limitations of Neural Networks

- Like with any data-driven models, they cannot be used if there is no or very little data available.
- There are many free parameters, such as the number of hidden nodes, the learning rate, minimal error, which may greatly influence the final result.
- Not good for arithmetic and precise calculations.
- Neural networks do not provide explanations. If there are many nodes, then there are too many weights that are difficult to interprete (unlike the slopes in linear models, which can be seen as correlations). In some tasks, explanations are crucial (e.g. air traffic control, medical diagnosis).

There are several classes of Neural Networks, classified according to their learning mechanism.

Fundamentally three different classes of Networks

1. Single Layer Feedforward Network
2. Multilayer Feedforward Network
3. Recurrent Networks

1. Single Layer Feedforward Network

Network comprises of two layers:

- Input Layer
- Output Layer

The input layer neurons receive the input signals and the output layer neurons receive the output signals.

The synaptic links carrying the weights connect every input neuron to the output neuron but not vice-versa. Such a network is said to be *feedforward* in type or acyclic in nature

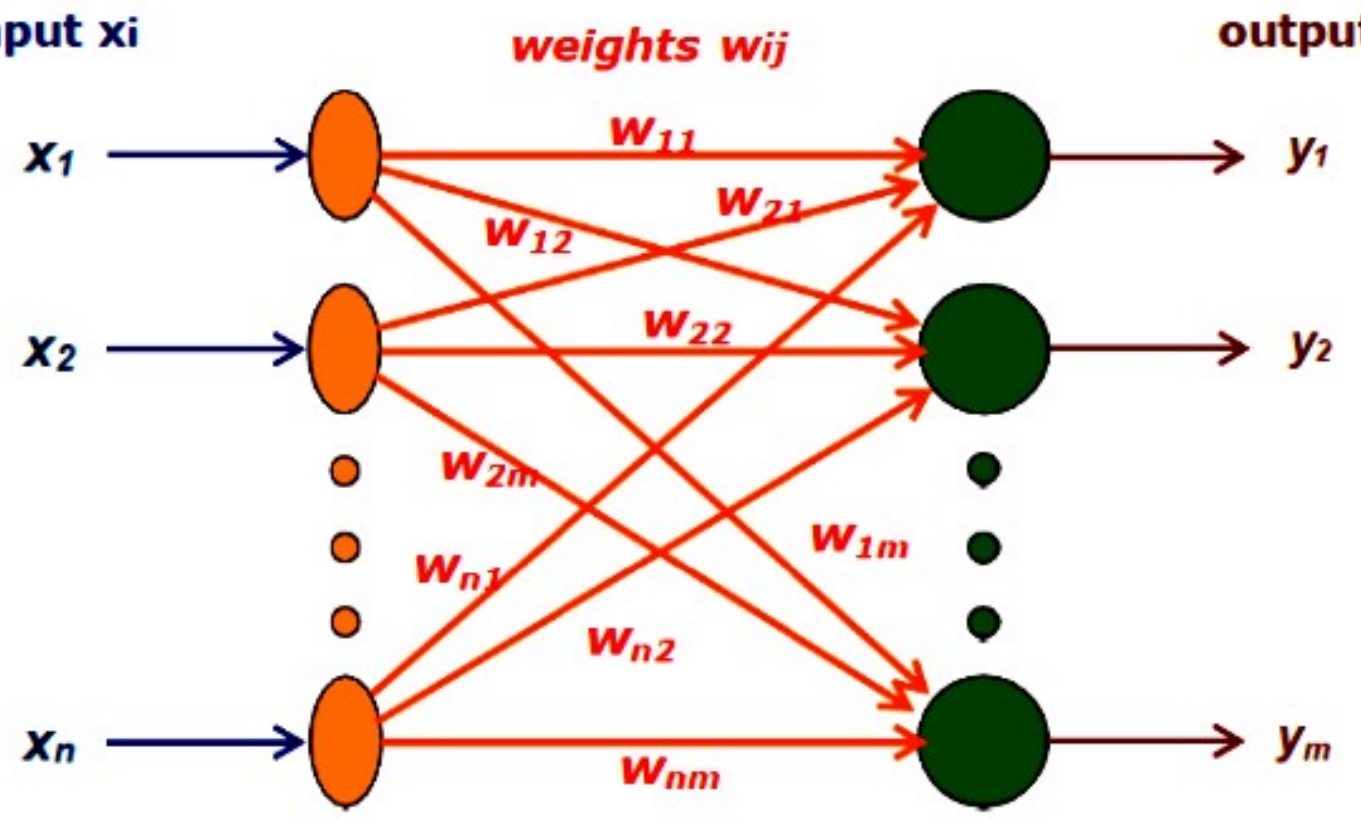


Fig. Single Layer Feed-forward Network

The sum of the products of the weights and the inputs is calculated in each neuron node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1).

X_i : Input neurons

Y_j : Output neurons

W_{ij} : Weights

Networks without cycles (feedback loops) are called a feed-forward networks (or perceptron).

2. Multilayer Feedforward Network

- The name suggests, it consists of multiple layers.
- The architecture of this class of network, besides having the input and the output layers, also have one or more intermediary layers called **hidden layers**.
- The computational units of the hidden layer are known as hidden neurons.
- Hidden layer aids in performing useful intermediatory computations before directing the input layer to the output layer.
- The input neurons are linked to the hidden layer neurons and the weights on these links are referred to as **input-hidden layer weights**.
- Again the hidden layer neurons are lined to the output layer neurons and the corresponding weights are referred to as **hidden-output layer weights**.
- A multilayer feedforward network with **l** input neurons in the input layer, **m** neurons in the hidden layer and **n** neurons in the output layer is with a configuration **l-m-n**.

