



ΔΙΣΙΔ

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

PGS. TS. Nguyễn Thanh Bình

Saturday, 13:00 - 15:00

HCMC - 2023

PGS. TS. Nguyễn Thanh Bình

Trưởng Bộ môn Ứng dụng Tin học, Khoa Toán - Tin học

Email: ngtbinh@hcmus.edu.vn

Webpage: <https://sites.google.com/site/ntbinhpolytechnique/>



- Over 10 years of experience in AI, Machine Learning, Data Engineering.
- Had over 90 publications in prestigious conferences/ journals and four patents filed in USA and Canada.
- Won various international and national scientific awards in Machine Learning and Artificial Intelligence.
- Help over 20 students win Ph.D. fellowships in well-known universities and research institutes in Europe, Asia, and America during the last five years.

Objective

- Understand basic concepts of Artificial Intelligence:
 - Heuristic Algorithms
 - Fuzzy Logic and Genetic Algorithms.
 - Neural Network and its applications.
 - Knowledge Representation
 - Data-Mining and Machine Learning
- For all undergraduates on Computer Science

References

- ★ Machine Learning, Nguyen Dinh Thuc, 2002.
- ★ Genetic Algorithm, Nguyen Dinh Thuc, 2001.
- ★ Artificial Intelligence: A modern approach, Russel and Norvig, 2003.
- ★ An introduction to Genetic Algorithms, Melanie Mitchell, MIT Press, 1999

Assessment Overview

- ★ **Homework:** 20%
- ★ **Programming assignments:** 20%
- ★ **Final Exam:** 60%
- ★ **Bonus:** 10%
- ★ **Group môn học:**

<https://www.facebook.com/groups/283107291178216>

Classroom Rules

- ★ Each homework and programming assignments should be done within 2 weeks.
- ★ No excuse for late cases.
- ★ You will be marked for the parts in which you claimed for your own contribution

What is Artificial Intelligence ?

- Course Overview
- What is AI?
- The History of AI
- What can AI do?

Course overview

- Introductions and agents
- Heuristic Algorithms and its applications
- Knowledge Representation
- Machine Learning and Data-Mining.
- Genetic Algorithms
- Fuzzy Logics and Neural Networks.

What is AI?

Artificial Intelligence in Movies

In movies, robots are able to talk, think, have emotions, and make decisions just like humans.



What do you think Artificial Intelligence is?



NETFLIX
 TikTok



Grammarly will inspect your writing carefully to find ways to improve clarity, word choice, and more.

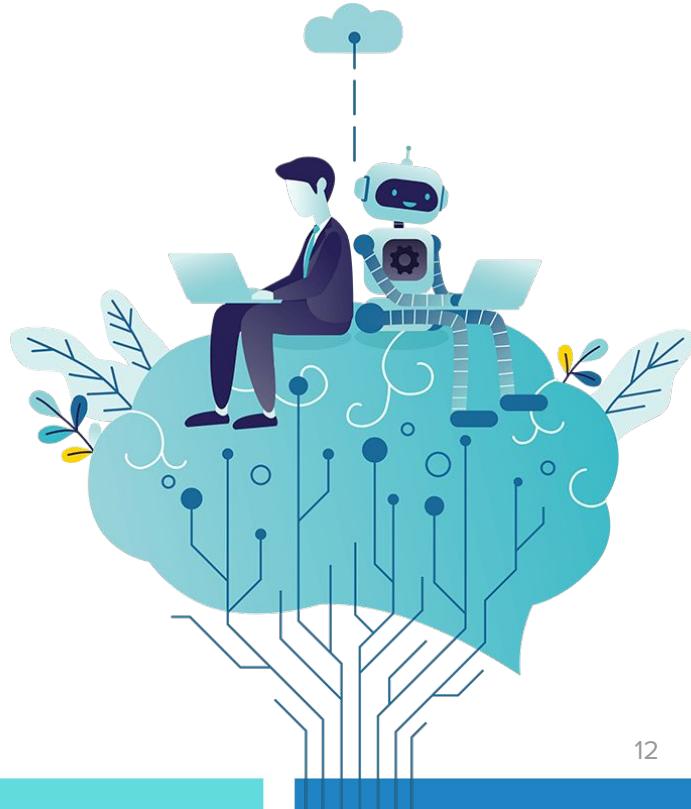
 grammarly

Enhance word choice
inspect-your-writing-carefully → scrutinize your writing



What is AI?

A.I is the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making and translation between languages....
(Oxford dictionary)



What is AI?

The power of a machine to copy and learn from intelligent human behavior.

BIG DATA

Capable of processing massive amounts of **structured and unstructured data** which can change constantly



REASONING

Ability to reason (deductive or inductive) and to draw inferences based on situation. **Context driven awareness** of system.

Ability to **learn** based on historical patterns, expert input and feed-back loop

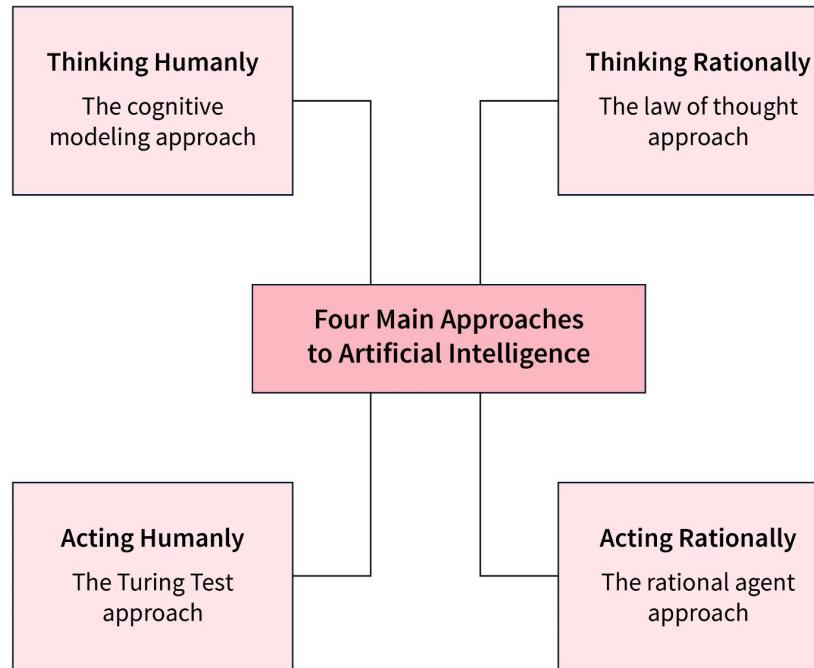
LEARNING

Capable of analyzing and **solving complex problems** in special-purpose and general-purpose domain

PROBLEM SOLVING

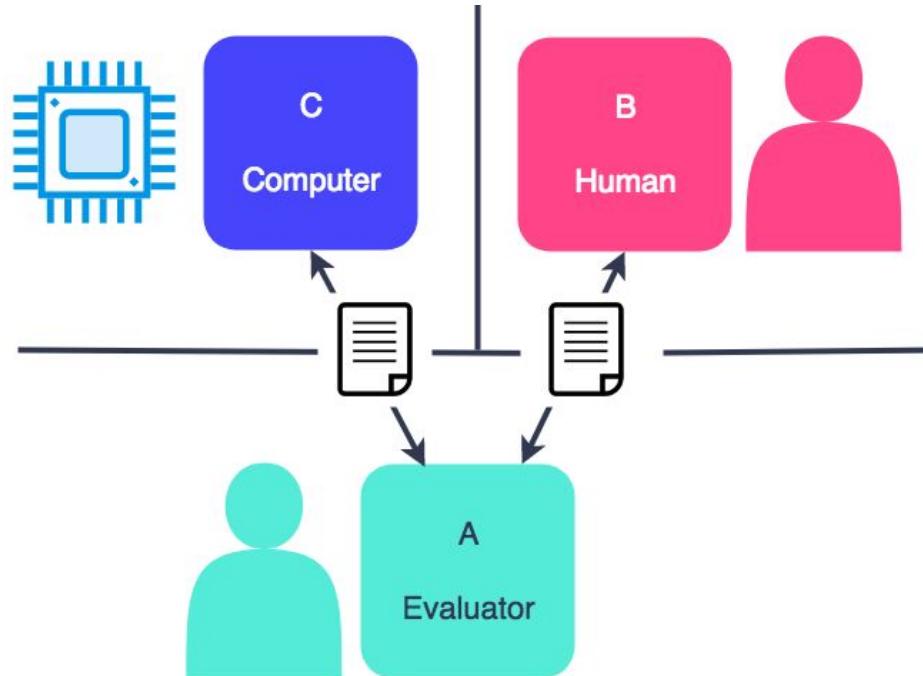
What is AI?

A.I can be separated into four categories as following:



Acting humanly: Turing Test

Was proposed by A. Turing in 1950.



Acting humanly: Turing Test

The computer would need to possess the following capabilities:

- natural language processing: to communicate successfully in English
- knowledge representation: to store what it knows or hears
- automated reasoning: to use the stored information to answer questions and to draw new conclusions
- machine learning: to adapt to new circumstances and to detect and extrapolate patterns.

Acting humanly: Turing Test

To pass the total Turing test, the computer will need:

- computer vision: to perceive objects.
- robotics: manipulate objects and move about.

Prepared for all major components of AT in following 60 years.

Thinking humanly: cognitive modeling

Requires scientific theories of internal activities of the brain:

- Predict and test behavior of human subjects (Cognitive Science)
- Direct identification from neurological data (Cognitive NeuroScience)

Thinking rationally: “law of thought”

- Aristotle attempted to codify “right thinking:
 - Use “laws of thought” to yield correct conclusions by logic.
 - For example: “Socrates is a man; all men are mortal; therefore, Socrates is mortal”.
- Direct line through mathematics and philosophy to AI.
- Obstacles:
 - Not all intelligent behavior can be described by logical notations.
 - A big difference between being able to solve a problem “in principle” and doing so in practice.

Acting rationally: rational agent

- A rational agent is one that can act so as to achieve the best results even when there is uncertainty.
- Rational behavior: doing the right thing.
- Doesn't necessarily involve thinking but thinking should be in the service of rational action.

Rational Agents

- An agent is something that can perceive and act.
- For a given class of environments and tasks, we look for the agent (or class of agents) which obtains the best performance.
- Computational limitations make perfect rationality unachievable. Therefore, one needs to design best program for given machine resources.

History of Artificial Intelligence

History of Artificial Intelligence

Turing Machine



1936



The history begins: The term “AI” is coined

Birth of the first chatbot ‘ELIZA’



1956

AI enters the Medical Field

“NETtalk” program speaks



1972



1986



‘Deep Blue’ from IBM beats World Chess Champion

AI enters everyday life
‘Siri’, ‘Cortana’, ‘Alexa’



2011



AI debates Space travel

The near future is intelligent



20XX

Source: Bosch

What can AI do?

What AI Can and Cannot Do

What AI Can Do

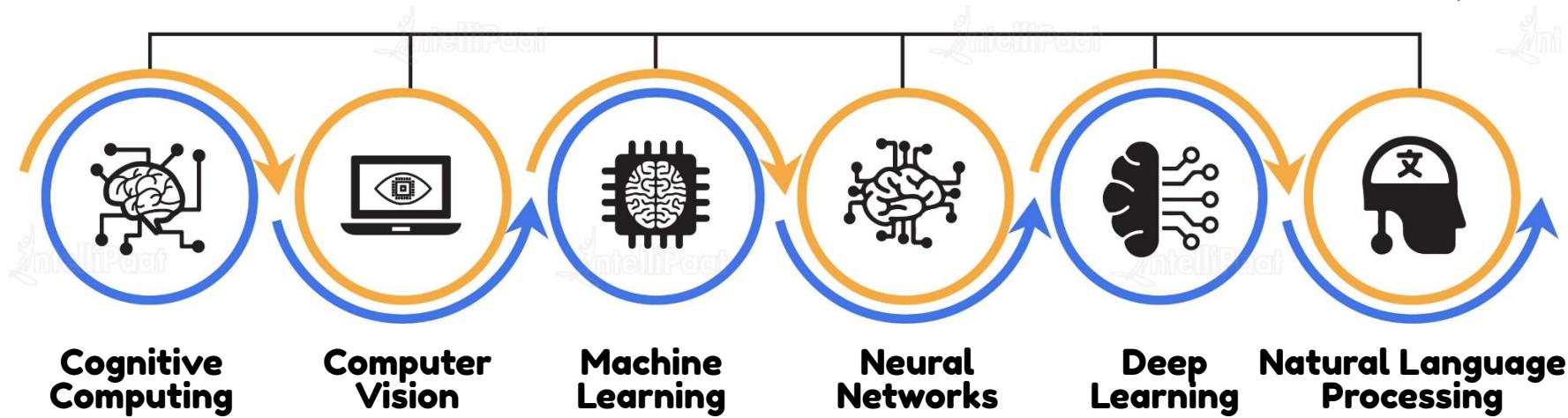
- Generate Quick Results
- Scan Large Databases and Search Facts in Few Seconds
- Find Mathematical and Logical Solutions with Fewer Errors
- Make Decisions based on Solely Objective Criteria

What AI Cannot Do

- Generate Results in Novel Situations
- Develop Own Unique Insights
- Think Intuitively and Abstractly
- Engage in Social Interactions and Communication
- Have Emotional Intelligence

Major subfields of Artificial Intelligence

Source: intellipaat



**Cognitive
Computing**

**Computer
Vision**

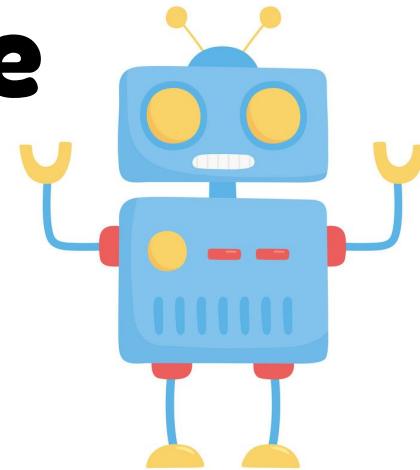
**Machine
Learning**

**Neural
Networks**

**Deep
Learning**

**Natural Language
Processing**

Types of Artificial Intelligence



Narrow AI (Weak AI)

Thực hiện một số công việc nhất định, không có khả năng bắt chước, tái tạo trí thông minh con người,

Trợ lý ảo, Chatbot, Xe tự lái, robots lau nhà

General AI (Strong AI)

Có trí thông minh chung bắt chước trí thông minh, hành vi của con người, có khả năng học hỏi và áp dụng trí thông minh để giải quyết các vấn đề.

Super AI

Giả định không chỉ bắt chước, hiểu được trí thông minh và hành vi của con người; tự nhận thức, vượt qua khả năng trí tuệ - khả năng của con người.

Predictive AI

Đưa ra dự đoán (lọc thư rác, chatbot, dự đoán hành vi của khách hàng...), nhằm tự động hóa chính xác các hoạt động, giảm sự can thiệp của con người.

Generative AI

Tạo nội dung mới: tạo ra văn bản, hình ảnh, âm thanh mới có hình thức tương đương với những gì con người có thể viết hoặc tạo.

Explainable AI (XAI)

Cho phép con người hiểu, tin tưởng vào kết quả do thuật toán AI tạo ra; giúp mô tả tính chính xác, công bằng, minh bạch và kết quả của mô hình trong quá trình ra quyết định do AI cung cấp.



ΔΙΣΙΔ

Q A

Intelligent Agents

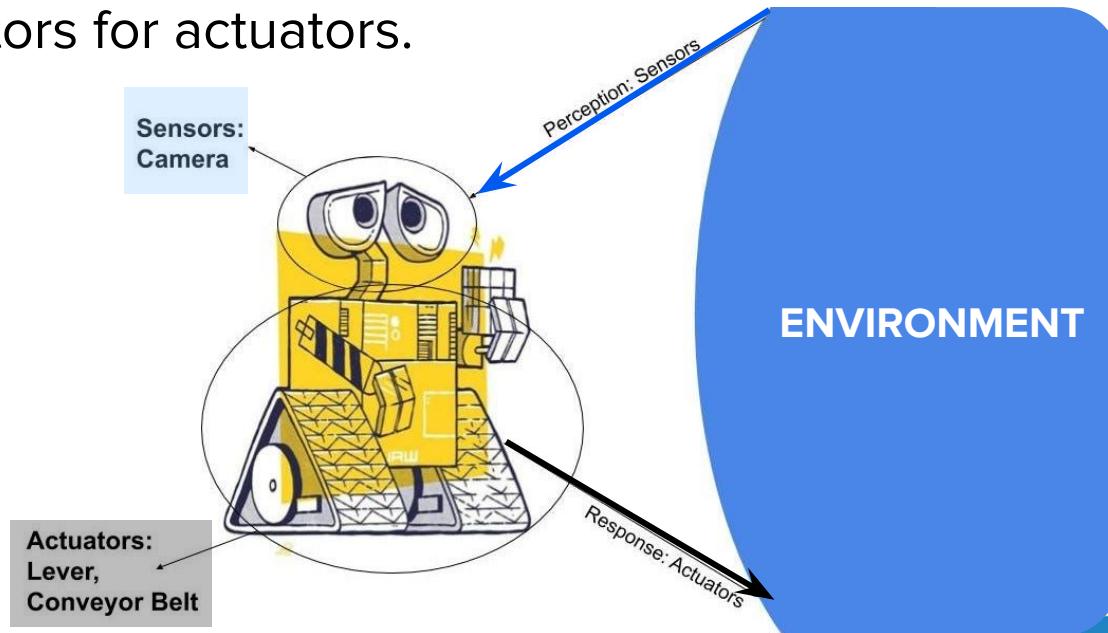
- Agents and environments
- Rationality
- PEAS model
- Environment types
- Agent types

Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acts upon that environment through actuators.
- ***Human agent:*** eyes, ears, skin and other organs for sensors; hands, legs, mouth and other body parts for actuators.

Agents

Robot agents: cameras and infrared range finders for sensors; different motors for actuators.



Agents

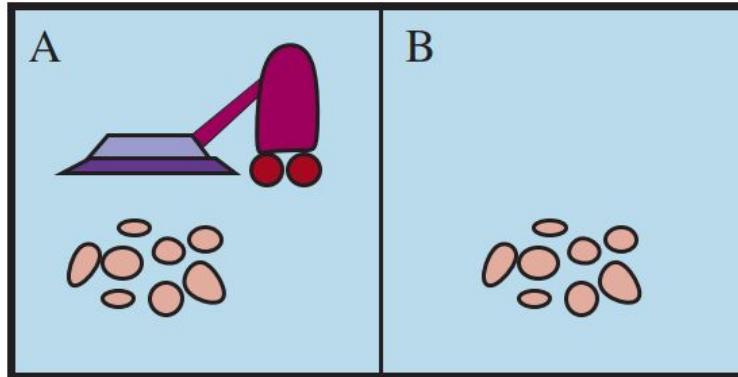
One can consider the agent function maps from its percept history to actions.

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

Agent = architecture + program

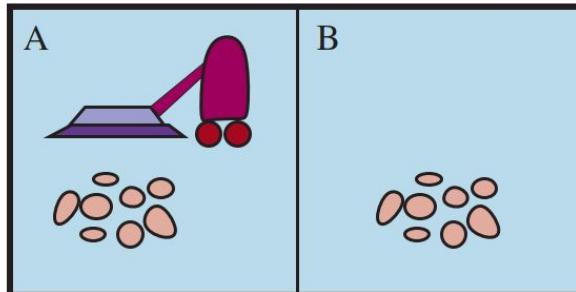
Vacuum-cleaner world

- The ***particular world*** has two locations: square A and B.
- The ***vacuum agent***: perceive which square it is in and whether there is dirt in the square.



Vacuum-cleaner world

- **The agent function:** if the current square is dirty, move right and suck up the dirt, otherwise move to another square.
- **Percepts:** locations and contents, e.g [A,Dirty].
- **Action:** left, right, suck, and NoOp.



Vacuum-cleaner world

Percept sequence	Action
$[A, Clean]$	<i>Right</i>
$[A, Dirty]$	<i>Suck</i>
$[B, Clean]$	<i>Left</i>
$[B, Dirty]$	<i>Suck</i>
$[A, Clean], [A, Clean]$	<i>Right</i>
$[A, Clean], [A, Dirty]$	<i>Suck</i>
:	:
$[A, Clean], [A, Clean], [A, Clean]$	<i>Right</i>
$[A, Clean], [A, Clean], [A, Dirty]$	<i>Suck</i>
:	:

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

Rational Agents

- A rational agent should make the great efforts to “*do the right thing*”, based on what it can perceive and the actions it can perform.
- The right action is the one that will cause the agent to be the most successful.
- **Performance measure:** An objective criterion for success of an agent’s behaviors.

Rational Agents

- For example: performance measure of a vacuum-cleaner can be proposed by [REDACTED], [REDACTED], [REDACTED], ..., [REDACTED]
- One should design performance measure according to what one actually wants in the environment, rather than according to how one thinks the agent should behave.

Rational Agents

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

Rational Agents

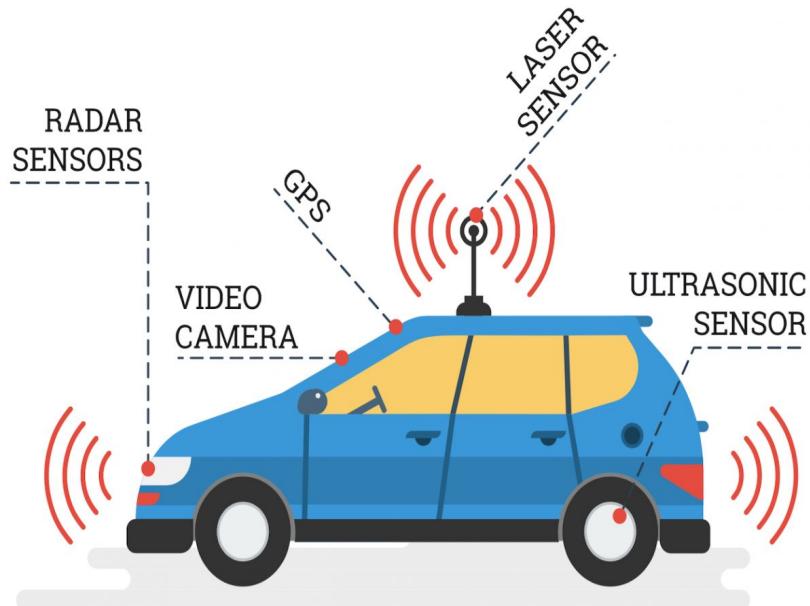
- Rationality is different from omniscience (all knowing with infinite knowledge).
- Agents can perform actions in order to modify future percepts so as to obtain useful information (information gathering, exploration).
- An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt).

PEAS

- **PEAS:** Performance Measure, Environment, Actuators, Sensors.
- Must first specify the setting for intelligent agent design.
- Example: *design an automated taxi-driver:*
 - Performance measure
 - Environment
 - Actuators
 - Sensors

PEAS

Design an automated taxi driver:



- **Performance measure:** Safe, fast, legal, comfortable trip, maximize profits.
- **Environment:** roads, other traffic, pedestrians, customers, police.
- **Actuators:** steering wheel, accelerator, brake, signal, horn.
- **Sensors:** cameras, sonar, speedometer, GPS, engine sensors, keyboards.

PEAS

Agent: Medical Diagnosis System:

- **Performance measure:** healthy patients, minimize costs, lawsuits.
- **Environment:** patients, hospitals and staffs.
- **Actuators:** Screen display (questions, tests, diagnoses, treatments, referrals).
- **Sensors:** keyboard (entry of symptoms, findings, patient's answers)



Source: GAO. | GAO-22-104629

PEAS

Agent: Part-picking robots.