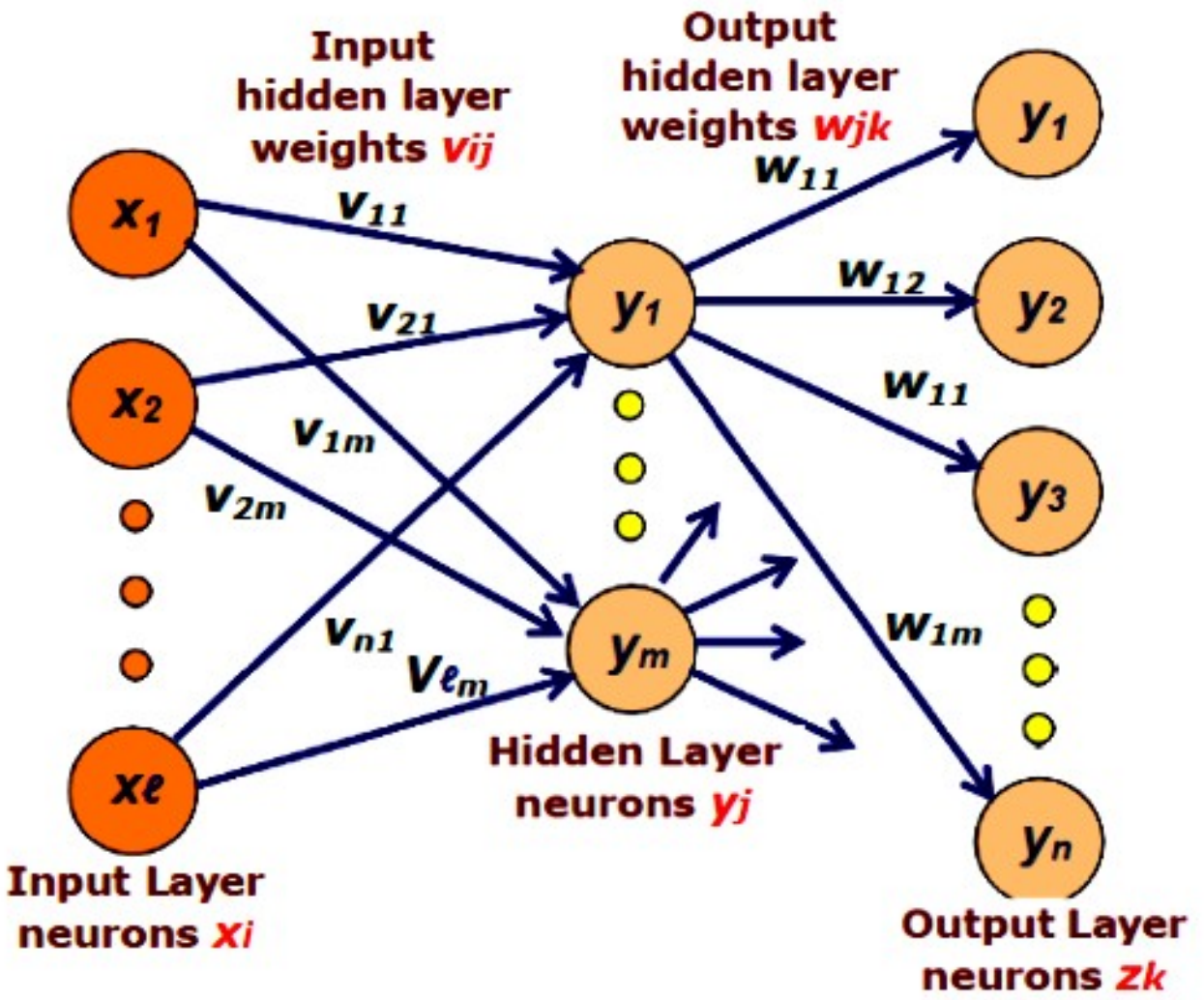


Fig. Multilayer feed-forward network in *(ℓ – m – n)* configuration.

A multi-layer feed-forward network with ℓ input neurons, m_1 neurons in the first hidden layers, ***m2*** neurons in the second hidden layers, and ***n*** output neurons in the output layers is written as **(ℓ - m_1 - m_2 – n).**

3. Recurrent networks

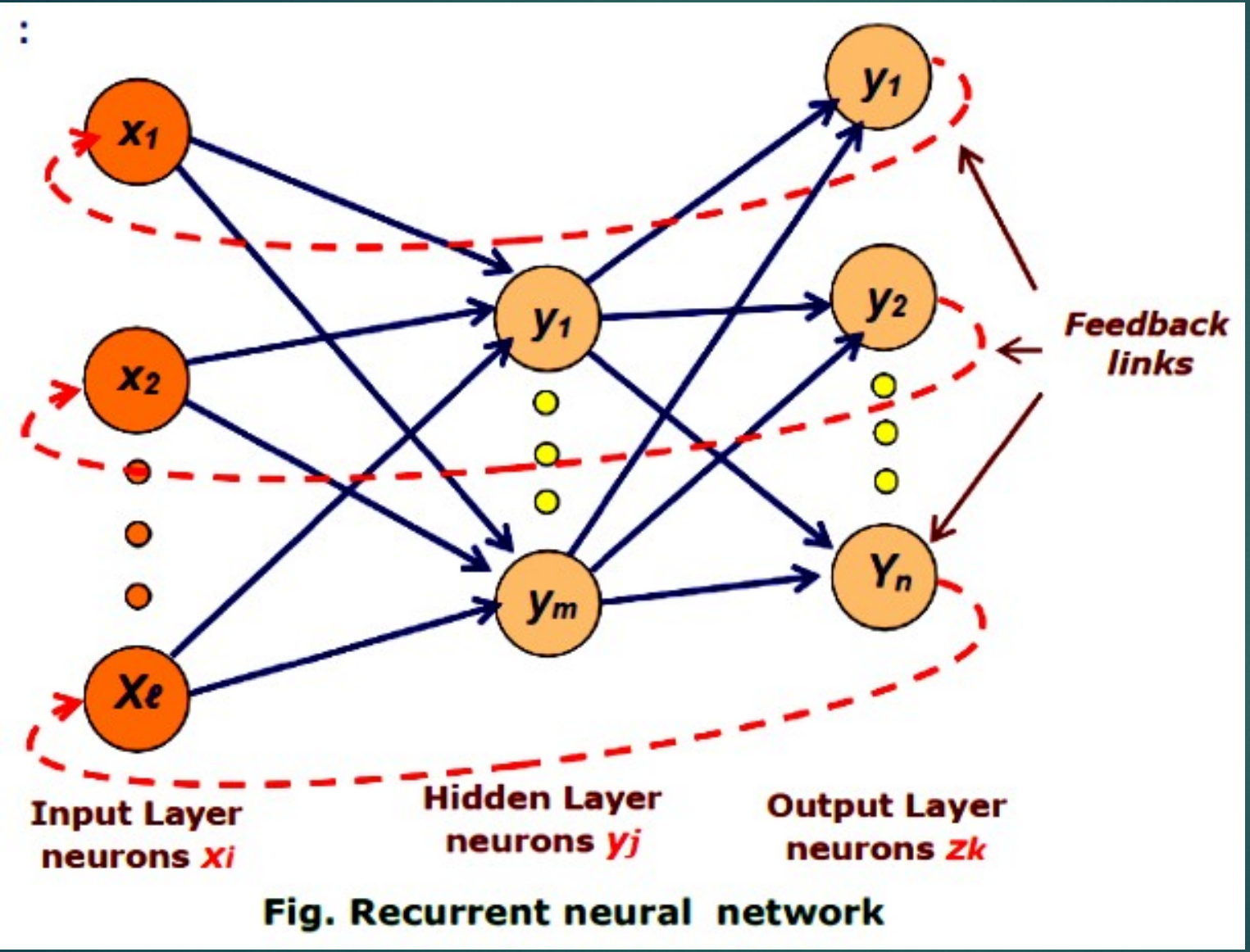


Fig. Recurrent neural network

- The Recurrent Networks differ from feed-forward architecture.
- A Recurrent network has at least one feed back loop.
- There could be neurons with self-feedback links; that is the output of a neuron is feedback into itself as input.

Characteristics of Neural Networks

- The NNs exhibits mapping capabilities, that is they can map input patterns to their associated output patterns
- The NNs learn by examples. Thus NN architectures can be trained with known examples of a problem before they are tested for their ‘inference’ capability on unknown instances of the problem. They can, therefore, identify new objects previously untrained.
- The NN possess the capability to generalize. Thus they can predict new outcomes from past trends.
- The NN are robust systems and are fault tolerant. They can therefor, recall full patterns from incomplete, partial or noisy patterns.
- The NNs can process information in parallel, at high speed, and in a distributed manner