- **Performance measure:** percentage of parts in correct bins.
- **Environments:** conveyor belt with parts and bins.
- **Actuators:** jointed arms and hands.
- **Sensors:** Camera, joint angle sensors.



PEAS



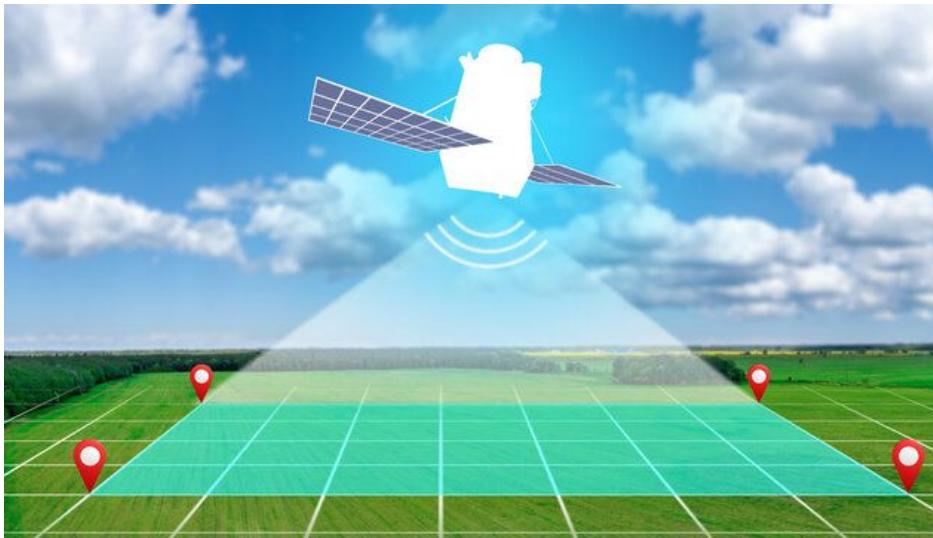
Agent: Interactive English tutor:

- **Performance measure:** maximized student's score on test.
- **Environment:** the set of students.
- **Actuators:** Screen display (exercises, suggestions, corrections).
- **Sensors:** keyboard

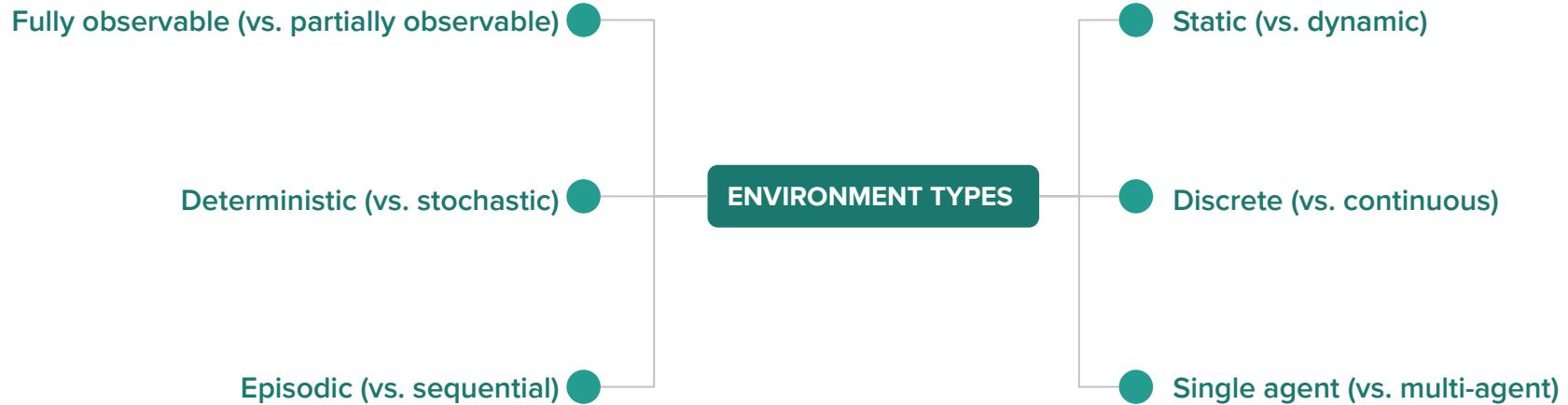
PEAS

Agent: Satellite image analysis system:

- **Performance measure:** correct image categorization.
- **Environment:** downlink from orbiting satellite.
- **Actuators:** display categorization of scene.
- **Sensors:** color pixel arrays



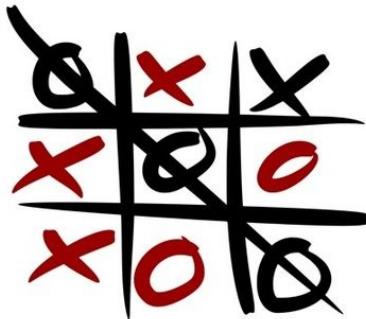
Environment types



Environment types

- **Fully observable** (vs. **partially observable**): an agent's sensors give it access to the complete state of the environment at each point in time.

Fully Observable Task Environment
Tic Tac Toe/Noughts & Crosses/X's and O's



Copyright 2019 @ www.perchingtree.com

Partially Observable Task Environment
Cards Game /Hidden Hand Poker



Copyright 2019 @ www.perchingtree.com

Environment types

- **Deterministic** (vs. **stochastic**): the next state of the environment is completely determined by the current state and the action executed by the agent. (*If the environment is deterministic except for the actions of other agents, the environment is strategic*). The stochastic environment is random in nature which is not unique and cannot be completely determined by the agent.
- Examples:
 - **Chess** : there would be only a few possible moves for a coin at the current state and these moves can be determined.
 - **Self-Driving Cars**: the actions of a self-driving car are not unique, it varies time to time.

Environment types

- **Episodic (vs. sequential):**

In an episodic task environment, the agent's experience is divided into atomic episodes. Each episode consists of the agent perceiving and then performing a single action. The choice of action in each episode depends only the episode itself.

In sequential environment, the ***current decision could affect all future decisions***, such as *chess* and *taxis driving*.

Environment types

- **Static** (vs. **dynamic**): the environment is unchanged while the agent is deliberating.
 - Taxi driving is dynamic: the other cars and taxi itself keep moving while the driving algorithm decides what to do next.
 - Crossword puzzles are static.

The environment is semi-dynamic if the environment itself does not change with the passage of time by the agent's performance score does. **For example:** chess when played with a clock is semi-dynamic.

Environment types

- **Discrete** (vs. **continuous**): A limited number of distinct, clearly defined percepts and actions.

Example:

- Chess has a discrete set of percepts and actions.
- Taxing driving is a continuous state and continuous-time problem.

Environment types

- **Single agent** (vs. **multi-agent**): an agent operating by itself in an environment.

Example: an agent solving a **crossword puzzle** by itself is clearly in a single-agent environment, whereas an agent **playing chess** is a two-agent environment.

Agent functions and programs

- An agent is **completely specified by** the **agent function mapping percept sequences to actions.**
- The job of AI is to design the agent program that implements the agent function mapping percept to actions.
- One agent function is rational.
- Aim: find a way to implement the rational agent function concisely.

Agent programs

- Take the current percept as input from the sensors (since nothing more is available from the environment) and return an action to the actuators.
- Different from the agent function that takes the entire percept history.
- If the agent's actions depend on the entire percept sequence, the agent has to remember the percepts.

Table-Driven Agent

- To keep track of the percept sequence and then use it to index into a table of actions to decide what to do next.
- To build a rational agent in this way, one must construct a table that contains the appropriate action for every possible percept sequence

```
function TABLE-DRIVEN-AGENT (percept) returns action
    static: percepts, a sequence, initially empty
            table, a table, indexed by percept sequences, initially fully specified

            append percept to the end of percepts
            action  $\leftarrow$  LOOKUP(percepts, table)
            return action
```

Table-Driven Agent

Cons:

- Huge table
- Take a long time to build the table
- No autonomy
- Need a long time to learn the table entries

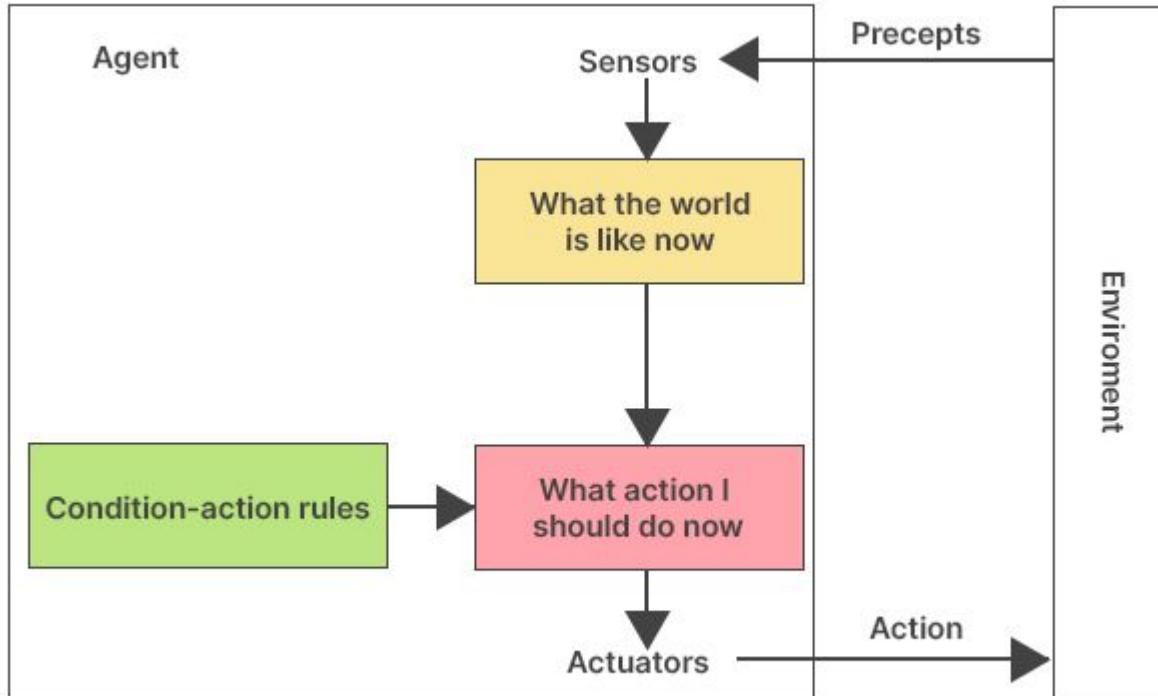
Vacuum-cleaner agent

```
Function Reflex-Vacuum-Agent ([Location, status]) returns an action
  If status = Dirty then return Suck
  Else if location = A then return Right
  Else if location = B then return Left
```

Four basic kinds of agent program:

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

Simple reflex agent



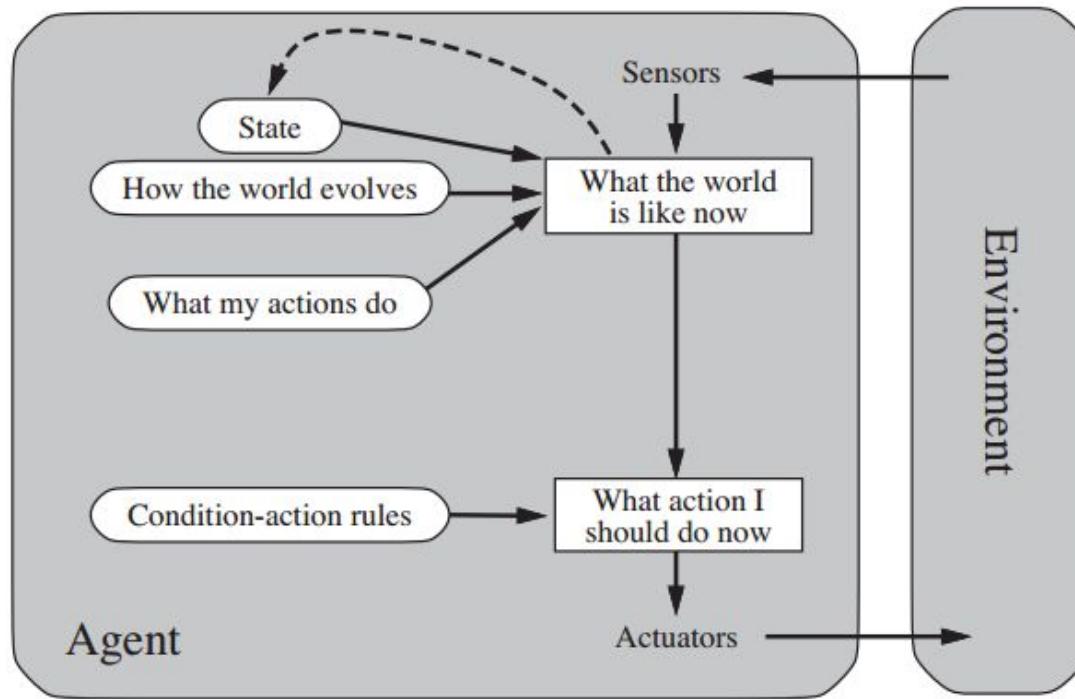
Simple reflex agent

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

    state  $\leftarrow$  INTERPRET-INPUT(percept)
    rule  $\leftarrow$  RULE-MATCH(state, rules)
    action  $\leftarrow$  RULE-ACTION[rule]
    return action
```

Figure 2.8 A simple reflex agent. It works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule.

Model-based reflex agents



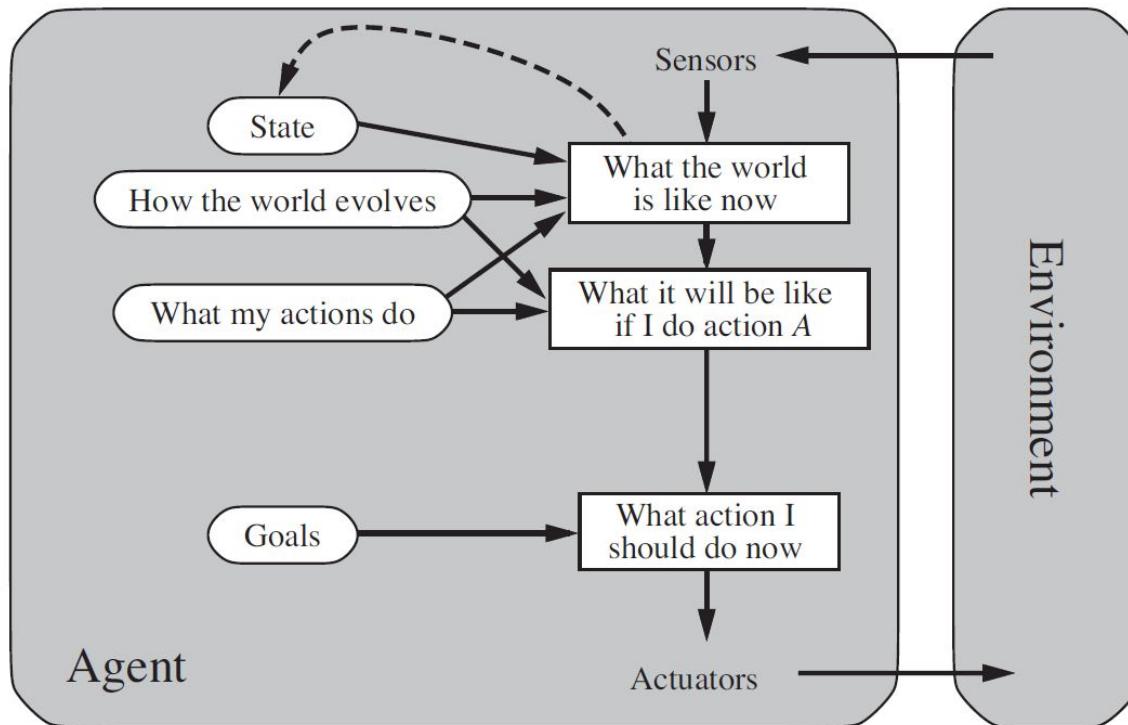
Model-based reflex agents

```
function REFLEX-AGENT-WITH-STATE(percept) returns action
    static: state, a description of the current world state
          rules, a set of condition-action rules

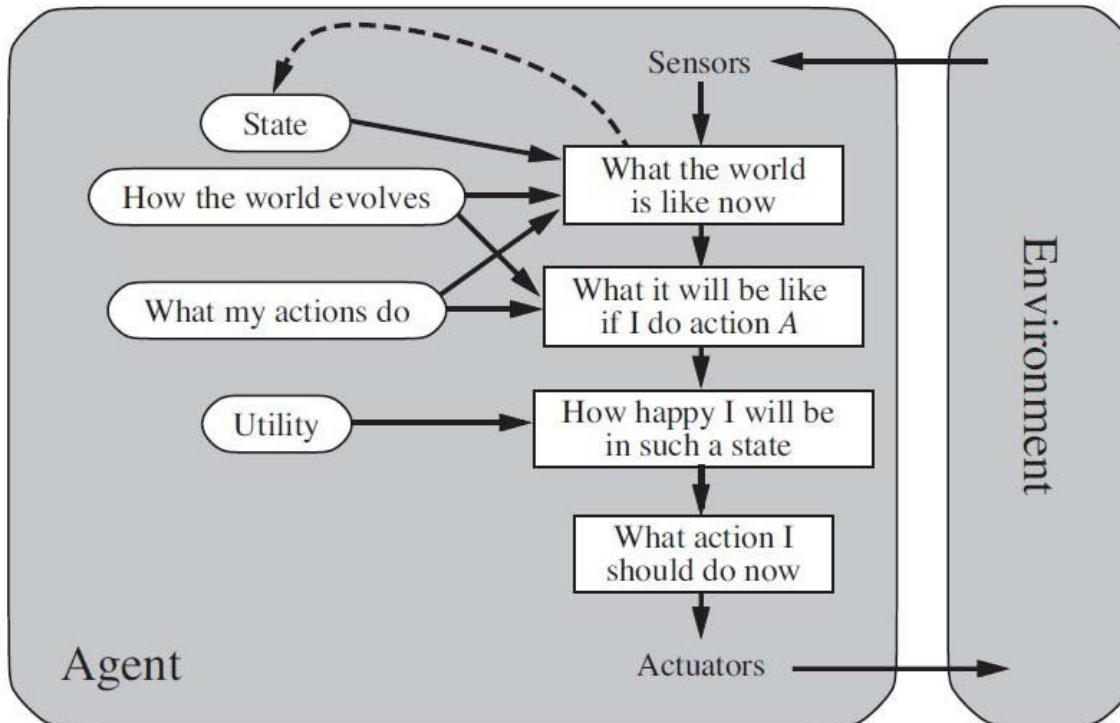
    state  $\leftarrow$  UPDATE-STATE(state, percept)
    rule  $\leftarrow$  RULE-MATCH(state, rules)
    action  $\leftarrow$  RULE-ACTION[rule]
    state  $\leftarrow$  UPDATE-STATE(state, action)
    return action
```

Figure 2.10 A reflex agent with internal state. It works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state) and then doing the action associated with that rule.

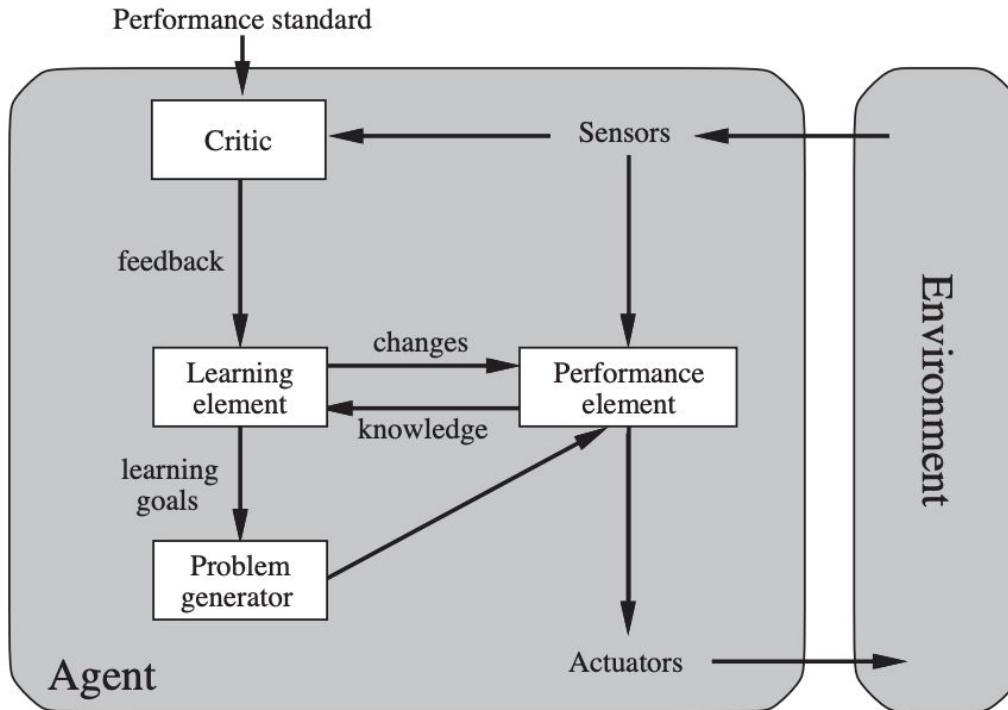
Goal-based agent



Utility-based agents



Learning agents





ΔΙΣΙΔ

Q A

Solving problems by searching

- Problem-solving agents
- Problem types
- Problem formulation
- Examples
- Basic search algorithms

Problem-solving agents

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

    state  $\leftarrow$  UPDATE-STATE(state, percept)
    if seq is empty then do
        goal  $\leftarrow$  FORMULATE-GOAL(state)
        problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
        seq  $\leftarrow$  SEARCH(problem)
    action  $\leftarrow$  FIRST(seq)
    seq  $\leftarrow$  REST(seq)
    return action
```

Problem-solving agents - Example

- On holiday in Vietnam; at the moment in Nha Trang.
- Flight leaves tomorrow in Hochiminh city.
- Formulate goal: be in Hochiminh city.
- Formulate problem:
 - States: Various provinces.
 - Actions: drive between provinces.
 - Find solutions: sequence of provinces.

Problem types

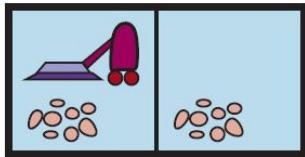
- **Deterministic, fully observable:** single-state problem. Agents know exactly which state it will be in; solution is a sequence.
- **No-observable:** sensorless problem (conformant problem). Agent may have no idea where it is and solution is a sequence.

Problem types

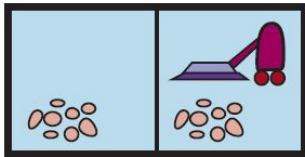
- **Nondeterministic and/or partially observable:** contingency problem: percepts provide new information about the current state often interleave search, execution
- **Unknown state space:** exploration problem

Example: vacuum world

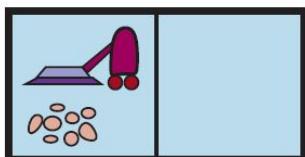
1



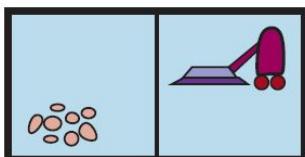
2



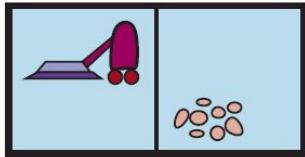
3



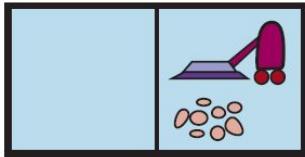
4



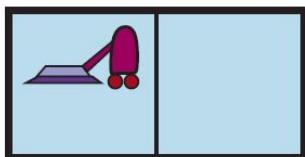
5



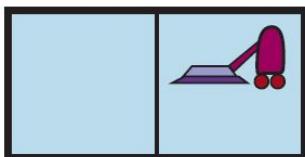
6



7



8

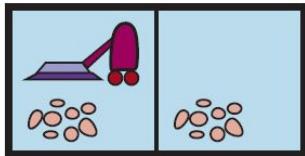


Single-state, start in state 5.