Classification of Learning Algorithms

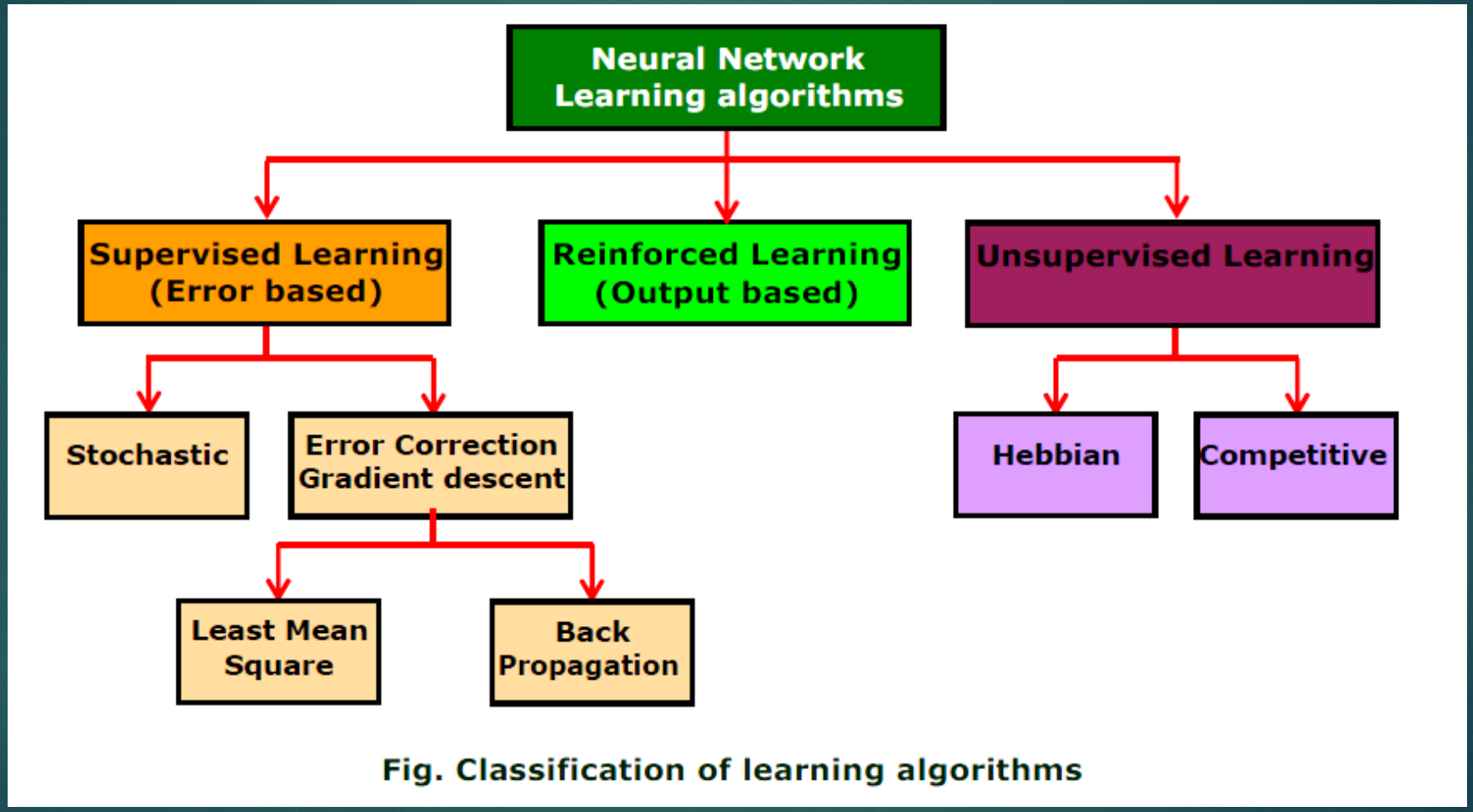


Fig. Classification of learning algorithms

Learning methods in Neural Networks

The learning methods in neural networks are classified into three basic types :

- Supervised Learning,
- Unsupervised Learning
- Reinforced Learning

1. Supervised Learning

- Every input pattern that is sued to train the network is associated with an output pattern, which is the target or the desired pattern.
- Learning process is based on comparison, between network's computed output and the correct expected output, generating "error".
- The "error" generated is used to change network parameters that result improved performance.

2. Unsupervised Learning

- The expected or desired output is not presented to the network. Ie. Target output is not presented to the network.
- The system learns of it own by discovering and adapting to the structural features in the input patterns.

3. Reinforced Learning

- Indicated if the computed output is correct or incorrect.
- The information provided helps the network in its learning process.
- A reward is given for correct answer computed and a penalty for a wrong answer.

Note : The Supervised and Unsupervised learning methods are most popular forms of learning compared to Reinforced learning.

Hebbian Learning

- Hebb proposed a rule based on correlative weight adjustment.
- In this rule, the input-output pattern pairs ***(Xi , Yi)*** are associated by the weight matrix ***W***, known as ***correlation matrix*** computed as

$$W = \sum_{i=1}^{n} X_i Y_i^T$$

where Y_i^T is the transpose of the associated output vector Y_i

- Oldest learning mechanism inspired by biology
- There are many variations of this rule proposed by the other researchers (Kosko, Anderson, Lippman) .