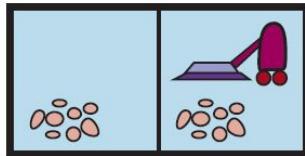
Solutions?

Example: vacuum world

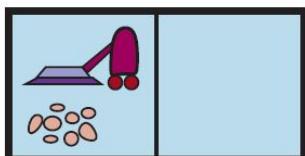
1



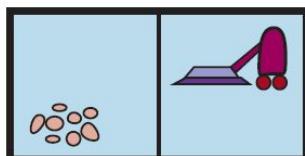
2



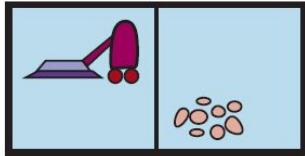
3



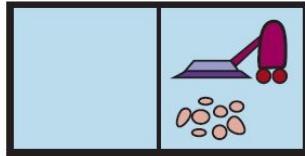
4



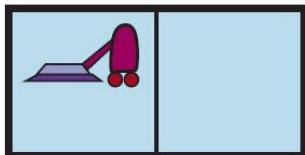
5



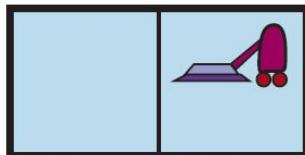
6



7



8



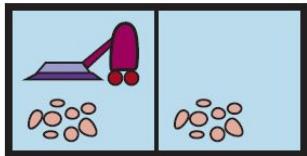
Single-state, start in state 5.

Solutions?

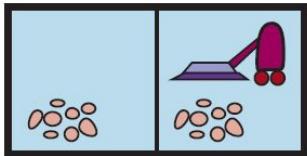
[Right,Suck]

Example: vacuum world

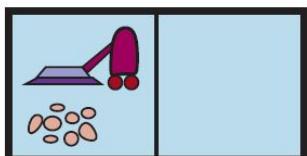
1



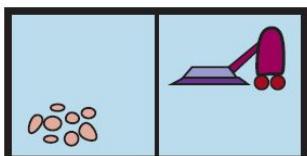
2



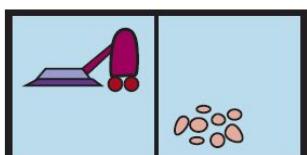
3



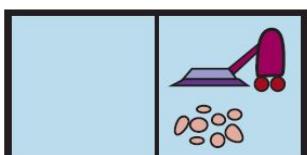
4



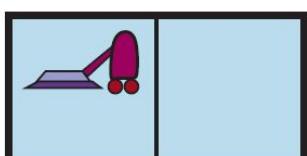
5



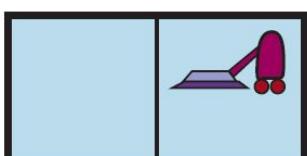
6



7



8



Sensorless:

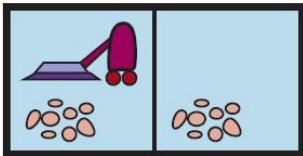
Start in $\{1, 2, \dots, 6, 7, 8\}$

Right goes to $\{2, 4, 6, 8\}$

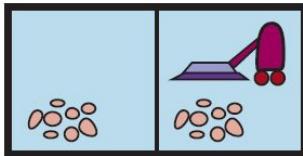
Solutions?

Example: vacuum world

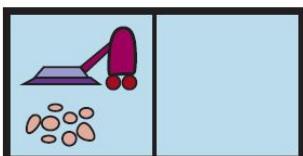
1



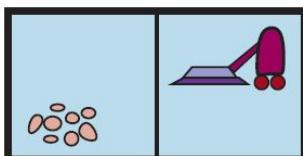
2



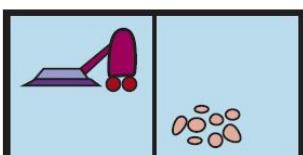
3



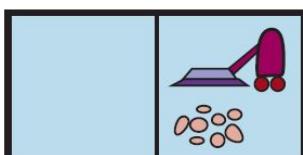
4



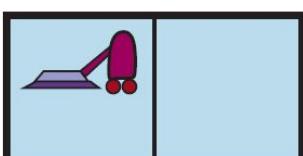
5



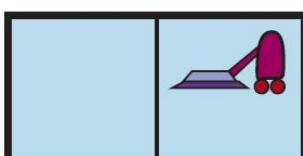
6



7



8



Contingency problems:

Nondeterministic: Suck may dirty a clean carpet.

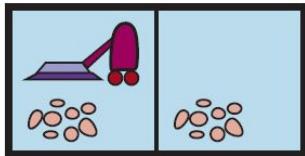
Partially observable: location, dirty at current location.

Percept: [L,Clean], i.e., start in the state 5 or 7.

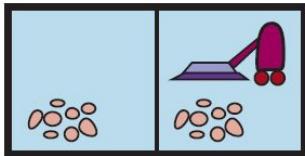
Solutions?

Example: vacuum world

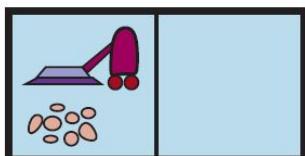
1



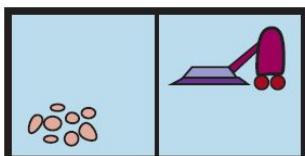
2



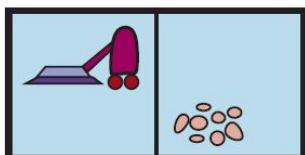
3



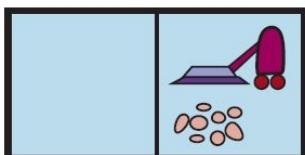
4



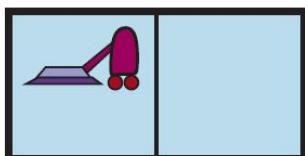
5



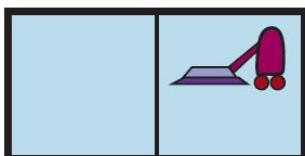
6



7



8



Contingency problems:

Nondeterministic: Suck may dirty a clean carpet.

Partially observable: location, dirty at current location.

Percept: [L,Clean], i.e., start in the state 5 or 7.

Solutions?

[Right, if Dirty then Suck]

Single-state problem formulation

- One problem can be determined by the following items:
 - **Initial state:** where the agent starts in, e.g., “at Nha Trang”
 - **Actions or successor function** $S(x) = \text{set of action-state pairs.}$
 - **Goal test:** explicit/implicit to determine whether a given state is a goal state.
 - **Path cost:** to show a numeric cost to each path, e.g., sum of distances, number of actions executed,...
 - $C(x,a,y) = \text{step cost (non-negative)}$

Single-state problem formulation

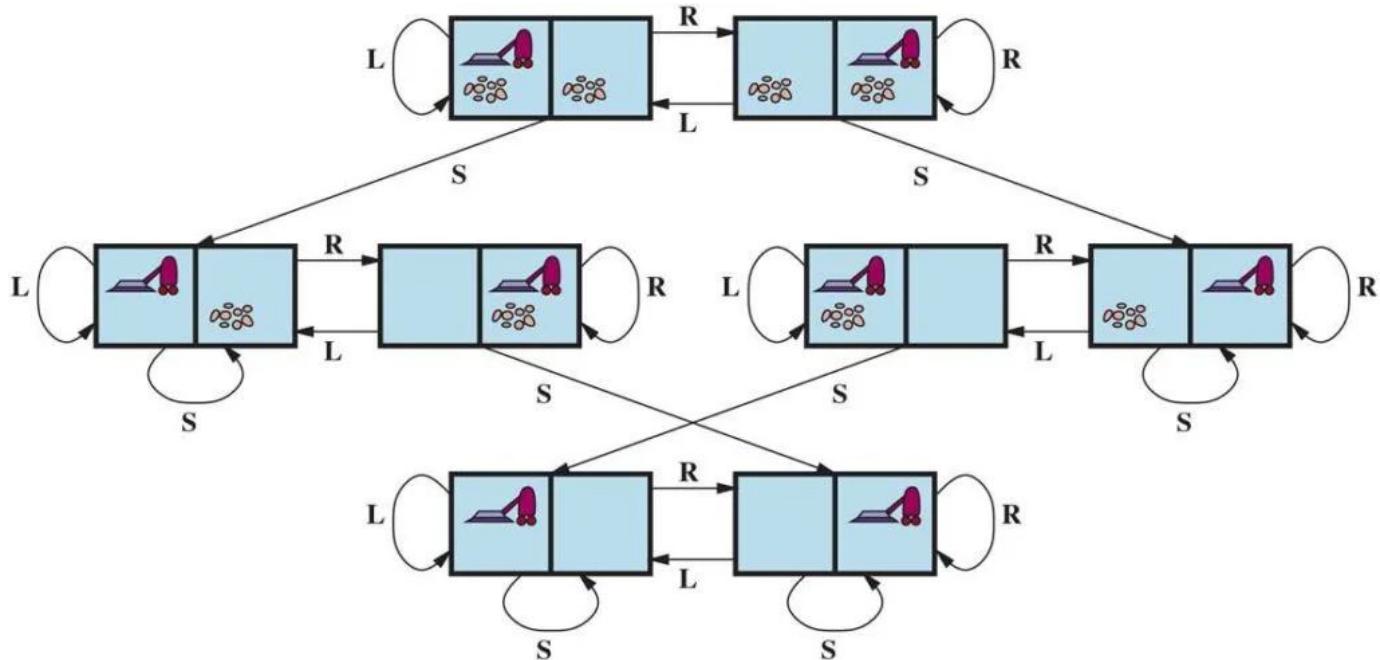
- A solution of the problem is a sequence of necessary actions from the initial state to achieve the goal state.
- Solution quality is measured by the path cost function and an optimal solution has the lowest path cost among all possible solutions.

Selecting a state space

- Real world is complicated. **State space** must be abstracted for problem solving.
- Abstract state = a set of real states.
- Abstract action = a complex combination of real actions.
- Abstract solution = a set of real paths that are solutions in the real world.

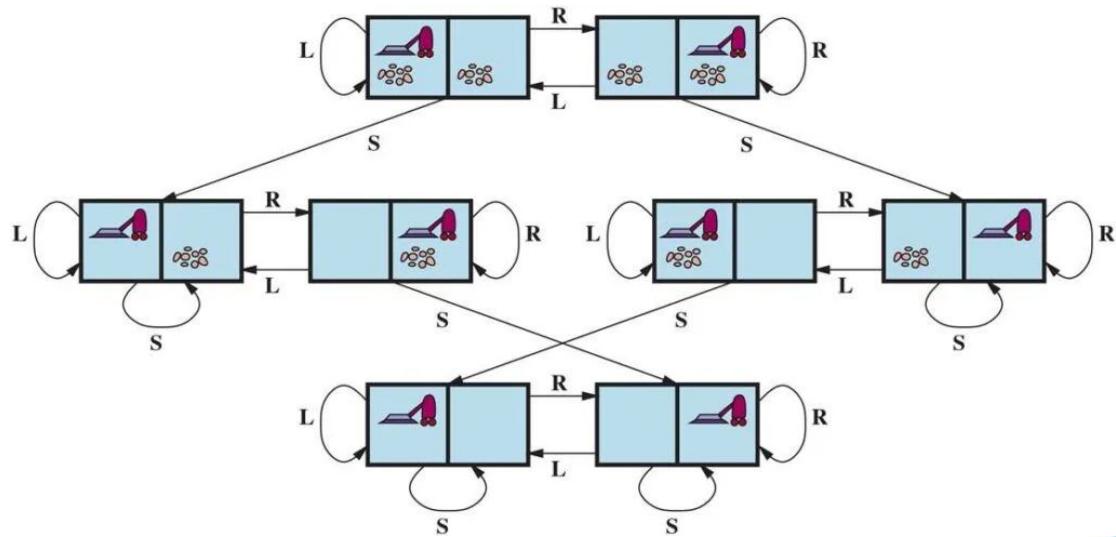
Vacuum world state space graph

- States?
- Actions?
- Goal test?
- Path cost?



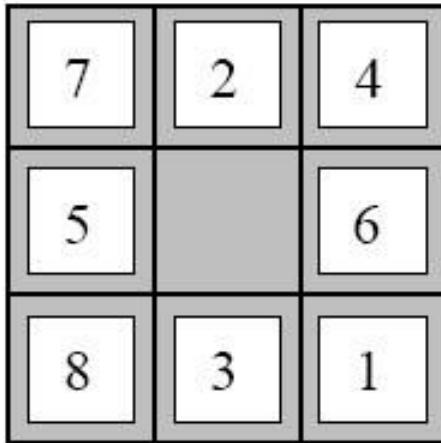
Vacuum world state space graph

- **States?**
 - integer dirt and robot location
- **Actions?** Left, Right, Suck
- **Goal test?**
 - No dirt at all locations
- **Path cost?**
 - 1 per action

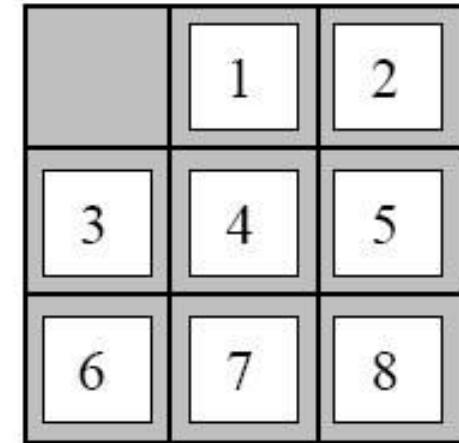


The 8-puzzle

- **States?**
- **Actions?**
- **Goal test?**
- **Path cost?**



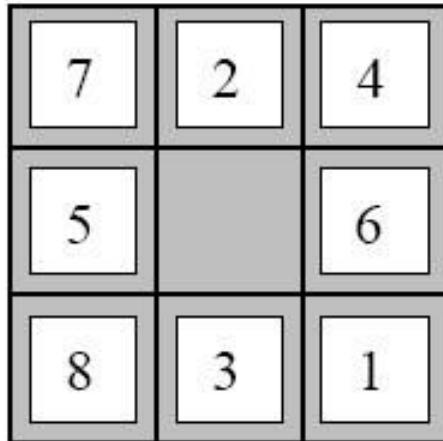
Start State



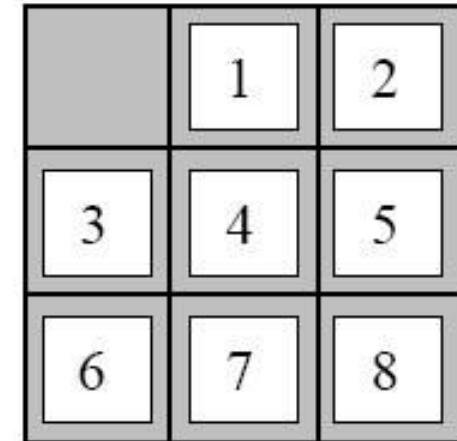
Goal State

The 8-puzzle

- **States?** locations of tiles
- **Actions?** move blank left, right, up, down
- **Goal test?** a given goal state
- **Path cost?** 1 per move
- Optimal solution of n-Puzzle family is NP-hard!!!



Start State



Goal State

Tree search algorithms

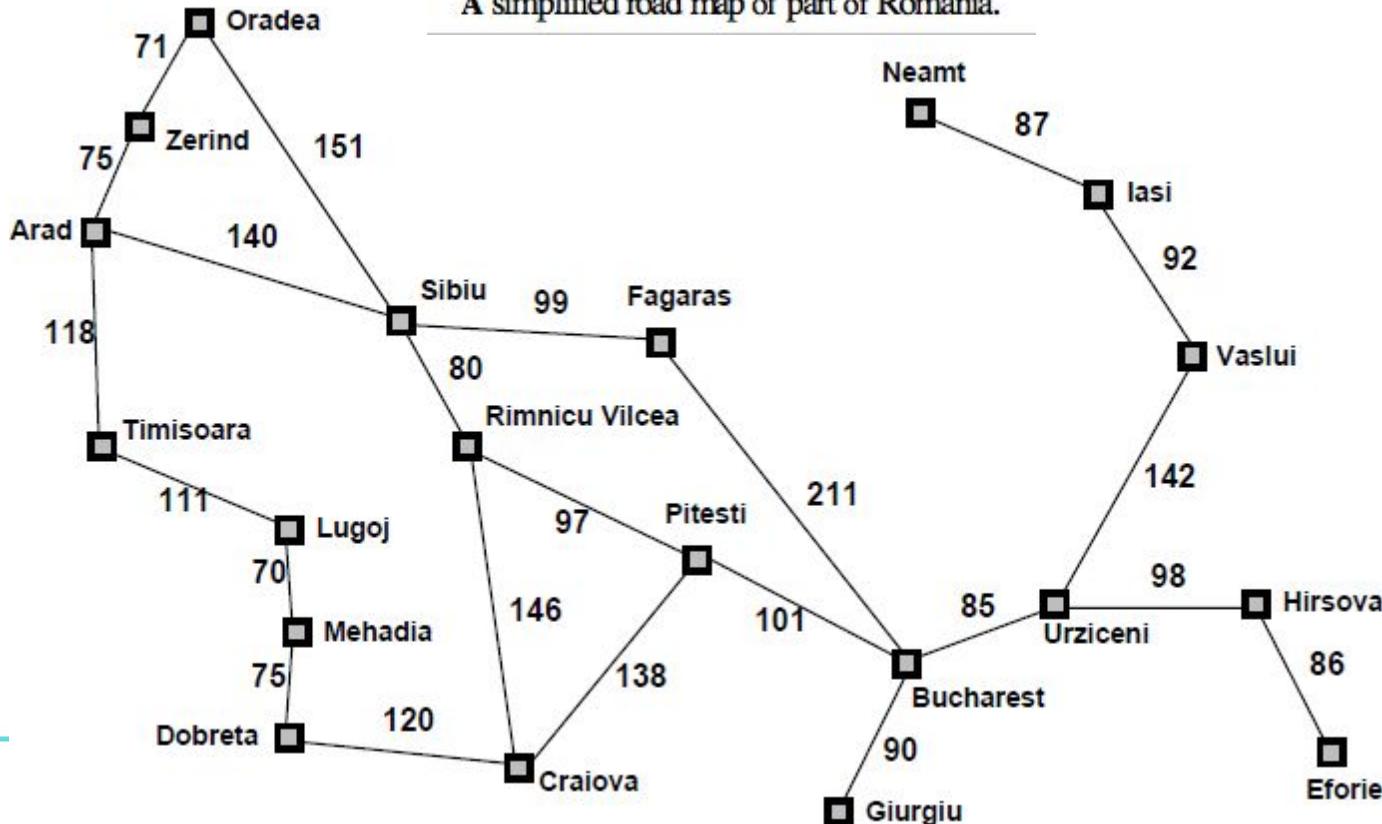
- After formulating the problem, we have to solve it. It can be done by a **search** through the **state space**.
- Basic idea:

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

Tree search algorithms

Example

A simplified road map of part of Romania.



Tree search algorithms

General tree-search algorithm

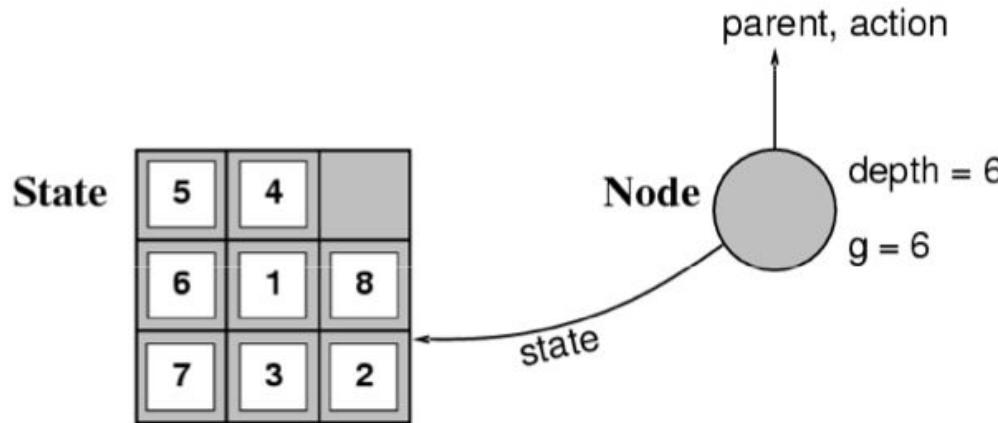
```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE(node)) then return node
    fringe  $\leftarrow$  INSERTALL(EXPAND(node, problem), fringe)

function EXPAND(node, problem) returns a set of nodes
  successors  $\leftarrow$  the empty set
  for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
    s  $\leftarrow$  a new NODE
    PARENT-NODE[s]  $\leftarrow$  node; ACTION[s]  $\leftarrow$  action; STATE[s]  $\leftarrow$  result
    PATH-COST[s]  $\leftarrow$  PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s]  $\leftarrow$  DEPTH[node] + 1
    add s to successors
  return successors
```

General tree-search algorithms

Implementation: States vs Nodes

- **A state** is a representation of a physical configuration
- **A node** is a data structure constituting part of a search tree, including: state, parent node, action, path-cost and depth (the number of steps along the path from the initial state).



General tree-search algorithms

Implementation: States vs Nodes

- The “**Expand**” functions creates new nodes, filling in the various fields and using the “SuccessorFn” of the problem to create the corresponding states.
- **A fringe** is a collection of nodes that have been generated but not expanded.

Search strategies

- A **search strategy** is defined by picking the **order of node expansion**.
- Strategies are **evaluated** along the following dimensions:
 - **completeness**: does it always find a solution if one exists?
 - **time complexity**: number of nodes generated
 - **space complexity**: How much memory is needed to perform the search
 - **optimality**: does it always find a least-cost solution?
- **Time and space complexity** are measured in terms of
 - b: maximum branching factor of the search tree
 - d: depth of the least-cost solution
 - m: maximum depth of the state space

Search algorithms

- Uninformed search algorithms
- Informed search algorithms

Uninformed search strategies

- Also called “blind search”.
- 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 any non-goal state.
- Different from the informed strategies or heuristic strategies.

Uninformed search strategies

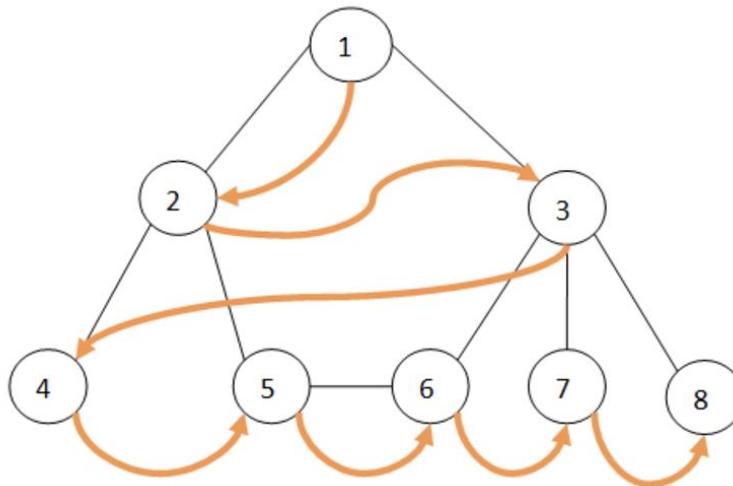
Several uninformed search strategies:

- Breadth-first search.
- Uniform-cost search.
- Depth-first search.
- Depth-limited search.
- Iterative deepening search

Breadth-first search

Expand **shallowest** unexpanded node.

Implementation: fringe is a **First-In-First-Out queue**, i.e., new successors go at end.



Complete?

Time?

Space?

Optimal?

→HOMEWORK!

Uniform-cost search

Expand **least-cost** unexpanded node.

Implementation: **fringe = queue ordered by path cost.**

Equivalent to breadth-first if step costs all equal.

Complete?

Time?

Space?

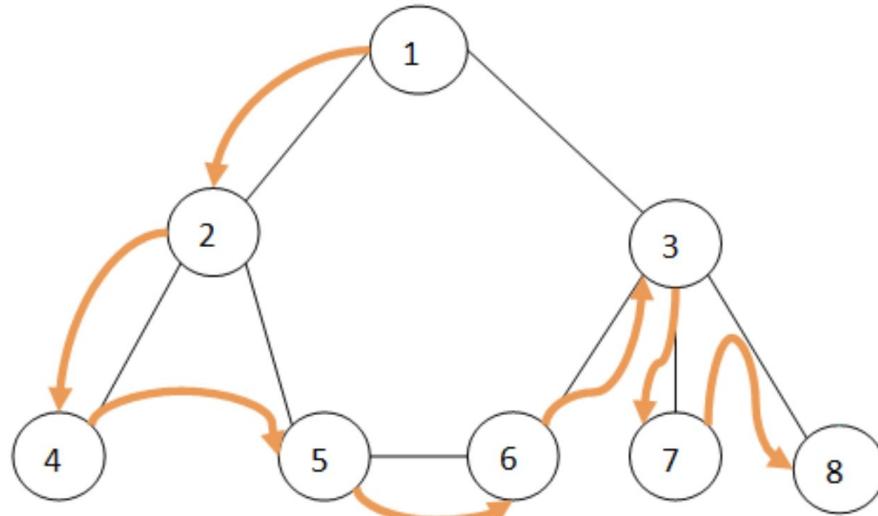
Optimal?

→HOMEWORK!

Depth-first search

Expand deepest unexpanded node.

Implementation: **fringe** = Last-In-First-Out, i.e, put successors at front



Complete?

Time?

Space?

Optimal?

→HOMEWORK!

Depth-limited search

A depth-first search with depth limit

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
    cutoff-occurred?  $\leftarrow$  false
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      result  $\leftarrow$  RECURSIVE-DLS(child, problem, limit - 1)
      if result = cutoff then cutoff-occurred?  $\leftarrow$  true
      else if result  $\neq$  failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

Sometimes used in deep-first search, that finds the best depth limit.

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

Figure 3.17 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.

Avoid repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!

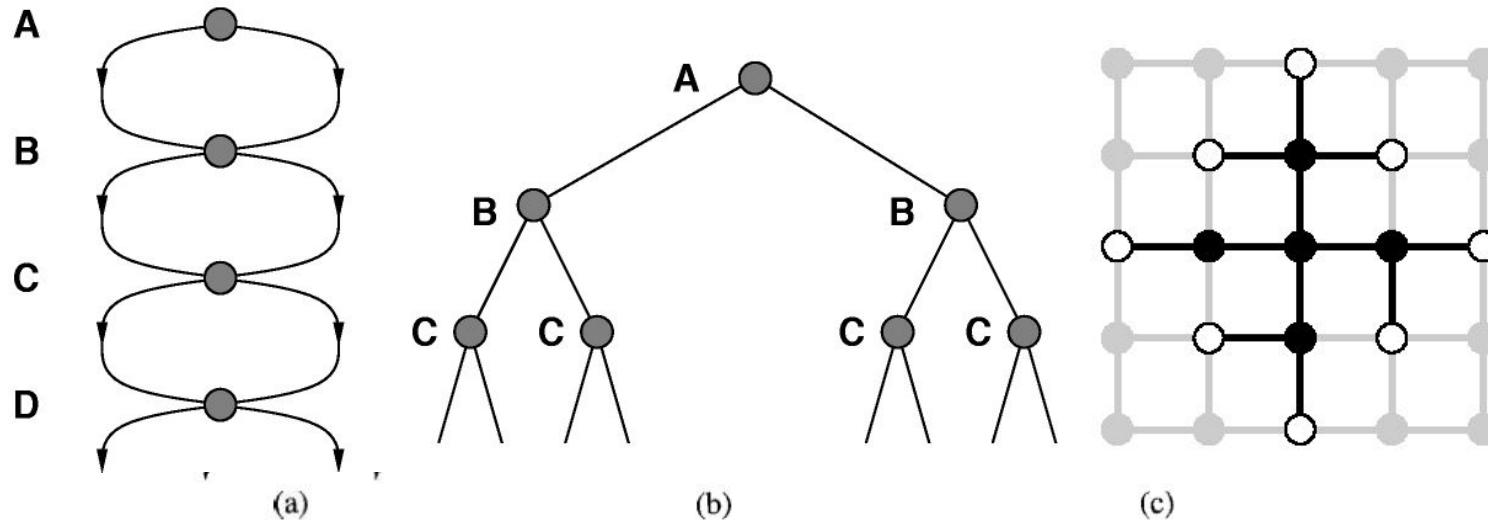


Figure 3.18 State spaces that generate an exponentially larger search tree. (a) A state space in which there are two possible actions leading from A to B, two from B to C, and so on. The state space contains $d + 1$ states, where d is the maximum depth. (b) The corresponding search tree, which has 2^d branches corresponding to the 2^d paths through the space. (c) A rectangular grid space. States within 2 steps of the initial state (A) are shown in gray.

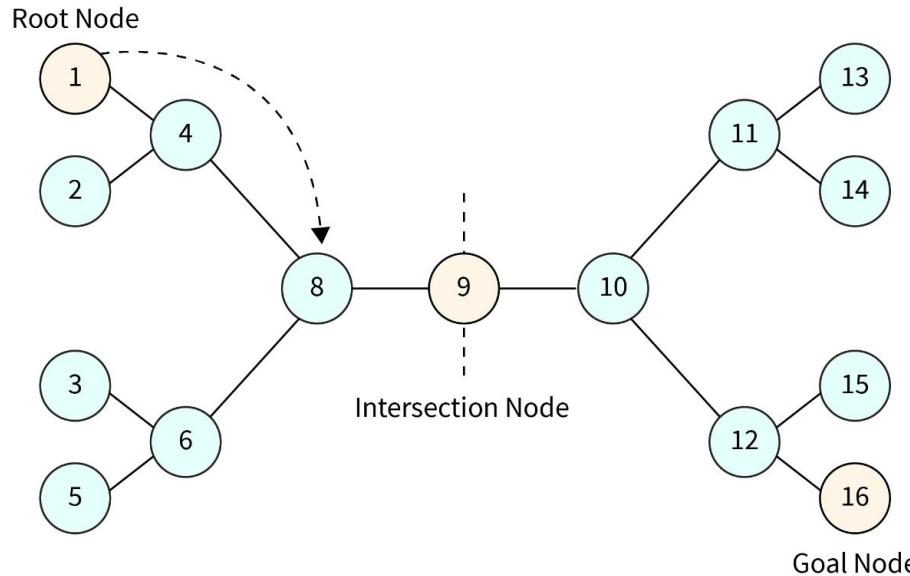
Avoid repeated states

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
    closed  $\leftarrow$  an empty set
    fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?( fringe) then return failure
        node  $\leftarrow$  REMOVE-FIRST( fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe  $\leftarrow$  INSERT-ALL(EXPAND(node, problem), fringe)
```

Figure 3.19 The general graph-search algorithm. The set *closed* can be implemented with a hash table to allow efficient checking for repeated states. This algorithm assumes that the first path to a state *s* is the cheapest (see text).

Bidirectional search

- Simultaneously search both forward (from the initial state) and backward (from the goal state)
- Stop when the two searches meet.





ΔΙΣΙΔ

Q A

Informed search algorithms

- Best-first search
 - Greedy best-first search
 - A* search
- Heuristics

Tree search

A search strategy can be determined by an order of node expansion:

```

function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE(node)) then return node
    fringe  $\leftarrow$  INSERTALL(EXPAND(node, problem), fringe)
  
```

```

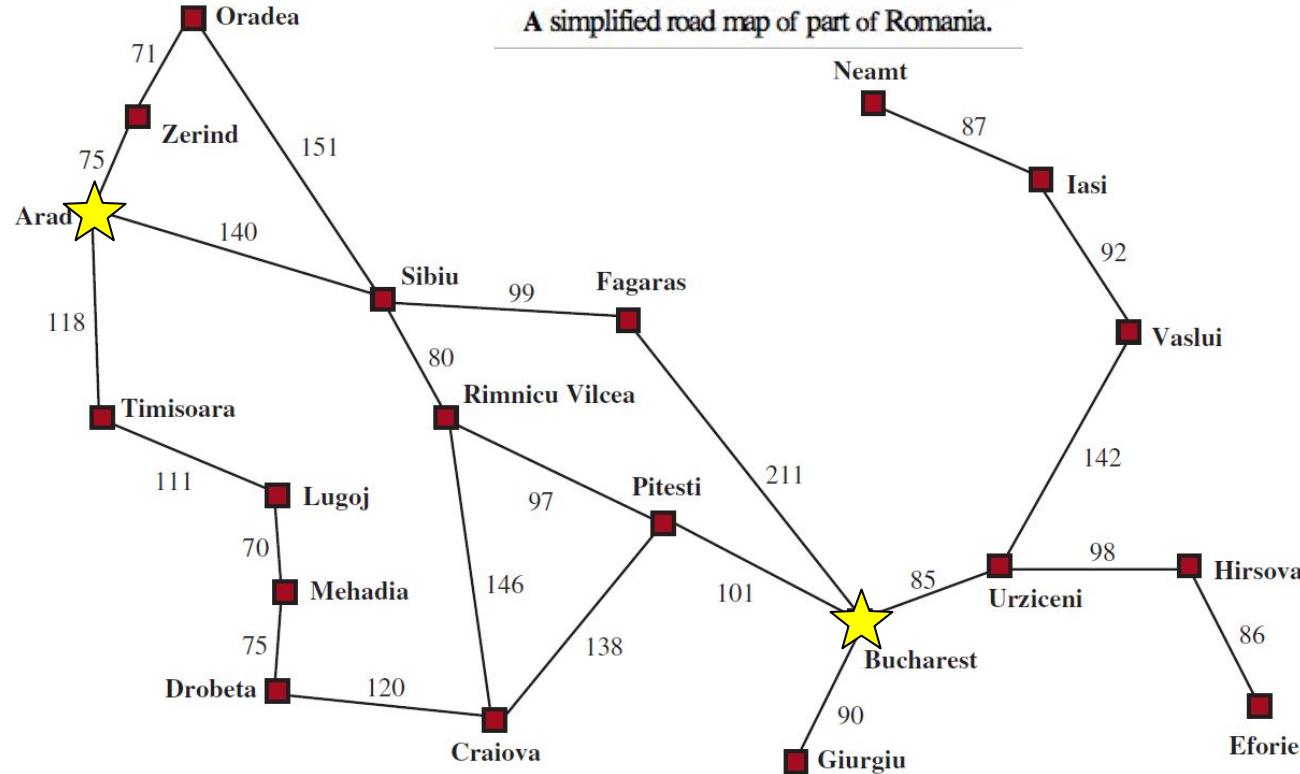
function EXPAND(node, problem) returns a set of nodes
  successors  $\leftarrow$  the empty set
  for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
    s  $\leftarrow$  a new NODE
    PARENT-NODE[s]  $\leftarrow$  node; ACTION[s]  $\leftarrow$  action; STATE[s]  $\leftarrow$  result
    PATH-COST[s]  $\leftarrow$  PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s]  $\leftarrow$  DEPTH[node] + 1
    add s to successors
  return successors
  
```

Best-first search

- Using an evaluation function $f(n)$ for each node:
 - Estimate of “desirability”
 - Expand the most desirable unexpanded node.
- Order all nodes in fringe in decreasing order of desirability.
- **Special cases:** greedy best-first search, A* search,...

Romanian traveling problem

How to go from Arad to Bucharest efficiently?



Straight-line distance
to Bucharest (**Value of h_{SL}**)

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

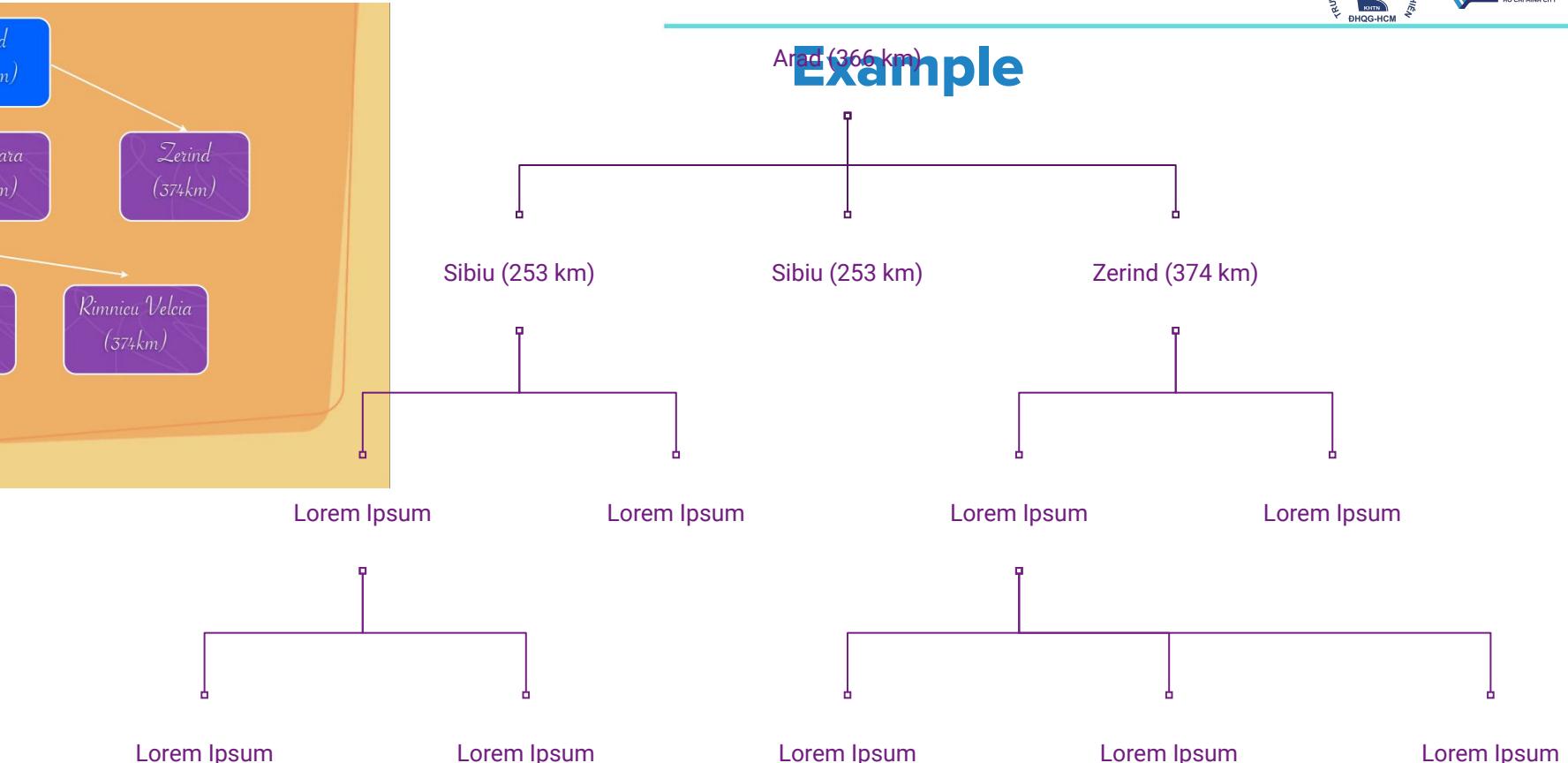
Greedy best-first search

- Evaluation function: $f(n) = h(n)$ (heuristic function)
- It can estimate the cost from step n to our goal.

For example: one can choose **$h_{SLD}(n)$** = the **straight-line distance** from the city at step n to Bucharest.

- Greedy best-first search **expands the node that appears to be closest to goal**

Example



Properties of greedy best-first search

Complete?

Time?

Space?

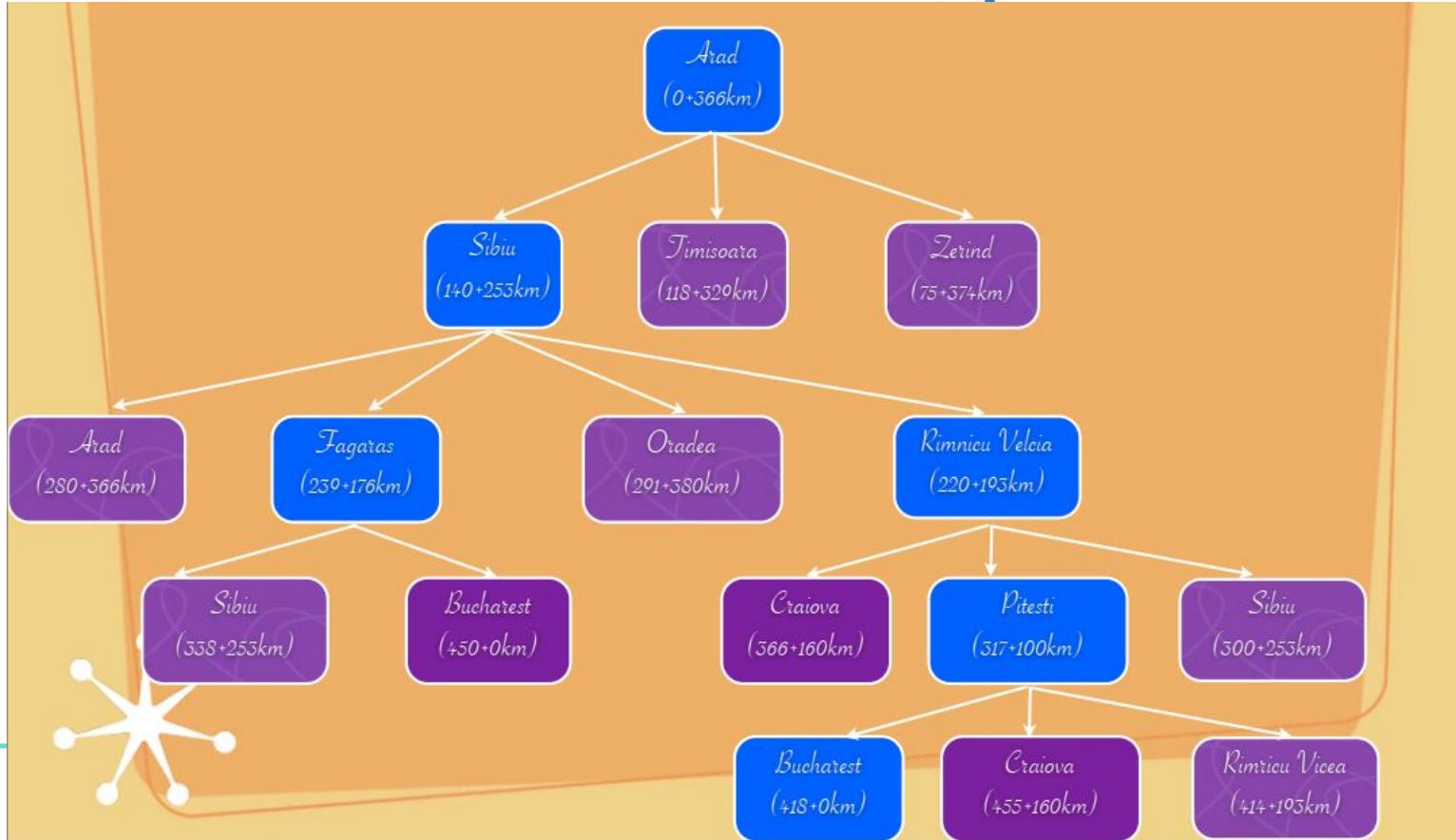
Optimal?

→HOMEWORK!

A* search

- Avoid expanding paths that are already expensive.
- Minimize the total estimated solution cost.
- Evaluate nodes by: $f(n) = g(n)+h(n)$
 - $g(n)$ = the path cost from the start node to node n.
 - $h(n)$ = the estimated cost of the cheapest path from node n to the goal.
 - $f(n)$ = estimated cost of the cheapest solution through n

A* search example



Admissible heuristics

- ❑ A heuristic $h(n)$ is admissible if for every node n , $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n .
- ❑ An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.
- ❑ Example: $h_{SLD}(n)$ (never overestimates the actual road distance).
- ❑ Since $g(n)$ is the exact cost to reach n , $f(n)$ never overestimates the true cost of a solution through n .

Admissible heuristics

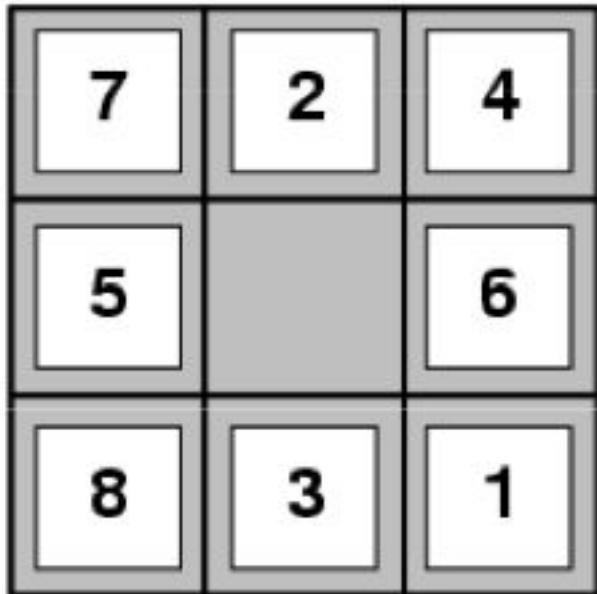
- ❑ Theorem: if $h(n)$ is admissible, A* using Tree-Search is optimal.
- ❑ Proof: **(homework 2)**
 - ❑ Suppose a suboptimal goal node G2 appears on the fringe and let the cost of the optimal solution be C^* . Prove that $f(G2) > C^*$.
 - ❑ Consider a fringe node n that is on an optimal solution path. Prove that $f(n) < C^*$.
 - ❑ So, G2 will not be expanded and A* must return an optimal solution.

Admissible heuristics

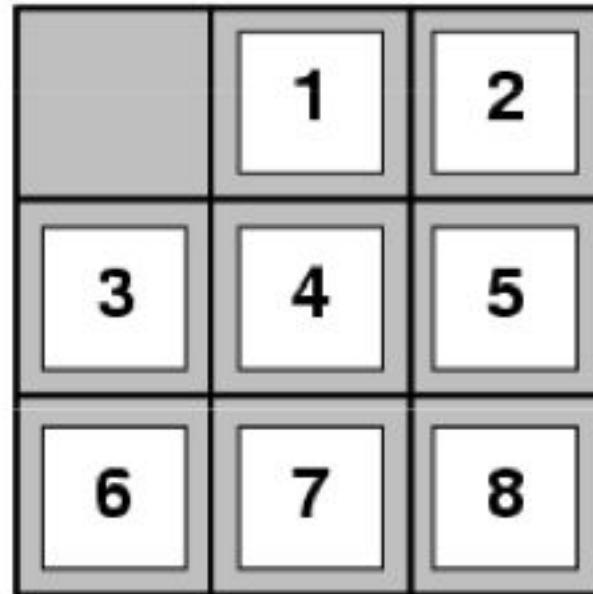
- ❑ If we use the Graph-Search instead of Tree-Search, the proof may fail.
- ❑ **Homework 3: Exercise 4.4 (book)**

Admissible heuristics

- ❑ Examples: the 8-puzzle:



Start State



Goal State

Admissible heuristics

- ❑ $h_1(n)$ = number of misplaced tiles.
- ❑ $h_2(n)$ = total Manhattan distance (i.e., the number of squares from desired location of each tile).
- ❑ $h_1(S) = ?$
- ❑ $h_2(S) = ?$

Consistent heuristics

- ❑ A heuristic is consistent if for every node n , every successor n' of n generated by an action “ a ”,
$$h(n) \leq c(n,a,n') + h(n')$$
- ❑ If h is consistent, we have:
$$f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n).$$
- ❑ i.e., $f(n)$ is non-decreasing along any path.

Consistent heuristics

- ❑ Theorem: If $h(n)$ is consistent, A* using GraphSearch is optimal.
- ❑ Proof: (**Homework 4**)

Properties of A* search

- Complete?
- Time?
- Space?
- Optimal?

(Homework 5)

Informed search algorithms

- ❑ Relaxed problems
- ❑ Local search algorithms

Relaxed problems

- ❑ A problem with fewer restrictions on the actions is called a relaxed problem
- ❑ The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
- ❑ If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, the heuristic function $h_1(n)$ gives the shortest solution

Relaxed problems

- ❑ If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.
- ❑ The original problem can be decomposed into many independent subproblems by the relaxed rules.

Learning heuristics

- ❑ A heuristic function $h(n)$ is supposed to estimate the cost of a solution beginning from state at node n .
- ❑ An agent can construct such a function by devising relaxed problems for which an optimal solution can be found easily.
- ❑ Another option is to learn from experience.

Learning heuristics

- ❑ For example, we solve a lot of 8-puzzles and get an optimal solution for each problem.
- ❑ Each such solution can provide examples for which one can learn an heuristic function $h(n)$.
- ❑ From these examples, one can construct a function $h(n)$ by an inductive learning that can predict solution costs from other states that arise during search (using neural networks, decision trees,...)

Learning heuristics

- ❑ Inductive learning methods work best when supplied with features of state that are relevant to its evaluation.
- ❑ For example: the feature “numbers of misplaced tiles” -> $x_1(n)$. Take 100 randomly generated 8-puzzle configurations and gather statistics on their actual solution costs.

Learning heuristics

- ❑ One may find that when $x1(n)$ is 5, the average solution is around 14,...
- ❑ A second feature $x2(n)$: “the number of pairs of adjacent tiles that are adjacent in the goal state as well”.
- ❑ Combine the statistical results from $x1(n)$ and $x2(n)$ to predict $h(n)$: $h(n)=c1.x1(n)+c2.x2(n)$

Local search algorithms

- ❑ Use a single current state and generally move only to neighbors of that state.
- ❑ Not systematic.
- ❑ Two key advantages:
 - ❑ use very little memory- usually a constant amount.
 - ❑ often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.

Local search algorithms

- ❑ Use to solve pure optimization problems

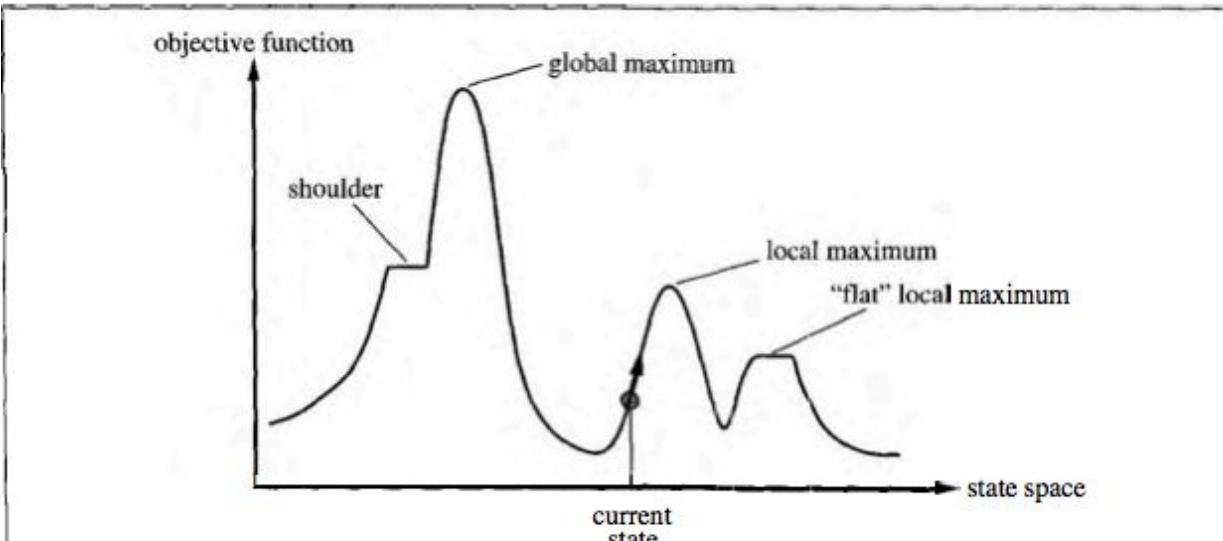


Figure 4.10 A one-dimensional state space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Hill-climbing algorithm

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

inputs: problem, a problem

local variables: current, a node

neighbor, a node

current \leftarrow MAKE-NODE(INITIAL-STATE[*problem*])

loop do

 neighbor \leftarrow a highest-valued successor of *current*

if VALUE[neighbor] \leq VALUE[current] **then return** STATE[current]

 current \leftarrow neighbor

Figure 4.11 The hill-climbing search algorithm (**steepest ascent** version), which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h .

Simulated annealing search

- ❑ A hill-climbing algorithm can get stuck on a local maximum.
Incomplete and fast.
- ❑ A purely random walk (moving to a successor chosen uniformly at random from the set of successors). Complete but extremely inefficient.
- ❑ Combine hill-climbing algorithm with a random walk in some ways that make efficiency and completeness.
→ simulated annealing.

Simulated annealing search

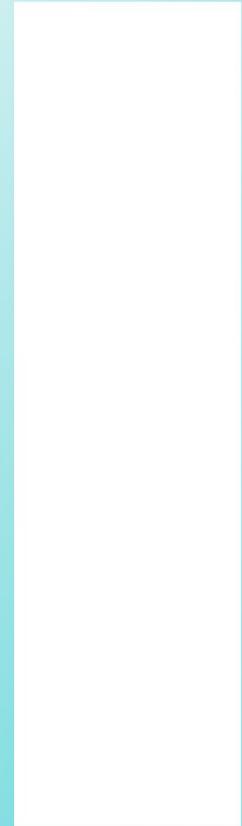
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                    next, a node
                    T, a "temperature" controlling the probability of downward steps

    current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
    for t  $\leftarrow$  1 to  $\infty$  do
        T  $\leftarrow$  schedule[t]
        if T = 0 then return current
        next  $\leftarrow$  a randomly selected successor of current
         $\Delta E \leftarrow$  VALUE[next] - VALUE[current]
        if  $\Delta E > 0$  then current  $\leftarrow$  next
        else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 
```