Gradient Descent Learning

- **This is based on the minimization of errors *E* defined in terms of weights and the activation function of the network.**
- **Here, the activation function of the network is required to be differentiable, because the updates of weight is dependent on the gradient of the error *E*.**

If ΔW_{ij} is the weight update of the link connecting the i^{th} and the j^{th} neuron of the two neighboring layers, then ΔW_{ij} is defined as

$$\Delta W_{ij} = \eta \left(\partial E / \partial W_{ij}\right)$$

where η is the learning rate parameters and $\left(\partial E / \partial W_{ij}\right)$ is error gradient with reference to the weight W_{ij}.

Note : The Hoffs Delta rule and Back-propagation learning rule are the examples of Gradient descent learning.

Competitive Learning

- In this method, those neurons which respond strongly to the input stimuli have their weights updated.
- When an input pattern is presented, all neurons in the layer compete, and the winning neuron undergoes weight adjustment .
- This strategy is called "winner-takes-all".

Stochastic learning

- In this method the weights are adjusted in a probabilistic fashion.

Example : Simulated annealing which is a learning mechanism employed by Boltzmann and Cauchy machines.

Taxonomy of Neural Network Architectures

Over the years, several NN systems have evolved. The following are some of the systems that have acquired significance.

- ADALINE (Adaptive Linear Neural Element)
- ART (Adaptive Resonance Theory)
- AM (Associative Memory)
- BAM (Bidirectional Associative Memory)
- Boltzmann Machine
- BSB (Brain-State-in-a-Box)
- CCN (Cascade Correlation)
- Cauchy Machine
- CPN (Counter Propagation Network)
- Hamming Network
- Hopfield Network
- LVQ (Learning Vector Quantization)
- MADALINE (Many ADALINE)
- MLFF (Multilayer Feedforward Network)
- Neocognitron
- Perceptron
- RBF (Radial Basis Function)
- RNN (Recurrent Neural Network)
- SOFM (Self-organizing Feature Map)

The classification of some NN system with respect to learning methods and architecture types

TYPE OF ARCHITECTURE	LEARNING METHOD				
		Gradient descent	*Hebbian*	*Competitive*	*Stochastic*
	Single-layer feedforward	ADALINE Hopfield Perceptron	AM Hopfield	LVQ SOFM	—
	Multilayer feedforward	CCN MLFF RBF	Neocognitron	—	—
	Recurrent neural network	RNN	BAM BSB Hopfield	ART	Boltzmann machine Cauchy machine

Single layer neural network system

A simple Perceptron Model

Single Layer Perceptron

Definition : An arrangement of one input layer of neurons feed forward to one output layer of neurons is known as Single Layer Perceptron.

$$y_j = f(net_j) = \begin{cases} 1 & \text{if } net_j \geq 0 \\ 0 & \text{if } net_j < 0 \end{cases} \qquad \text{where } net_j = \sum_{i=1}^{n} x_i w_{ij}$$

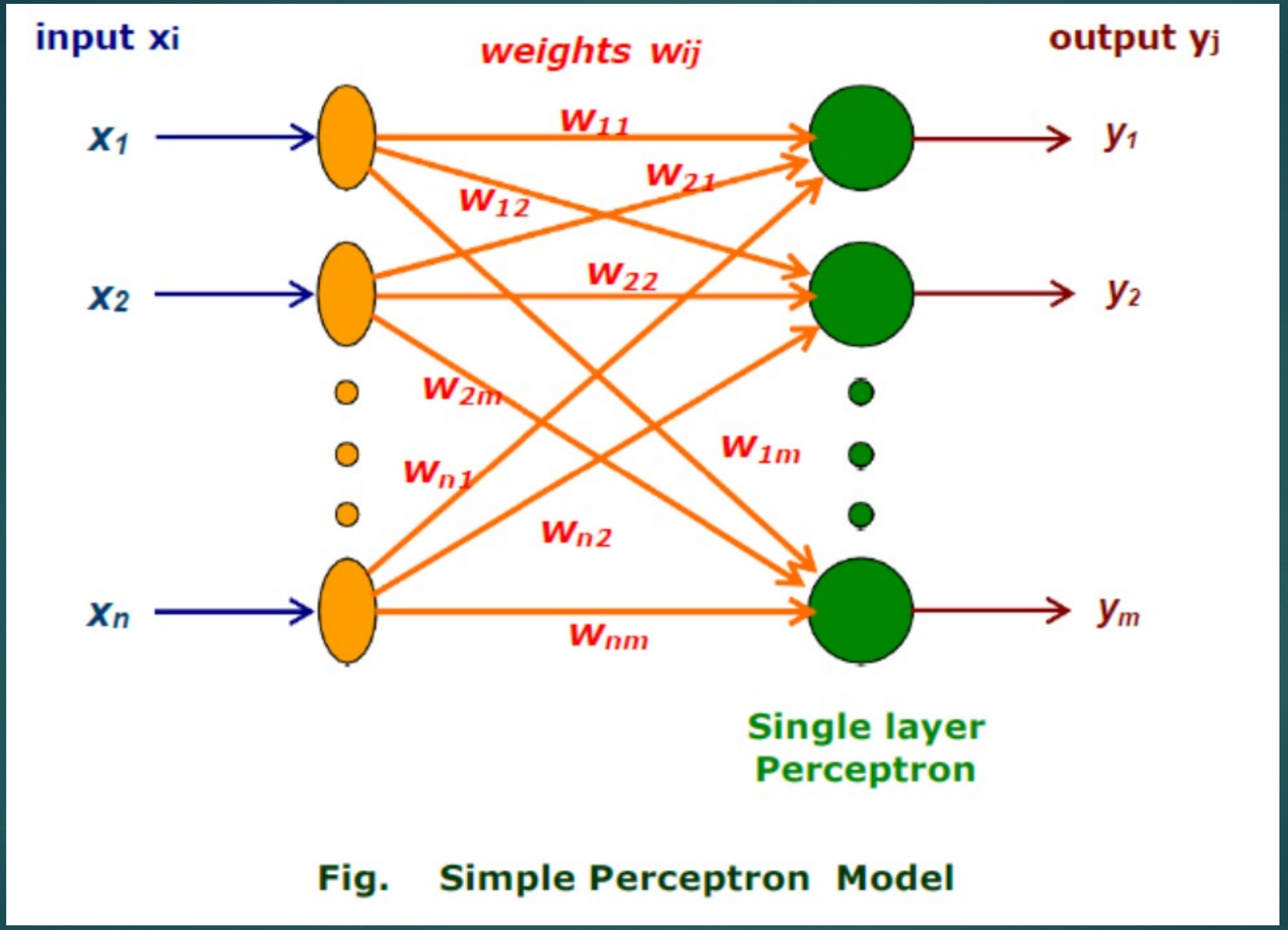


Fig. Simple Perceptron Model

Perceptron Learning Algorithm

The algorithm is illustrated step-by-step.

- **Step 1 :**

  Create a peceptron with **(n+1)** input neurons $x_0, x_1, \ldots\ldots, x_n$,

  where $x_0 = 1$ is the bias input.

  Let **O** be the output neuron.

- **Step 2 :**

  Initialize weight $W = (w_0, w_1, \ldots\ldots, w_n)$ to random weights.

- **Step 3 :**

  Iterate through the input patterns X_j of the training set using the weight set; ie compute the weighted sum of inputs $net\ j = \sum_{i=1}^{n} x_i w_i$ for each input pattern **j** .

- **Step 4 :**

  Compute the output y_j using the step function

$$y_j = f(net_j) = \begin{cases} 1 & \text{if } net_j \geq 0 \\ 0 & \text{if } net_j < 0 \end{cases} \quad \text{where} \quad net_j = \sum_{i=1}^{n} x_i w_{ij}$$

- **Step 5 :**

  Compare the computed output y_j with the target output y_j for each input pattern j .

  If all the input patterns have been classified correctly, then output (read) the weights and exit.

- **Step 6 :**

  Otherwise, update the weights as given below :

  If the computed outputs y_j is **1** but should have been **0**,

  Then $w_i = w_i - \alpha x_i$, $i = 0, 1, 2, \ldots\ldots, n$

  If the computed outputs y_j is **0** but should have been **1**,

  Then $w_i = w_i + \alpha x_i$, $i = 0, 1, 2, \ldots\ldots, n$

  where α is the learning parameter and is constant.