Figure 4.14 The simulated annealing search algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of *T* as a function of time.



ΔΙΣΙΔ



Knowledge representation

- Introduction
- Knowledge-based agent
- Kinds of knowledge
- Knowledge representation

Introduction

- ❑ Con người có thể cảm nhận thế giới quan bằng các giác quan, sử dụng các tri thức tích luỹ được và bằng các lập luận, suy diễn để đưa ra những hành động thích hợp.
- ❑ Các hệ thông minh (intelligent agent) cần phải có tri thức về thế giới hiện thực và môi trường xung quanh để đưa ra những quyết định đúng đắn

Knowledge-based agents

- ❑ Knowledge-based agent: là hệ tri thức chứa các cơ sở tri thức (knowledge base).
- ❑ Cơ sở tri thức là tập hợp các tri thức được biểu diễn dưới dạng nào đó.
- ❑ Mỗi khi nhận được thông tin đưa vào (input data), các agent phải có khả năng suy diễn để đưa ra những phương án chính xác, hợp lý.

Knowledge-based agents

- ❑ Hệ tri thức cần được trang bị một cơ chế suy diễn.
- ❑ Đối với hệ thống thông minh giải quyết một vấn đề nào thì cơ sở tri thức sẽ chứa các tri thức tương ứng.
- ❑ Tri thức (knowledge) là khái niệm trừu tượng.
- ❑ Theo từ điển Oxford, “knowledge is information and skills acquired through experience or education”.

Kinds of knowledge

- ❑ Tri thức lý thuyết (theoretical or a priori knowledge): là tri thức đạt được mà không cần quan sát thế giới quan.
- ❑ Tri thức thực tiễn (empirical knowledge): tri thức đạt được bằng những quan sát và tương tác với môi trường xung quanh
- ❑ Tri thức mô tả (declarative knowledge): bao gồm những từ ngữ mô tả chính xác về một sự vật, hiện tượng

Kinds of knowledge

- ❑ Tri thức quy trình (procedural knowledge): bao gồm những từ ngữ dùng để mô tả một quá trình (process) nào đó.
- ❑ Tri thức heuristic (heuristic knowledge): là tri thức nông cạn do không đảm bảo chính xác hoặc tối ưu khi giải quyết vấn đề. Thường được coi như mẹo nhầm dẫn dắt quá trình lập luận.

Expert system

- ❑ Chuyên gia (expert) là những người có kiến thức sâu sắc về một vấn đề nào đó và có thể giải quyết tốt những việc mà ít ai làm được.
- ❑ Hệ chuyên gia (expert system) là chương trình máy tính có thể thực hiện những công việc trong một lĩnh vực nào đó như một chuyên gia thực thụ.

Expert system

- ❑ Là các hệ tri thức dựa trên luật suy diễn phức tạp.
- ❑ Áp dụng trên rất nhiều lĩnh vực trong công nghiệp như marketing, nông nghiệp, y tế, điện lực,...

Knowledge representation

- ❑ Logic mệnh đề (propositional logic)
- ❑ Logic vị từ cấp một (first-order predicate logic).

Propositional logic

- ❑ Logic mệnh đề là công cụ logic trong đó các mệnh đề được mã hoá cho một biến hoặc hằng, còn các biểu thức là sự liên kết có nghĩa giữa các biến và các toán tử logic nhất định.
- ❑ Ví dụ: “Nếu tôi cố gắng làm bài tập (A) thì tôi sẽ thi tốt (B)”
được mô tả như sau: $A \rightarrow B$

Propositional logic

- ❑ Tri thức sẽ được mô tả dưới dạng các mệnh đề trong ngôn ngữ biểu diễn tri thức.
- ❑ Gồm hai thành phần cơ bản: cú pháp và ngữ nghĩa.
- ❑ Cú pháp của một ngôn ngữ bao gồm các ký hiệu và quy tắc liên kết các ký hiệu (luật cú pháp) để tạo thành các câu (công thức) trong ngôn ngữ.

Propositional logic

- ❑ Ngữ nghĩa của ngôn ngữ cho phép chúng ta xác định ý nghĩa của các câu trong một miền nào đó của thế giới thực.
- ❑ Ví dụ: $1+2+3+\dots+n$
- ❑ Ngoài cú pháp và ngữ nghĩa, ngôn ngữ biểu diễn tri thức cần được cung cấp cơ chế suy diễn, giúp chúng ta tìm ra một công thức mới từ một tập nào đó các công thức.

Propositional logic

- ❑ Ngôn ngữ biểu diễn tri thức = cú pháp + ngữ nghĩa+ cơ chế suy diễn.
- ❑ Một ngôn ngữ biểu diễn tri thức tốt cần có khả năng biểu diễn rộng, tức là mô tả được hầu hết điều chúng ta muốn.
- ❑ Hiệu quả đi đến kết luận + thủ tục suy diễn đòi hỏi ít thời gian và không gian nhớ.
- ❑ Càng gần với ngôn ngữ tự nhiên càng tốt

Cú pháp

- ❑ Cú pháp của logic mệnh đề đơn giản và cho phép xây dựng các công thức.
- ❑ Gồm tập các ký hiệu và tập các luật xây dựng công thức.

❑ Các ký hiệu:

- ❑ Hằng logic: true và false
- ❑ Các ký hiệu mệnh đề: P, Q, R,...
- ❑ Các phép kết nối logic: \wedge , \vee , \rightarrow , \leftrightarrow
- ❑ Các dấu mở ngoặc và đóng ngoặc

- ❑ Các quy tắc xây dựng công thức:

- ❑ $A \vee B$
- ❑ $A \wedge B$
- ❑ $A \leftrightarrow B$
- ❑ $A \rightarrow B \dots$

Propositional logic

- ❑ Ngữ nghĩa của logic mệnh đề giúp ta xác định được ý nghĩa thực sự của các công thức.
- ❑ Bất kỳ một sự kết hợp các ký hiệu mệnh đề với các sự kiện trong thế giới thực được gọi là minh họa (interpretation). Thường được gán một giá trị chân lý True hoặc False.

Propositional logic

- ❑ Bảng chân lý giúp ta xác định ngữ nghĩa câu phức hợp.
- ❑ Một công thức được gọi là thoả mãn (satisfiable) nếu nó đúng trong một minh họa nào đó.
- ❑ Ví dụ:
 $(P \vee Q) \wedge R$ thoả mãn vì nó có giá trị True khi {P: False, Q: True, R: True}

Propositional logic

- ❑ Một công thức gọi là vững chắc (valid) nếu nó đúng trong mọi minh họa.
- ❑ Một công thức được gọi là không thoả được nếu nó sai trong mọi minh họa.
- ❑ Ta gọi một mô hình (model) của một công thức là một minh họa để công thức đúng trong trường hợp đó

Propositional logic

- ❑ Cách xác định một công thức có vững chắc (thoả mãn, không thoả mãn): lập bảng chân trị.
- ❑ Một tập các công thức $G = (G_1, \dots, G_n)$ là vững chắc (thoả mãn, không thoả mãn) nếu hội của chúng $G_1 \wedge G_2 \wedge \dots \wedge G_n$ là vững chắc (thoả mãn, không thoả mãn).
- ❑ Một mô hình của G là mô hình của công thức $G_1 \wedge G_2 \wedge \dots \wedge G_n$.

Propositional logic

- ❑ Hai công thức được gọi là tương đương nếu chúng có cùng giá trị chân lý trong mọi trường hợp. Ta chỉ hai công thức A và B tương đương, ta viết $A \equiv B$.
- ❑ Luật De Morgan, giao hoán, kết hợp, phân phối.

Propositional logic

- ❑ Để viết chương trình trên máy tính thao tác các công thức, chúng ta thường chuẩn hoá chúng về dạng chuẩn tắc.
- ❑ Một công thức được gọi là chuẩn tắc nếu nó là hội của các câu phức tạp (clause). Một câu phức tạp có dạng $A_1 \vee A_2 \vee \dots \vee A_n$, trong đó A_i là các câu đơn (literal).

Propositional logic

➤ Phương pháp chuẩn hoá:

- Bỏ các dấu kéo theo bằng các luật.
- Chuyển các dấu phủ định bằng luật De Morgan.
- Áp dụng luật phân phối, thay công thức
 $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

Propositional logic

- ❑ Khi biểu diễn tri thức bởi các công thức trong logic mệnh đề, cơ sở tri thức là một tập nào đó các công thức. Bằng phương pháp chuẩn hoá vừa nêu, thì cơ sở tri thức là một tập hợp các câu phức hợp (clause)

Propositional logic

- ❑ Như vậy, mọi công thức đều có thể đưa về dạng chuẩn tắc là hội của các clause. Mỗi clause có dạng:

$\text{Not}(A_1) \vee \dots \vee \text{Not}(A_m) \vee B_1 \vee \dots \vee B_n$

trong đó A_i, B_i là các mệnh đề (literal dương).

- ❑ Câu Kowalski có dạng:

$A_1 \wedge \dots \wedge A_m \rightarrow B_1 \vee \dots \vee B_n$

Propositional logic

- ❑ Khi $n \leq 1$, clause chỉ chứa nhiều nhất một literal dương. Ta gọi những câu như thế là câu Horn (Alfred Horn, 1951).
- ❑ Nếu $m > 0$, $n = 1$, câu Horn có dạng: $A_1 \wedge \dots \wedge A_m \rightarrow B$
- ❑ trong đó các A_i được gọi là các điều kiện, còn B là kết luận. Các câu Horn còn gọi là các luật if-then.

Propositional logic

- ❑ Khi $m = 0$ và $n = 1$, câu Horn trở thành các câu đơn.
- ❑ Không phải mọi công thức đều có thể biểu diễn dưới dạng hội các câu Horn.
- ❑ Trong các ứng dụng, cơ sở tri thức thường là tập hợp các câu Horn (tập hợp các luận if-then)

Propositional logic

- ❑ Luật suy diễn:
 - ❑ Một công thức H được xem là hệ quả logic (logical consequence) của một tập các công thức G = {G₁,..,G_n} nếu trong bất kỳ minh họa nào mà {G₁,..,G_m} đúng thì H cũng đúng.
 - ❑ Luật suy diễn là phương pháp sử dụng những tri thức có sẵn trong cơ sở tri thức để suy ra tri thức mới là hệ quả logic của các công thức đó.

Propositional logic

- ❑ Luật Modus Ponens: $(A \rightarrow B, A) \Rightarrow B$
- ❑ Luật Mondus Tollens: $(A \rightarrow B, \text{not}(B)) \Rightarrow \text{not}(A)$
- ❑ Luật bắc cầu
- ❑ Luật loại bỏ hội

Propositional logic

- ❑ Luật đưa vào hội
- ❑ Luật đưa vào clause
- ❑ Luật phân giải: $(A \vee B, \text{not}(B) \vee C) \Rightarrow (A \vee C)$
- ❑ Một luật suy diễn là tin cậy nếu bất kỳ mô hình nào của giả thiết của luật cũng là mô hình của kết luận.

Propositional logic

- ❑ **Chứng minh:** luật phân giải là luật suy diễn tổng quát, bao gồm luật Modus Ponens, Modus Tollens, luật bắc cầu
(Homework)

Propositional logic

- ❑ Giả sử chúng ta có một tập các công thức. Bằng các luật suy diễn, ta có thể suy ra những công thức mới.
- ❑ Các công thức đã được cho được gọi là các tiên đề.
- ❑ Các công thức được suy ra được gọi là các định lý.
- ❑ Dãy các luật suy diễn được áp dụng để dẫn đến các định lý được gọi là một chứng minh của định lý

Propositional logic

- ❑ Nếu các luật suy diễn là tin cậy, thì các định lý là hệ quả logic của các tiên đề.
- ❑ Trong các hệ tri thức, bằng cách sử dụng các luật suy diễn, người ta thiết kế lên các thủ tục suy diễn để từ các tri thức trong cơ sở tri thức, ta suy ra các tri thức mới đáp ứng nhu cầu người sử dụng

Propositional logic

- ❑ Hệ hình thức (formal system) bao gồm một tập các tiên đề và một tập các luật suy diễn nào đó trong một ngôn ngữ biểu diễn tri thức nào đó.
- ❑ Một tập suy diễn được gọi là đầy đủ nếu mọi hệ quả logic của một tập các tiên đề đều chứng minh được bằng cách chỉ sử dụng các luật trong tập đó

Proof by contradiction

- ❑ Phương pháp chứng minh bác bỏ là phương pháp được sử dụng phổ biến trong toán học.
- ❑ Chứng minh bác bỏ bằng luật phân giải:
 - ❑ Luật phân giải trên các clause.
 - ❑ Luật phân giải trên các câu Horn.
 - ❑ Thuật toán Harvard (1970).
 - ❑ Thuật toán Robinson (1971)

Knowledge representation



continue

Proof by contradiction

- ❑ Để chứng minh P đúng, ta giả sử P sai và dẫn đến một mâu thuẫn.
- ❑ Để thuận tiện hơn cho việc sử dụng luật phân giải, ta sẽ cụ thể hoá luật phân giải trên các câu quan trọng

Proof by contradiction

- ❑ Luật phân giải trên các câu tuyên (clause)

$$\frac{A_1 \vee \dots \vee A_m \vee C \\ \neg C \vee B_1 \vee \dots \vee B_n}{A_1 \vee \dots \vee A_m \vee B_1 \vee \dots \vee B_n}$$

trong đó $A_1, A_2, \dots, A_m, C, B_1, \dots, B_n$ là literals.

Proof by contradiction

- ❑ Luật phân giải trên các câu Horn:

$$P_1 \wedge \dots \wedge P_m \wedge S \Rightarrow Q,$$

$$R_1 \wedge \dots \wedge R_n \Rightarrow S$$

$$P_1 \wedge \dots \wedge P_m \wedge R_1 \wedge \dots \wedge R_n \Rightarrow Q$$

- ❑ Hai câu có thể áp dụng được luật phân giải được gọi là hai câu phân giải được và kết quả nhận được gọi là phân giải thức của chúng

Proof by contradiction

- ❑ Hai câu phân giải được nếu một câu chứa một literal đối lập với một literal trong câu kia.
- ❑ Giả sử G là một tập các câu tuyển (clause). Ta ký hiệu $R(G)$ là tập bao gồm các câu thuộc G và tất cả các câu nhận được từ G thông qua dãy áp dụng luật phân giải.

Proof by contradiction

- ❑ Luật phân giải là luật đầy đủ để chứng minh một tập câu là không thoả được.
- ❑ Định lý phân giải: Một tập câu tuyển là không thoả được khi và chỉ khi $R(G)$ chứa câu rỗng

Proof by contradiction

❑ procedure Resolution

Input: G, tập các câu tuyển

- Repeat

+ Chọn 2 câu A,B thuộc G.

+ Nếu A, B phân giải được tính Res(A,B). Nếu Res(A,B) là câu mới, thêm vào G.

until nhận được [] hoặc ko có câu mới xuất hiện

- if nhận được câu rỗng then G không thoả được

else G thoả được.

Proof by contradiction

- ❑ Định lý phân giải có nghĩa là nếu từ các câu thuộc G, bằng cách áp dụng luật phân giải, ta dẫn đến câu rỗng thì G là không thoả được, còn nếu không thể sinh ra câu rỗng bằng luật phân giải thì G thoả được.
- ❑ Việc dẫn đến câu rỗng tức là đã dẫn đến hai literal đối lập nhau (hay dẫn đến mâu thuẫn)

Proof by contradiction

- ❑ Ví dụ: giả sử G là tập hợp các câu tuyển sau:

$$\neg A \vee \neg B \vee P \quad (1)$$

$$\neg C \vee \neg D \vee P \quad (2)$$

$$\neg E \vee C \quad (3)$$

$$A \quad (4)$$

$$E \quad (5)$$

$$D \quad (6)$$

Chứng minh P là hệ quả logic của các câu trên.

Proof by contradiction

- ❑ Thông thường chúng ta sẽ sử dụng bảng chân lý để kiểm tra tính đúng đắn của một biểu thức.
- ❑ Ngoài ra, chúng ta có thể sử dụng hai thuật toán sau đây:
 - ❑ Thuật toán Harvard (1970)
 - ❑ Thuật toán Robinson (1971)

Harvard algorithm

- Step 1: phát biểu lại giả thiết (GT) và kết luận (KL) của bài toán dưới dạng chuẩn sau:

$$GT_1, \dots, GT_n \rightarrow KL_1, \dots, KL_m$$

trong đó, GT_i , KL_j được xây dựng từ các biến m

- Step 2: bước bỏ phủ định. Khi cần bỏ các phủ định, chuyển vẽ

GT_i sang vẽ kết luận KL_j và ngược lại.

- Step 3: Thay dấu AND ở GT_i và OR ở KL_j bằng “,”

Harvard algorithm

- ❑ Step 4: Nếu GT_i còn dấu OR và KL_j còn dấu AND, tách chúng thành hai dòng con.
- ❑ Step 5: Một dòng được chứng minh nếu tồn tại chung một mệnh đề ở cả hai vế.
- ❑ Step 6: Bài toán được chứng minh khi và chỉ khi các dòng được chứng minh. Ngược lại, bài toán không được chứng minh.

Robinson algorithm

Robinson đã cải tiến thuật toán Havard.

- ❑ Step 1: phát biểu lại giả thiết (GT) và kết luận (KL) của bài toán dưới dạng chuẩn sau:

$$GT_1, \dots, GT_n \rightarrow KL_1, \dots, KL_m$$

trong đó, GT_i , KL_j được xây dựng từ các biến mệnh đề và các phép nối: AND, OR, NOT.

- ❑ Step 2: Thay dấu AND ở GT_i và OR ở KL_j bằng “,”

Robinson algorithm

- ❑ Step 3: Chuyển vẽ KL_j sang vẽ GT_i với dấu phủ định để còn một vẽ.
- ❑ Step 4: Xây dựng một mệnh đề mới bằng cách tuyển một cặp mệnh đề từ danh sách các mệnh đề. Nếu mệnh đề mới có các biến mệnh đề đối ngẫu thì mệnh đề đó được loại bỏ.
- ❑ Step 5: bổ sung mệnh đề mới này vào danh sách và lặp lại bước 4.
- ❑ Step 6: bài toán được chứng minh khi và chỉ khi còn hai mệnh đề đối ngẫu. Ngược lại bài toán không được chứng minh

Predicate logic

- ❑ Logic mệnh đề cho phép chúng ta biểu diễn các sự kiện.
- ❑ Mỗi ký hiệu trong logic mệnh đề được minh họa như những sự kiện trong thế giới thực, sử dụng các kết nối logic để tạo ra những câu phức hợp biểu diễn các sự kiện có ý nghĩa phức tạp hơn.
- ❑ Khả năng biểu diễn của logic mệnh đề chỉ giới hạn trong phạm vi các sự kiện.

Predicate logic

- ❑ Thế giới thực bao gồm các đối tượng. Các đối tượng có tính chất riêng và có quan hệ với nhau.
- ❑ Mỗi quan hệ giữa các đối tượng rất đa dạng, phong phú.
- ❑ Đối tượng: một sinh viên, một cái bàn, một giáo viên...

Predicate logic

- ❑ Tính chất: cái bàn có bốn chân, làm bằng gỗ, con số có tính chất là số thực, số hữu tỉ,...
- ❑ Quan hệ: cha con, anh em, bạn bè, thầy trò,..
- ❑ Hàm: quan hệ hàm. Ví dụ: quan hệ hàm ứng với mỗi người với ba mẹ họ.
- ❑ Logic vị từ đóng vai trò quan trọng vì khả năng biểu diễn của nó (cho phép ta biểu diễn tri thức về thế giới với các đối tượng, thuộc tính và các quan hệ của đối tượng), là cơ sở cho nhiều ngôn ngữ logic khác

Predicate logic - first order

- ❑ Để mô tả các thuộc tính của đối tượng, trong logic vị từ, người ta đưa vào các vị từ (predicate).
- ❑ Ngoài các kết nối logic như trong logic mệnh đề, logic vị từ cấp một còn sử dụng các lượng tử. Chẳng hạn, lượng tử \forall cho phép ta tạo ra các câu nói tới mọi đối tượng trong một miền đối tượng nào đó.

Predicate logic - first order

- ❑ Các ký hiệu:
 - ❑ Các ký hiệu hằng: a,b,c, John, Jerry,...
 - ❑ Các ký hiệu biến: x,y,z,u,v,w,...
 - ❑ Các ký hiệu vị từ: P, Q, R, S, Prime, Odd, Love...
- ❑ Mọi vị từ là vị từ của n biến. Ví dụ: Love là vị từ của hai biến, Prime là vị từ một biến.
- ❑ Các ký hiệu vị từ không biến là các ký hiệu mệnh đề.
- ❑ Các ký hiệu hàm: f, g, cos, sin,...

Predicate logic - first order

- ❑ Mỗi hàm là hàm của n biến.
- ❑ Các ký hiệu kết nối logic giống như trong logic mệnh đề.
- ❑ Các ký hiệu lượng tử: \forall , \exists
- ❑ Các ký hiệu ngăn cách: dấu phẩy, dấu mở ngoặc, đóng ngoặc.

Predicate logic - first order

- ❑ Các hạng thức (term) là các biểu thức mô tả đối tượng.
- ❑ Các hạng thức được xác định đệ quy như sau:
 - ❑ Các ký hiệu hằng hay biến là hạng thức.
 - ❑ Nếu a, b, c, \dots, z là n hạng thức và h là hàm n biến thì $h(a, b, c, \dots, z)$ cũng là hạng thức.
 - ❑ Một hạng thức không chứa biến được gọi là hạng thức cụ thể

Predicate logic - first order

- ❑ Chúng ta sẽ biểu diễn các tính chất của đối tượng và các quan hệ giữa các đối tượng bằng công thức phân tử (câu đơn).
- ❑ Các câu đơn được xác định đệ quy như sau:
 - ❑ Các ký hiệu vị từ không biến (các ký hiệu mệnh đề) là câu đơn.
 - ❑ Nếu a, b, c, \dots, z là n hạng thức và P là vị từ của n biến thì $P(a, b, \dots, z)$ là công thức phân tử (câu đơn).
 - ❑ Ví dụ: Mary là một ký hiệu hằng, Love là một vị từ hai biến, husband là hàm 1 biến thì $\text{Love}(\text{Mary}, \text{husband}(\text{Mary}))$ là một công thức phân tử.

Predicate logic - first order

- ❑ Từ các công thức phân tử, ta sử dụng các kết nối logic và các lượng từ để xây dựng các công thức (các câu) bằng đệ quy như sau:
 - ❑ Các công thức phân tử là các công thức.
 - ❑ Nếu G, H là các công thức thì các biểu thức logic của G và H là công thức.
 - ❑ Nếu G là một công thức và x là biến thì các biểu thức $(\exists x G), (\forall x G)$ là công thức.
 - ❑ Các công thức không phải là công thức phân tử thì được gọi là các công thức phức hợp

Predicate logic - first order

- ❑ Các công thức không chứa biến thì được gọi là các công thức cụ thể.
- ❑ Lượng tử phổ dụng \forall cho phép ta mô tả một lớp các đối tượng.
- ❑ Lượng tử tồn tại \exists cho phép ta nói đến một đối tượng nào đó trong một lớp đối tượng.
- ❑ Một công thức phân tử hoặc phủ định công thức phân tử được gọi là literal.
- ❑ Một công thức là tuyển của các literal được gọi là câu tuyển.

Predicate logic - first order

- ❑ Một công thức mà các biến bắt buộc xuất hiện thì gọi là công thức đóng.
- ❑ Ý nghĩa của các công thức trong một thế giới hiện thực nào đó thì được gọi là minh họa.
- ❑ Trong một minh họa, các ký hiệu vị từ sẽ được gắn với một thuộc tính hoặc một quan hệ cụ thể nào đó.
- ❑ Khi đã xác định được ngữ nghĩa một câu đơn, ta có thể xác định được ngữ nghĩa của các câu phức hợp.

Predicate logic - first order

- ❑ Hai công thức tương đương nếu như nó cùng sai hoặc cùng đúng trong mọi minh họa.
- ❑ Từ các câu phân tự, bằng cách sử dụng các kết nối logic và các lượng tử, ta có thể tạo ra các câu phức hợp có câu trúc phức tạp. Để dễ dàng cho việc lưu trữ các câu trong bộ nhớ và thuận lợi cho việc xây dựng các thủ tục suy diễn, ta cần chuẩn hoá các câu bằng cách đưa chúng về dạng chuẩn tắc hội (hội của các câu tuyển).

Predicate logic - first order

❑ Thủ tục chuẩn hoá các công thức:

- ❑ Loại bỏ các kéo theo
- ❑ Chuyển các phủ định tới các phân tử
- ❑ Loại bỏ các lượng tử tồn tại
- ❑ Loại bỏ các lượng tử phổ dụng
- ❑ Chuyển các tuyển tới literals
- ❑ Loại bỏ các hội
- ❑ Đặt tên lại các biến

Predicate logic - first order

❑ Các luật suy diễn:

- ❑ Luật thay thế phổ dụng
- ❑ Hợp nhất
- ❑ Luật Modus Ponens tổng quát.
- ❑ Luật phân giải tổng quát
- ❑ Luật phân giải trên các câu Horn

Predicate logic - first order

- ❑ Logic vị từ cấp 1 cho phép chúng ta biểu diễn các đối tượng trong thế giới thực với các tính chất của chúng và các quan hệ của chúng.
- ❑ Để biểu diễn tri thức của chúng ta về một miền các đối tượng nào đó trong logic vị từ cấp một, chúng ta cần đưa ra các ký hiệu hằng để chỉ ra các đối tượng cụ thể, các ký hiệu biến để chỉ ra các đối tượng bất kỳ trên miền đối tượng, các ký hiệu hàm để biểu diễn quan hệ hàm, các ký hiệu vị từ biểu diễn mối quan hệ khác nhau của các đối tượng.

Predicate logic - first order

- ❑ Các ký hiệu đã nêu tạo thành hệ thống từ vựng về miền đối tượng mà chúng ta quan tâm.
- ❑ Sử dụng các từ vựng đã đưa ra, chúng ta sẽ tạo ra các câu trong logic vị từ cấp một để biểu diễn tri thức của chúng ta về miền đối tượng đó.
- ❑ Tập hợp tất cả các câu được tạo thành sẽ lập nên cở sở tri thức trong hệ tri thức mong muốn.
- ❑ Ngoài ra, có thể sử dụng vị từ bằng, danh sách và các phép toán trên danh sách và tập hợp để biểu diễn tri thức mong muốn

Knowledge representation



continue

Thủ tục chuẩn hóa các công thức

❑ Loại bỏ các kéo theo:

Để loại bỏ các kéo theo, ta chỉ cần thay thế:

$$P \rightarrow Q \equiv \bar{P} \vee Q$$

$$P \rightarrow Q \equiv (\bar{P} \vee Q) \wedge (P \vee \bar{Q})$$

❑ Chuyển các phủ định tới các phân tử:

$$\overline{(\bar{P})} \equiv P \quad \overline{P \wedge Q} \equiv \bar{P} \vee \bar{Q} \quad \overline{P \vee Q} \equiv \bar{P} \wedge \bar{Q}$$

$$\overline{(\forall x P)} \equiv \exists x, \bar{P} \quad \overline{(\exists x P)} \equiv \forall x, \bar{P}$$

❑ Loại bỏ các lượng tử tồn tại:

Giả sử $P(x,y)$ có nghĩa là x nhỏ hơn y . Khi đó, $\forall x, \exists y P(x,y)$ có nghĩa là “Với mọi x , tồn tại y sao cho x nhỏ hơn y . Ta có thể xem y như là một hàm của x : $y=f(x)$. Khi đó, ta có thể loại bỏ lượng tử $\exists y$ và công thức trở thành: $\forall x, P(x,f(x))$.

Ví dụ: $\forall x (\exists y P(x,y)) \vee \forall u (\exists v (Q(a,v) \wedge \exists y \overline{R(x,y)}))$

sau khi loại bỏ lượng tử tồn tại trở thành:

$$P(x, f(x)) \vee (Q(a, g(u, x)) \wedge \overline{R(x, h(x, u))})$$

- ❑ Loại bỏ các lượng tử phổ dụng:

$$\forall x, P(x, f(x)) \vee \forall u \left(Q(a, g(u, x)) \wedge \overline{R(x, h(x, u))} \right)$$

trở thành:

$$P(x, f(x)) \vee \left(Q(a, g(u, x)) \wedge \overline{R(x, h(x, u))} \right)$$

❑ Chuyển các tuyển tới literal:

Thay các công thức dạng $P \vee (Q \wedge R)$ thành $(P \vee Q) \wedge (P \vee R)$ và các công thức dạng $(P \wedge Q) \vee R$ thành $(P \vee R) \wedge (Q \vee R)$

Khi đó

$$P(x, f(x)) \vee (Q(a, g(u, x)) \wedge \overline{R(x, h(x, u))})$$

sẽ được chuẩn hóa thành:

$$(P(x, f(x)) \vee Q(a, g(u, x))) \wedge (P(x, f(x)) \vee \overline{R(x, h(x, u))})$$

Thủ tục chuẩn hóa các công thức

❑ Loại các hội

Một câu hội là đúng nếu tất cả các thành phần của nó đều đúng. Do đó, công thức ở chuẩn tắc hội tương đương các thành phần của nó.

Do đó,

$$(P(x, f(x)) \vee Q(a, g(u, x))) \wedge (P(x, f(x)) \vee \overline{R(x, h(x, u))})$$

tương đương hai câu tuyển:

$$P(x, f(x)) \vee Q(a, g(u, x)) \text{ và } P(x, f(x)) \vee \overline{R(x, h(x, u))}$$

❑ Đặt tên lại các biến:

Đặt tên lại các biến sao cho biến ở hai câu khác nhau thì khác nhau.

Ví dụ:

$$P(x, f(x)) \vee Q(a, g(u, x)) \text{ và } P(x, f(x)) \vee \overline{R(x, h(x, u))}$$

có cùng tên biến là x, ta có thể đặt tên lại:

$$P(x, f(x)) \vee Q(a, g(u, x)) \text{ và } P(z, f(z)) \vee \overline{R(z, h(z, u))}$$

Thủ tục chuẩn hóa các công thức

- ❑ Khi tri thức là tập hợp nào đó các công thức trong logic vị từ, bằng cách áp dụng các thủ tục vừa nêu, chúng ta xây dựng được cơ sở tri thức chỉ gồm các câu tuyễn.
- ❑ Tương tự như logic mệnh đề, các câu tuyễn có thể biểu diễn dưới dạng các câu Kowalski:

$$P_1 \wedge P_2 \wedge \dots \wedge P_n \rightarrow Q_1 \vee Q_2 \vee \dots \vee Q_m$$

- ❑ Một trường hợp đặc biệt của câu Kowalski là câu Horn (if...then)

Các luật suy diễn

- ❑ Trong logic mệnh đề, ngoài những luật quan trọng như luật Modus Ponens, luật Modus Tolens, bắc cầu..., ta đã chỉ ra được luật phân giải là luật đầy đủ cho chứng minh bác bỏ. Ta sẽ mở rộng kết quả này cho logic vị từ.
- ❑ Tất cả các luật suy diễn được đưa ra trong logic mệnh đề đều đúng trong logic vị từ cấp một.

Luật thay thế phổ dụng

- ❑ Là luật suy diễn quan trọng trong logic vị từ.
- ❑ Giả sử A là một câu, câu $\forall x A$ là đúng trong một minh họa nào đó nếu và chỉ nếu A đúng đối với tất cả các đối tượng nằm trong miền đối tượng của minh họa đó.
- ❑ Mỗi hạng thức ứng với một đối tượng khi thế vào biến x trong câu $\forall x A$ sẽ cho ra câu đúng nếu $\forall x A$ đúng. Công thức nhận được từ công thức A bằng cách thay thế tất cả các xuất hiện của biến x bởi t ký hiệu là $A[x/t]$.

Luật thay thế phổ dụng

- ❑ Luật thay thế phổ dụng (universal instantiation): từ công thức $\forall x A$, ta suy ra công thức $A[x/t]$.
- ❑ Ví dụ: $\forall x \text{Like}(x, \text{ăn chè})$ có nghĩa là một người đều thích ăn chè. Thay $x = \text{Khoa}$, ta suy ra $\text{Like}(\text{Khoa}, \text{ăn chè})$ nghĩa là Khoa thích ăn chè.

Luật hợp nhất

- Giả sử ta có hai câu phân tử:

$\forall y \text{Like}(\text{Nam}, y)$ và $\forall x \text{ Like}(x, \text{Football})$,

Bằng cách sử dụng phép thẽ $[x/\text{Nam}, y/\text{Football}]$, ta có thể hợp nhất hai câu trên thành $\text{Like}(\text{Nam}, \text{Football})$.

- Trong các suy diễn, ta cần sử dụng phép hợp nhất các câu bằng phép thẽ như ví dụ trên.

Luật hợp nhất

- ❑ Ví dụ: cho trước hai câu sau đây:
 $\forall x, \text{Friend}(x, \text{Nam}) \rightarrow \text{Good}(x)$: Mọi người bạn của Nam đều tốt
 $\forall y, \text{Friend}(\text{Lan}, y)$: Lan là bạn của tất cả mọi người
- ❑ Ta có thể hợp nhất hai câu trên bằng cách thay thế [x/Lan,y/Nam].
Áp dụng luật thay thế phổ dụng, ta sinh ra hai câu mới:
 $\text{Friend}(\text{Lan}, \text{Nam}) \rightarrow \text{Good}(\text{Lan})$ và $\text{Friend}(\text{Lan}, \text{Nam})$
Từ hai câu trên, theo luật Modus Ponens, ta suy ra $\text{Good}(\text{Lan})$.

Luật hợp nhất

- Không mất tính tổng quát, ta gọi phép thế θ là một dãy các cặp

$$x_i/t_i, \theta = [x_1/t_1, \dots, x_n/t_n]$$

trong đó x_i là các biến khác nhau, t_i là các hạng thức và x_i không có mặt trong t_i .

- Áp dụng phép thế θ vào công thức A, ta được công thức A_θ .
- Hai công thức phân tử A và B mà tồn tại phép thế θ sao cho $A_\theta = B_\theta$ được gọi là hợp nhất được và phép thế θ được gọi là hợp nhất tử của A và B.

Luật Modus Ponens tổng quát

- ❑ Giả sử P_i, P'_i ($i=1\dots n$) và Q là các công thức phân tử sao cho tất cả các cặp P_i, P'_i đều hợp nhất được qua phép thế θ .
- ❑ Khi đó ta có luật:

$$\frac{(P_1 \wedge \dots \wedge P_n \rightarrow Q), P'_1, \dots, P'_n}{Q'}$$

trong đó, $Q = Q_\theta$

Luật phân giải trên các câu tuyển

- ❑ Giả sử ta có hai câu tuyển $A_1 \vee \dots \vee A_m \vee C$ và $B_1 \vee \dots \vee B_n \vee D$, trong đó A_i và B_j là các literals còn C và D là các câu phân tử có thể hợp nhất được bằng phép thế θ : $C_\theta = D_\theta$.
- ❑ Khi đó, ta có luật sau:

$$\frac{A_1 \vee \dots \vee A_m \vee C, B_1 \vee \dots \vee B_n \vee D}{A'_1 \vee \dots \vee A'_m, B'_1 \vee \dots \vee B'_n}$$

trong đó:

$$A'_i = (A_i)_\theta \text{ và } B'_j = (B_j)_\theta$$

Luật phân giải trên các câu Horn

- ❑ Các câu Horn là các câu có dạng:

$$P_1 \wedge P_2 \dots \wedge P_n \rightarrow Q$$

- ❑ Giả sử ta có hai câu Horn: $P_1 \wedge P_2 \dots \wedge P_n \wedge S \rightarrow Q$ và $R_1 \wedge R_2 \dots \wedge R_m \rightarrow T$

trong đó hai câu S và T hợp nhất được bằng phép thế θ : $S_\theta = T_\theta$.

Khi đó ta có luật:

$$\frac{P_1 \wedge P_2 \dots \wedge P_n \wedge S \rightarrow Q \\ R_1 \wedge R_2 \dots \wedge R_m \rightarrow T}{P'_1 \wedge P'_2 \dots \wedge P'_n \wedge R'_1 \wedge R'_2 \dots \wedge R'_m \rightarrow Q'}$$

trong đó: $P'_i = (P_i)_\theta$, $R'_j = (R_j)_\theta$ và $Q' = Q_\theta$

Luật phân giải trên các câu Horn

- Thông thường, ta thường sử dụng:

$$\frac{P_1 \wedge P_2 \dots \wedge P_n \wedge S \rightarrow Q}{T}$$
$$\frac{}{P'_1 \wedge P'_2 \dots \wedge P'_n \rightarrow Q'}$$

trong đó:

$$P'_i = (P_i)_{\theta} \text{ và } Q' = Q_{\theta}$$

- Ví dụ: Student(x) \wedge Male(x) \rightarrow Play(x,Football) và Male(Nam).

Biểu diễn tri thức bằng luật và lập luật

- ❑ Với cơ sở tri thức gồm các câu trong logic vị từ cấp một, ta có thể chứng minh công thức có là hệ quả logic của cơ sở tri thức hay không bằng phương pháp chứng minh bắc bỏ hoặc thủ tục phân giải.
- ❑ Tuy nhiên, thủ tục này có độ phức tạp cao và đòi hỏi chiến lược giải một cách thích hợp.
- ❑ Chính vì thế, chúng ta cần tìm các tập con của logic vị từ cấp một sao cho chúng có đủ khả năng biểu diễn cơ sở tri thức trong nhiều lĩnh vực và có thể đưa ra thủ tục suy diễn hiệu quả.¹⁴

Biểu diễn tri thức bằng luật và lập luật

- ❑ Các tập con này của logic vị từ cấp một sẽ xác định các ngôn ngữ biểu diễn tri thức đặc biệt.
- ❑ Trong bài giảng hôm nay, chúng ta sẽ nghiên cứu ngôn ngữ chỉ bao gồm các câu Horn, hay các luật if...then.
- ❑ Nhìn chung, nếu chỉ sử dụng câu Horn, chúng ta không thể biểu diễn hết mọi điều trong logic vị từ cấp một.
- ❑ Tuy nhiên, ta vẫn có thể biểu diễn được một khối lượng lớn tri thức trong nhiều lĩnh vực khác nhau và có thể sử dụng các thủ tục suy diễn hiệu quả.

Biểu diễn tri thức bằng luật sinh

- ❑ Ngôn ngữ bao gồm các luật if-then được gọi là luật sản xuất hay luật sinh (production rule) là ngôn ngữ phổ biến nhất trong biểu diễn tri thức.
- ❑ Các câu Horn thường được viết dưới dạng:

“Nếu $P_1, P_2, \dots, \text{và } P_n$ thì Q”,

trong đó các câu P_i ($i=1\dots n$) được gọi là các điều kiện và Q được gọi là kết luận của luật.

Biểu diễn tri thức bằng luật sinh

- ❑ Các luật if-then có ưu điểm sau đây:
 - ❑ Một luật if-then mô tả một phần nhỏ tương đối độc lập của tri thức.
 - ❑ Có thể thêm những luật mới hoặc loại bỏ một số luật cũ ra khỏi cơ sở tri thức mà không ảnh hưởng nhiều đến các luật khác.
 - ❑ Có khả năng đưa ra lời giải thích cho các quyết định của hệ tri thức.
 - ❑ Là dạng biểu diễn tự nhiên của tri thức.
 - ❑ Giúp chúng ta biểu diễn một số lượng lớn tri thức của con người trong tất cả các lĩnh vực của đời sống, khoa học, kỹ thuật...

Biểu diễn tri thức bằng luật sinh

❑ Một luật chuẩn đoán bệnh:

Nếu:

- + Bệnh nhân ho lâu ngày.
- + Bệnh nhân thường sốt vào buổi chiều

Thì: bệnh nhân có khả năng bệnh lao.