- **Step 7 :**

  goto step 3

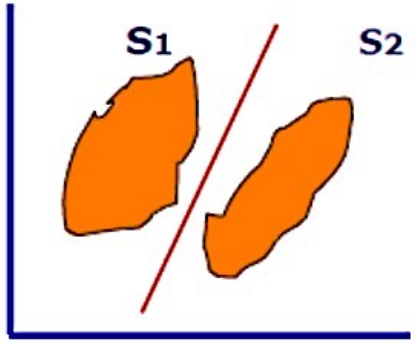
- **END**

Perceptron and Linearly Separable Task

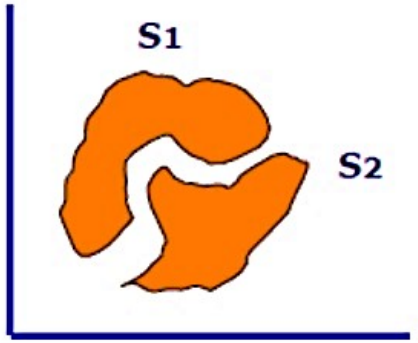
Perceptron can not handle tasks which are not separable.

- Definition : Sets of points in 2-D space are linearly separable if the sets can be separated by a straight line.
- Generalizing, a set of points in n-dimensional space are linearly separable if there is a hyper plane of (n-1) dimensions separates the sets.

Example



(a) Linearly separable patterns



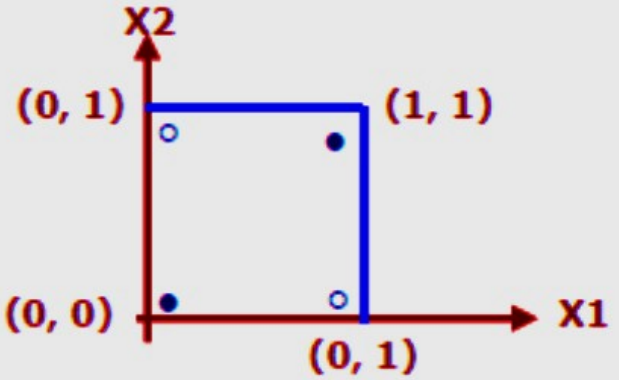
(b) Not Linearly separable patterns

Note : Perceptron cannot find weights for classification problems that are not linearly separable.

XOR Problem : Exclusive OR operation

Input x1	Input x2	Output	
0	0	0	Even parity •
1	1	0	
0	1	1	Odd parity °
1	0	1	

XOR truth table



Output of XOR in X1 , x2 plane

Even parity means even number of 1 bits in the input

Odd parity means odd number of 1 bits in the input

- There is no way to draw a single straight line so that the circles are on one side of the line and the dots on the other side.
- Perceptron is unable to find a line separating even parity input patterns from odd parity input patterns.

Neural Network Applications can be grouped in following categories:

Clustering:

A clustering algorithm explores the similarity between patterns and places similar patterns in a cluster. Best known applications include data compression and data mining.

Classification/Pattern recognition:

The task of pattern recognition is to assign an input pattern (like handwritten symbol) to one of many classes. This category includes algorithmic implementations such as associative memory.

Function approximation :

The tasks of function approximation is to find an estimate of the unknown function subject to noise. Various engineering and scientific disciplines require function approximation.

Prediction Systems:

The task is to forecast some future values of a time-sequenced data. Prediction has a significant impact on decision support systems. Prediction differs from function approximation by considering time factor. System may be dynamic and may produce different results for the same input data based on system state (time).

References:

http://www.myreaders.info/html/soft_computing.html

J-S R Jang and C-T Sun, Neuro-Fuzzy and Soft Computing, Prentice Hall, 1997

S. Rajasekaran and G.A. Vijayalaksmi Pai ,Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications, (2005), Prentice Hall.