Biểu diễn tri thức bằng luật sinh

- ❑ Thông thường, mỗi phần kết luận của một luật xác định một khẳng định mới được suy ra khi tất cả các điều kiện của luật thỏa mãn.
- ❑ Tuy nhiên, trong một số trường hợp, phần kết luận của luật là một hành động hệ cần thực hiện. Ta gọi các luật như thế là luật hành động.
- ❑ Hành động trong luật hành động có thể là: thêm vào sự kiện mới, loại bỏ một sự kiện có trong bộ nhớ làm việc hoặc thực hiện một thủ tục nào đó...

Biểu diễn tri thức bằng luật sinh

- ❑ Phân biệt hai dạng hệ: hệ diễn dịch và hệ hành động dựa trên luật.
- ❑ Một luật được gọi là cháy được nếu tất cả các điều kiện của nó đền thỏa mãn.
- ❑ Trong các hệ hành động, khi có nhiều hơn một luật có thể cháy, ta cần có chiến lược giải quyết và chậm để quyết định cho luật nào cháy trong các luật có thể cháy.

Biểu diễn tri thức không chắc chắn

- ❑ Trong thực tế, có rất nhiều điều mà chúng ta không tin tưởng chúng là đúng hoặc sai.
- ❑ Ví dụ: dự báo thời tiết, chuẩn đoán máy móc hỏng,...
- ❑ Trong các hệ dựa vào luật, chúng ta phải đưa vào mức độ chắc chắn của các luật và sự kiện trong cơ sở tri thức.
- ❑ Mức độ chắc chắn là một con số nằm giữa 0 và 1.
- ❑ Cách viết:

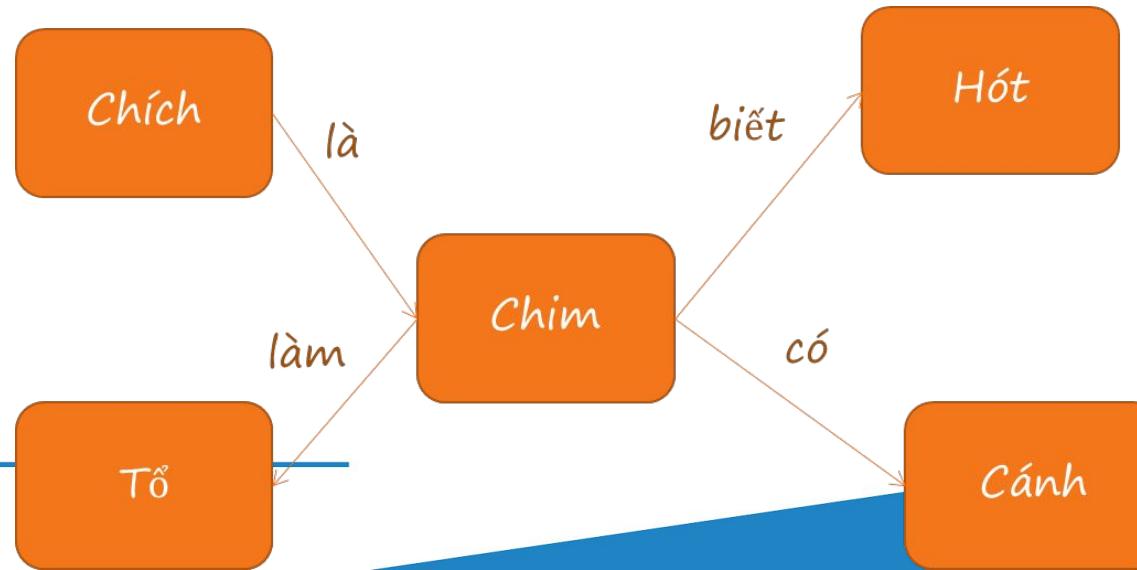
$$A_1 \wedge \dots \wedge A_n \rightarrow B : C$$

Biểu diễn tri thức không chắc chắn

- Có nghĩa là luật $A_1 \wedge \dots \wedge A_n \rightarrow B : C$ có độ chắc chắn là C.
- Đòi hỏi phải có phương pháp xác định mức độ chấn chấn của các kết luận được suy ra.
- Giả sử ta xét luật chỉ có một điều kiện: $A \rightarrow B : C$
- Ta có: $P(B) = P(B|A)P(A)$
- Khi đó: $C = P(B|A)$ là xác suất có điều kiện của B khi A xảy ra.
- Trong các trường hợp khác, C có thể được tính bằng các phương pháp khác nhau.

Biểu diễn tri thức bằng mạng ngữ nghĩa

- Chúng ta có thể biểu diễn tri thức thông qua mạng ngữ nghĩa.
- Tri thức được biểu diễn dựa trên bản đồ, trong đó định là các đối tượng còn các cung biểu diễn mối quan hệ giữa các đối tượng.



Neural Network



Tổng quan về neural network

Tổng quan về Neural Network

- ❑ Artificial Neural Network là mô hình xử lý thông tin được mô phỏng dựa trên hoạt động của hệ thần kinh sinh vật.
- ❑ Bao gồm số lượng lớn các Neural được gắn kết để xử lý thông tin.
- ❑ Artificial Neural Network giống như não người, được học bở kinh nghiệm thông qua huấn luyện, có khả năng lưu trữ tri thức và sử dụng chúng trong việc dự đoán những dữ liệu chưa biết.

Natural neurons

- ❑ Natural neurons nhận tín hiệu thông qua các khớp thần kinh (synapses) trên các cấu trúc hình cây (dendrites) hoặc các màng (membrane) của tế bào thần kinh (neuron).

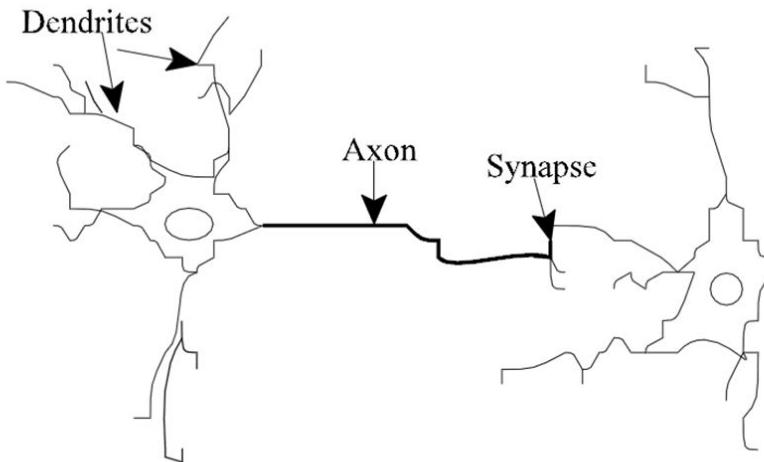


Figure 1. Natural neurons (artist's conception).

Natural neurons

- When a signal is received strong enough (crossing a threshold), neurons will be activated and transmit a signal through an axon (nerve fibre)

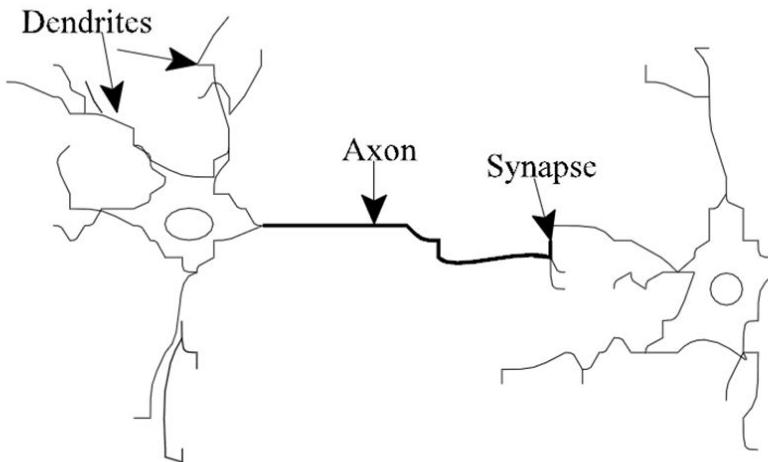


Figure 1. Natural neurons (artist's conception).

Natural neurons

- ❑ Tín hiệu này có thể được truyền đến synapse khác và có kích hoạt tiếp những neuron khác.

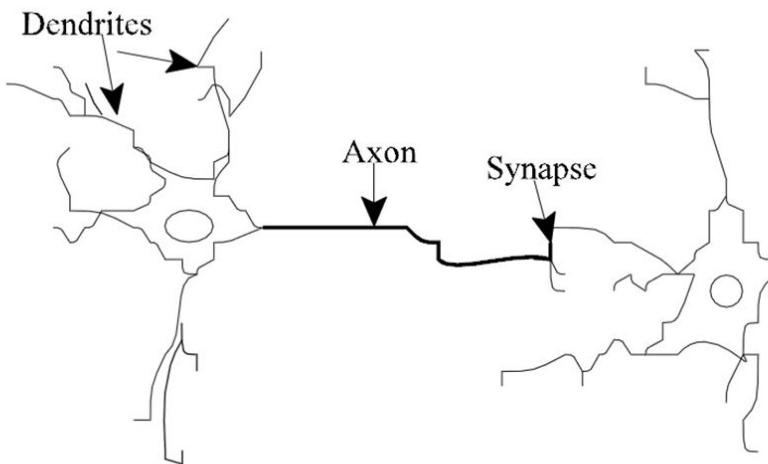


Figure 1. Natural neurons (artist's conception).

Cấu trúc của một neuron nhân tạo

- ❑ Một neural nhân tạo (artificial neuron) bao gồm ba thành phần chính:
 - ❑ Input (giống như synapses): được nhân bởi các trọng số (weights), mô tả độ lớn của tín hiệu.

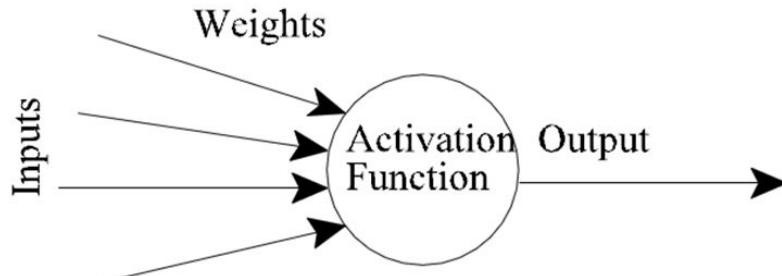


Figure 2. An artificial neuron

Cấu trúc của một neuron nhân tạo

- ❑ Một neural nhân tạo (artificial neuron) bao gồm ba thành phần chính:
 - ❑ Các inputs sau đó được tính toán thông qua một hàm toán học để xác định sự kích hoạt của neuron.

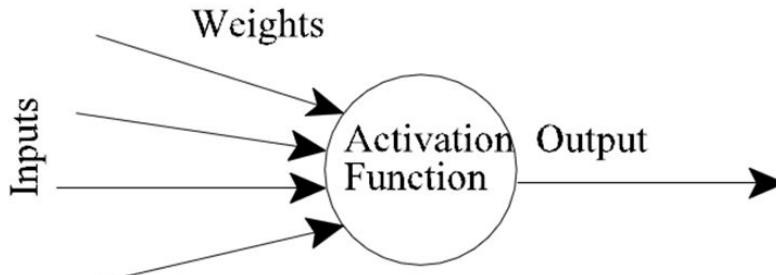


Figure 2. An artificial neuron

Cấu trúc của một neuron nhân tạo

- ❑ Một neural nhân tạo (artificial neuron) bao gồm ba thành phần chính:
 - ❑ Mạng neuron nhân tạo sẽ kết hợp nhiều artificial neuron như thế để tiến hành xử lý thông tin. Kết quả xử lý của một neuron có thể làm input cho một neuron khác.

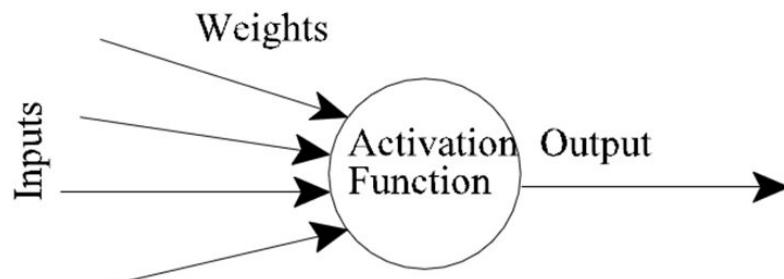
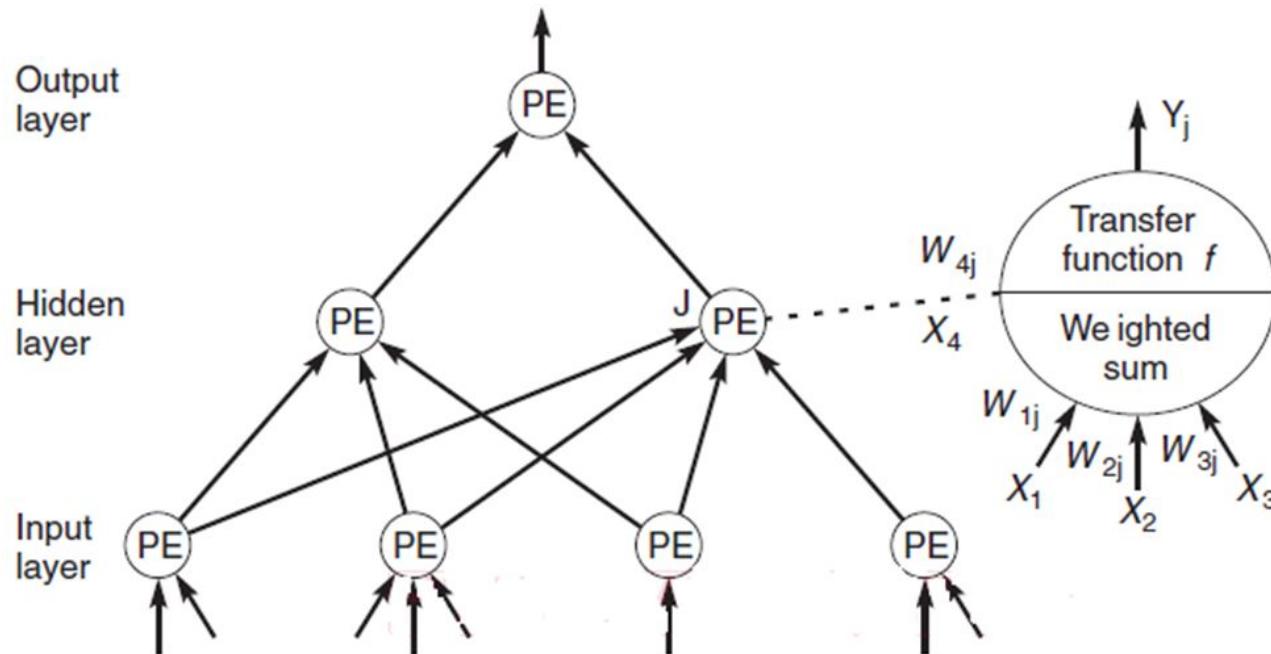


Figure 2. An artificial neuron

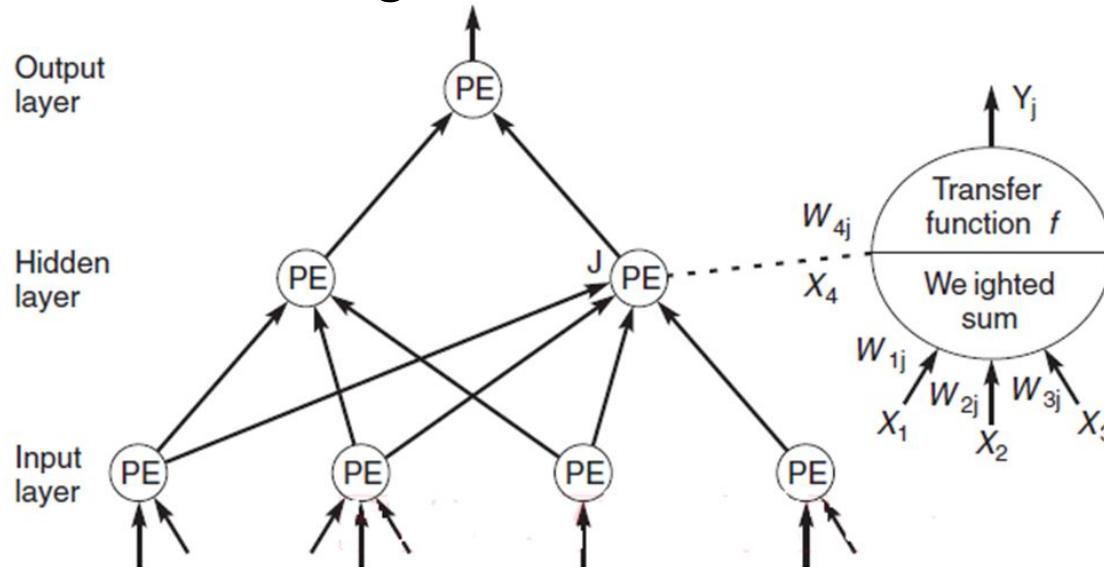
Cấu trúc của một neuron nhân tạo



(PE) = processing element

Cấu trúc của một neuron nhân tạo

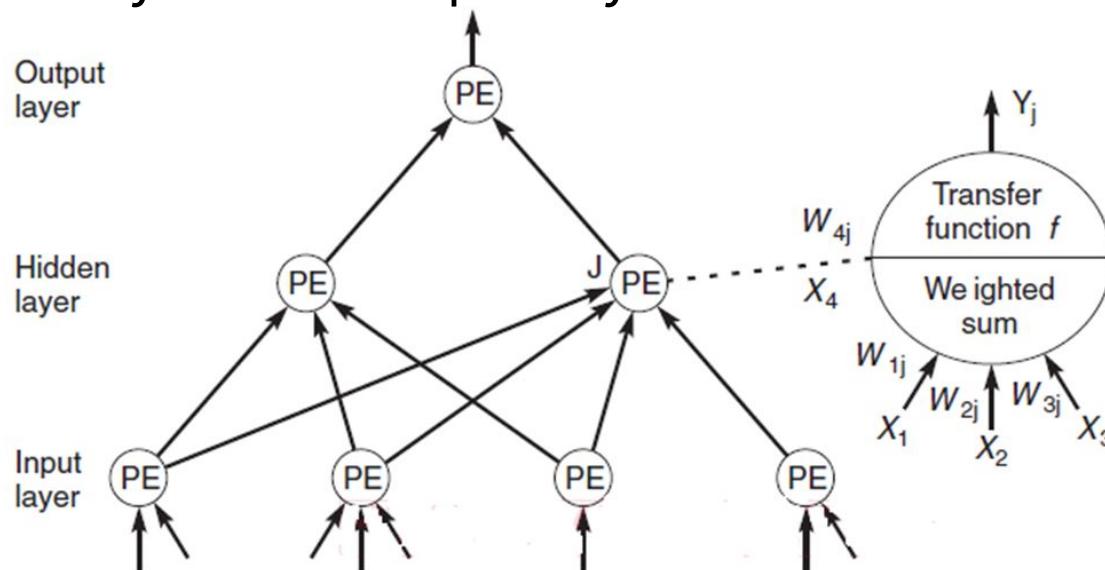
- ❑ Các đơn vị xử lý (processing elements) của một artificial neuron network là những neuron.



(PE) = processing element

Cấu trúc của một neuron nhân tạo

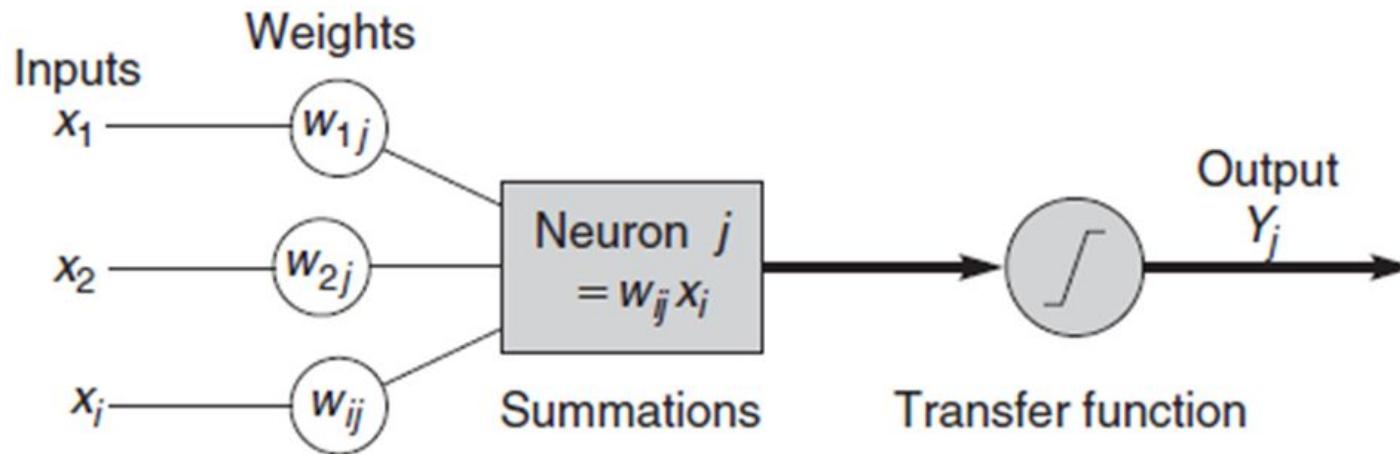
- Một artificial neuron network gồm 3 thành phần chính: input layer, hidden layer and output layer.



(PE) = processing element

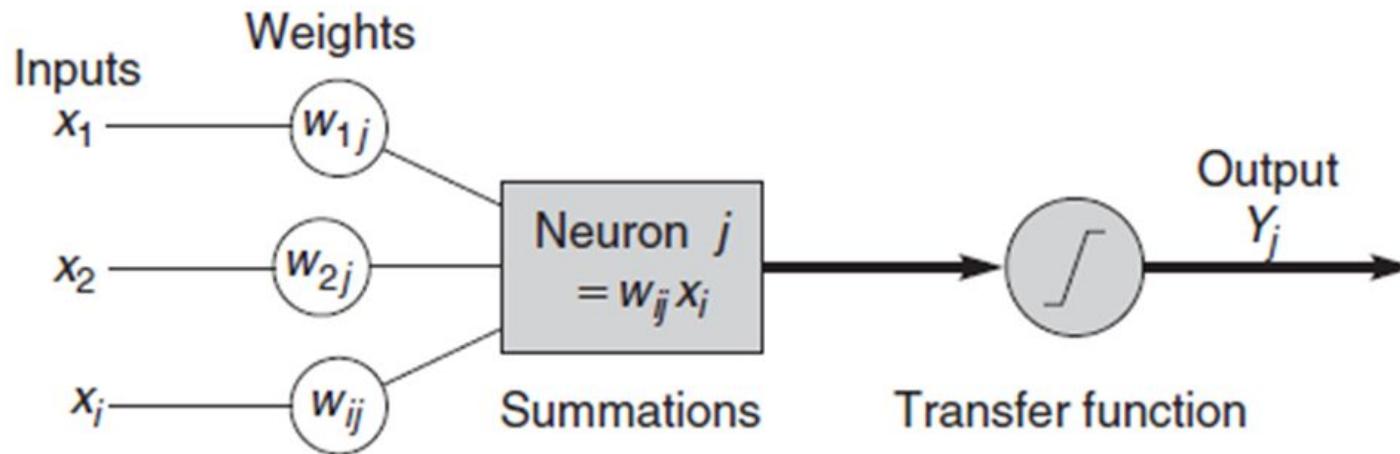
Quá trình xử lý thông tin của ANN

- Sau đây là mô hình chi tiết cho quá trình xử lý thông tin trên ANN:



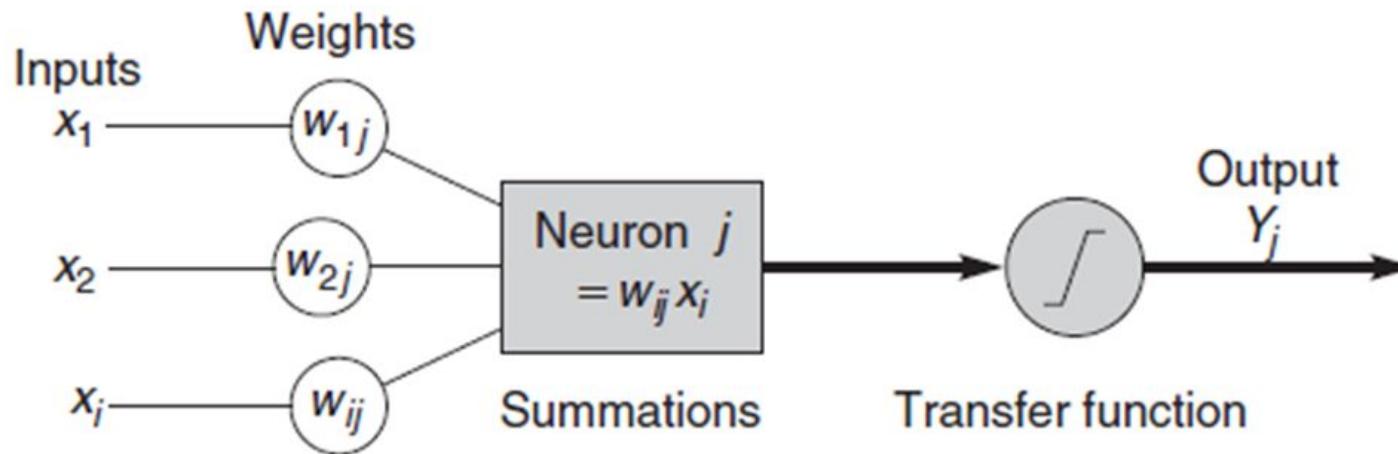
Quá trình xử lý thông tin của ANN

- Inputs: mỗi input tương ứng với một thuộc tính (attribute) của dữ liệu (patterns). Ví dụ: xét hệ thống đánh giá mức độ rủi ro cho vay trong ngân hàng, mỗi input là một thuộc tính của khách hàng như thu nhập, nghề nghiệp, giới tính...



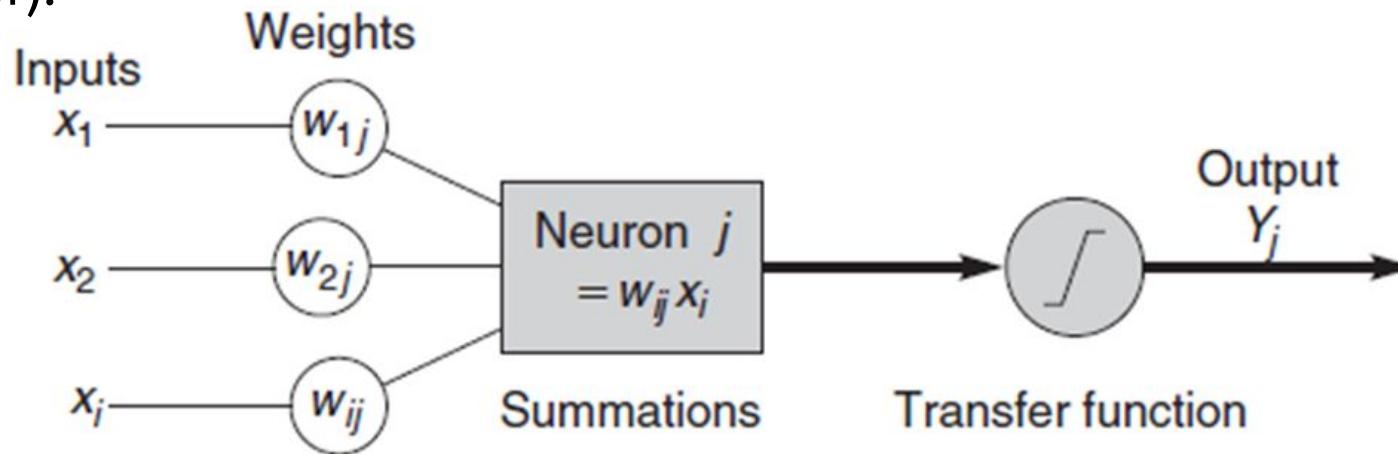
Quá trình xử lý thông tin của ANN

- Outputs: là kết quả của Artificial Neuron Networks hay một giải pháp cho một vấn đề. Ví dụ: trong bài toán xem xét chấp nhận cho khách hàng vay tiền trong ngân hàng sẽ là cho vay hay không cho vay...



Quá trình xử lý thông tin của ANN

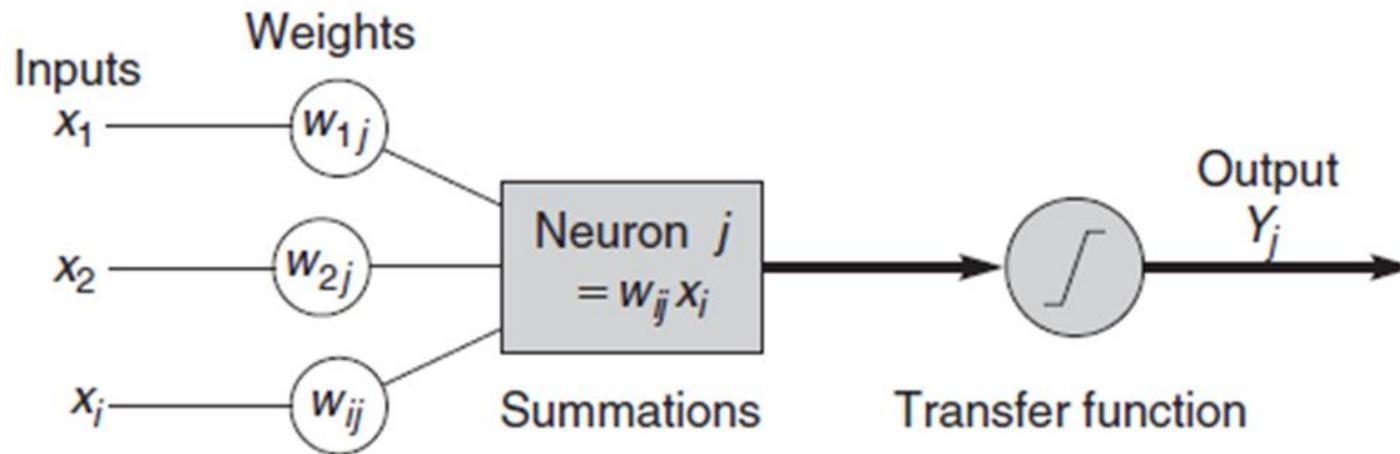
- Trọng số liên kết (connection weights): là thành phần vô cùng quan trọng trong một hệ thống mạng neuron nhân tạo. Nó thể hiện mức độ quan trọng của dữ liệu đầu vào đối với quá trình xử lý thông tin (quá trình chuyển đổi dữ liệu giữa các layer).



Quá trình xử lý thông tin của ANN

- ❑ Hàm tổng (summation function): tính tổng trọng số của tất cả các input được đưa vào mỗi neuron. Một hàm tổng của một neuron đối với n input sẽ được tính theo công thức sau đây:

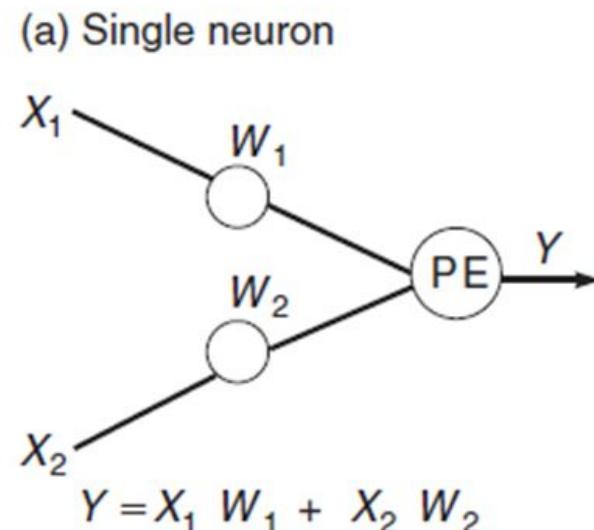
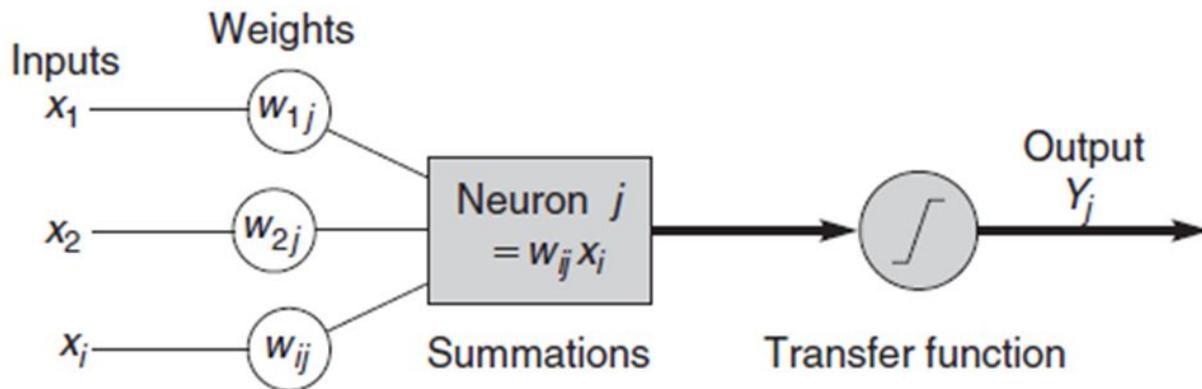
$$Y = \sum_{i=1}^n X_i W_i$$



Quá trình xử lý thông tin của ANN

- ❑ Hàm tổng đối với nhiều neurons trong cùng một layer:

$$Y_j = \sum_{i=1}^n X_i W_{ij}$$

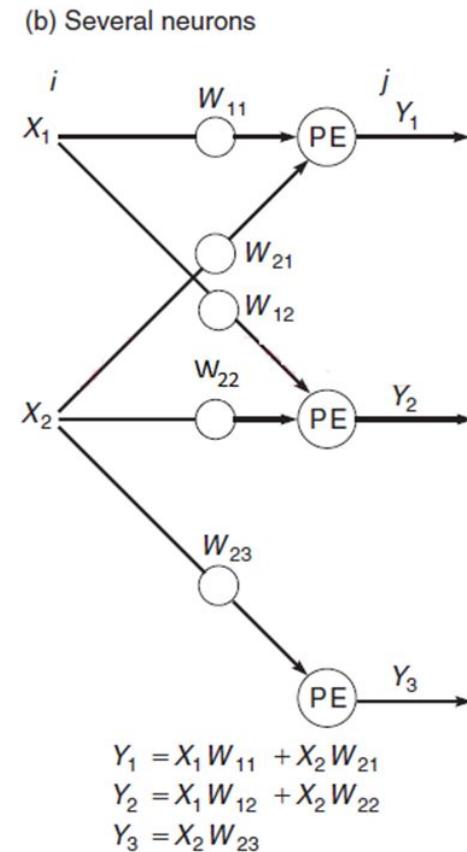
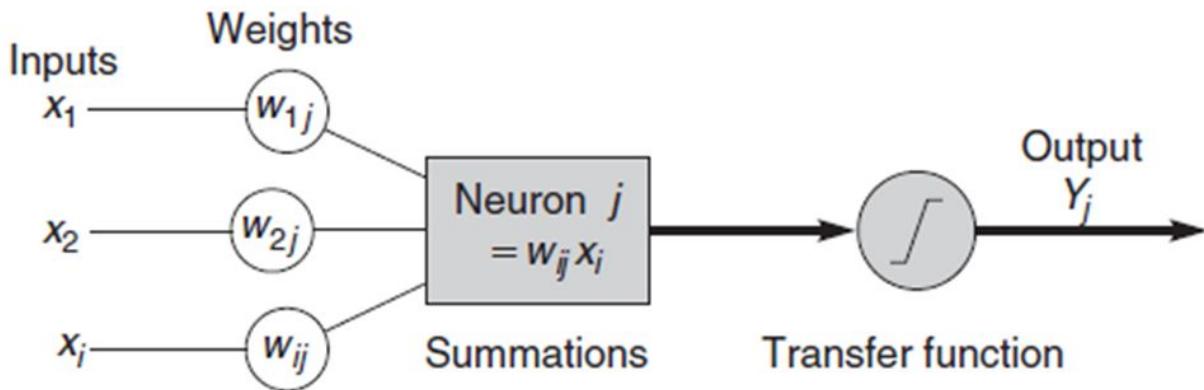


PE = processing element

Quá trình xử lý thông tin của ANN

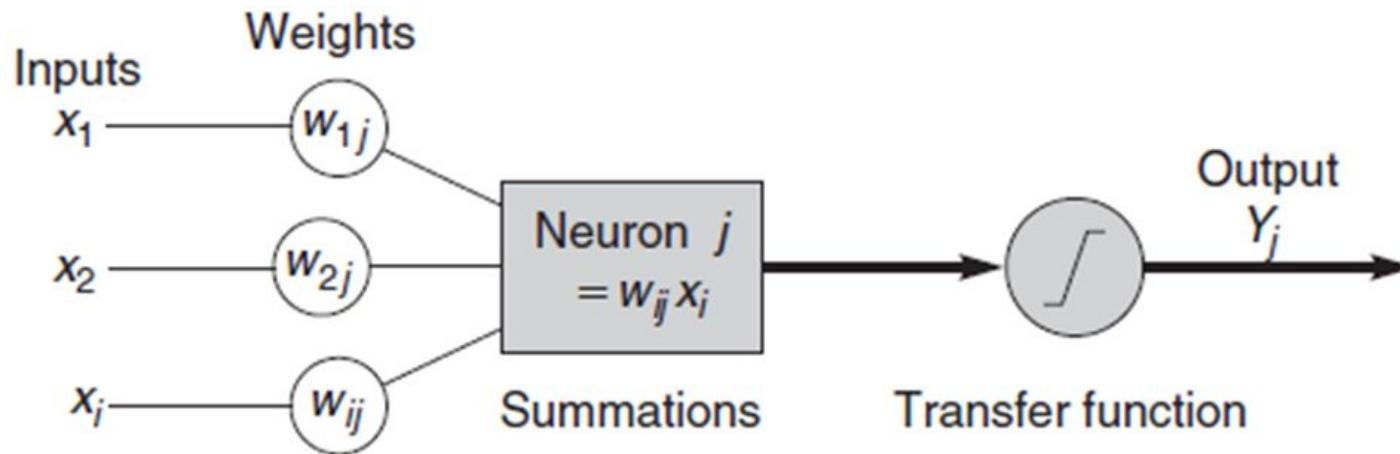
- ❑ Hàm tổng đối với nhiều neurons trong cùng một layer:

$$Y_j = \sum_{i=1}^n X_i W_{ij}$$



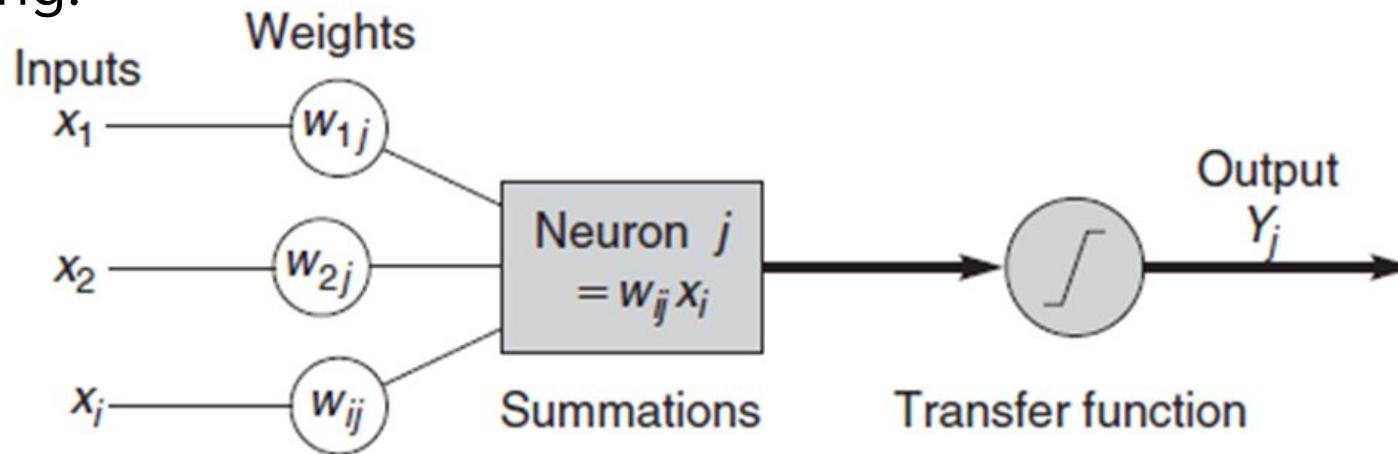
Quá trình xử lý thông tin của ANN

- ❑ Hàm chuyển đổi (transformation function): hàm tổng của một neuron cho chúng ta biết khả năng kích hoạt của neuron đó và còn gọi là kích hoạt bên trong (internal activation).



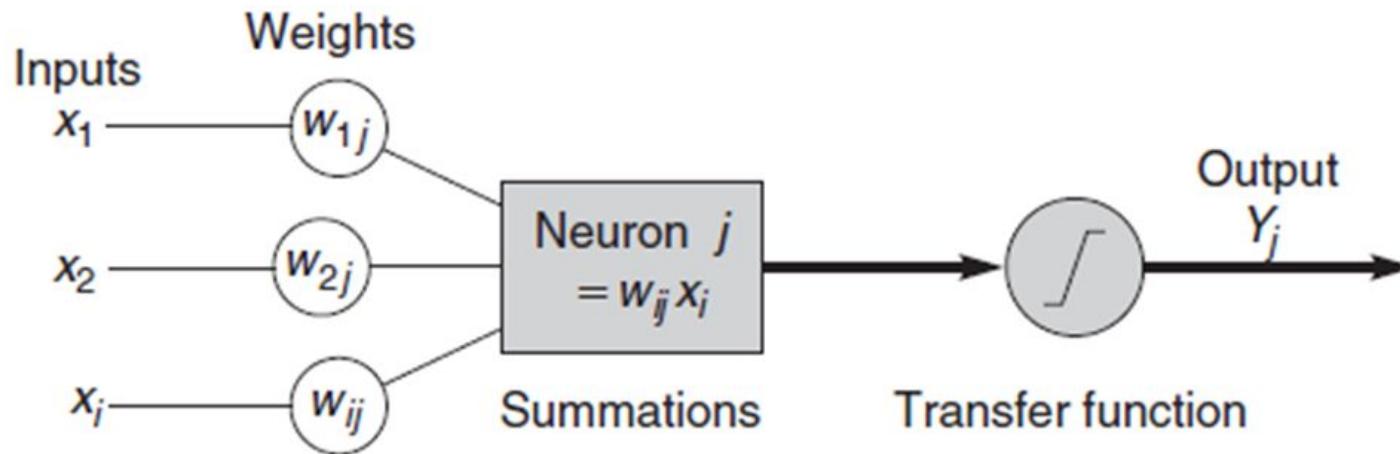
Quá trình xử lý thông tin của ANN

- ❑ Hàm chuyển đổi (transformation function): các neuron này có thể sinh ra một output hoặc không trong hệ thống mạng neuron nhân tạo hay nói cách khác output của một neuron có thể được chuyển đến layer tiếp theo trong mạng neuron hay không.



Quá trình xử lý thông tin của ANN

- ❑ Hàm chuyển đổi (transformation function): Mỗi quan hệ giữa Internal Activation và kết quả (Output) được thể hiện bằng hàm chuyển đổi (transfer function).



Quá trình xử lý thông tin của ANN

- ☐ Hàm chuyển đổi (transformation function): việc lựa chọn hàm chuyển đổi có tác động lớn đến kết quả ANN. Hàm chuyển đổi phi tuyến hay sử dụng trong mạng neuron nhân tạo là sigmoid (logical activation) function.

Summation function:

$$Y = 3(0.2) + 1(0.4) + 2(0.1) = 1.2$$

Transformation (transfer) function: $Y_T = 1/(1 + e^{-1.2}) = 0.77$

$$X_1 = 3 \quad W_1 = 0.2$$

$$X_2 = 1 \quad W_2 = 0.4$$

$$X_3 = 2 \quad W_3 = 0.1$$



$$Y_T = 1/(1 + e^{-Y})$$

Quá trình xử lý thông tin của ANN

- ☐ Hàm chuyển đổi (transformation function): kết quả của sigmoid function thuộc khoảng [0,1] nên còn gọi là hàm chuẩn hóa (normalized function). Kết quả xử lý tại các neuron đôi khi rất lớn. Chính vì thế, hàm chuyển đổi được sử dụng để xử lý những kết quả này trước khi chuyển đến layer tiếp theo.

Summation function:

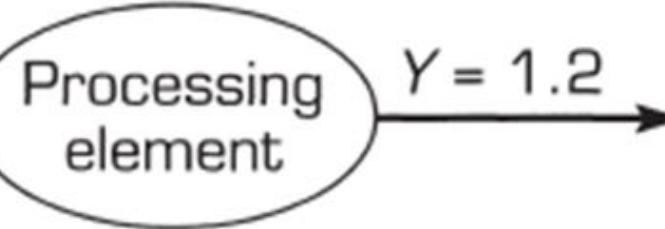
$$Y = 3(0.2) + 1(0.4) + 2(0.1) = 1.2$$

Transformation (transfer) function: $Y_T = 1/(1 + e^{-1.2}) = 0.77$

$$X_1 = 3 \quad W_1 = 0.2$$

$$X_2 = 1 \quad W_2 = 0.4$$

$$X_3 = 2 \quad W_3 = 0.1$$



$$Y_T = 1/(1 + e^{-Y})$$

Quá trình xử lý thông tin của ANN

- Trong thực tế, thay vì sử dụng các hàm chuyển đổi đã nêu trên, người ta có thể sử dụng giá trị ngưỡng (threshold value) để kiểm soát các output của các neuron tại một layer nào đó trước khi chuyển các output này đến các layer tiếp theo. Nếu output của một neuron nào đó nhỏ hơn threshold thì nó sẽ không được chuyển đến layer tiếp theo.

Summation function:

$$Y = 3(0.2) + 1(0.4) + 2(0.1) = 1.2$$

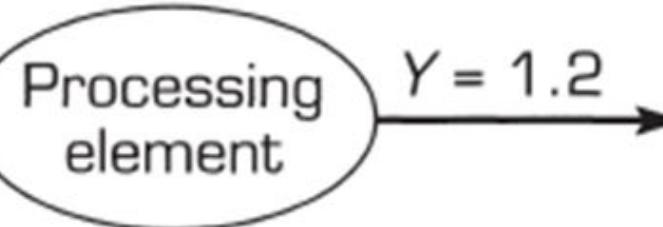
Transformation (transfer) function: $Y_T = 1/(1 + e^{-1.2}) = 0.77$

$$X_1 = 3 \quad W_1 = 0.2$$

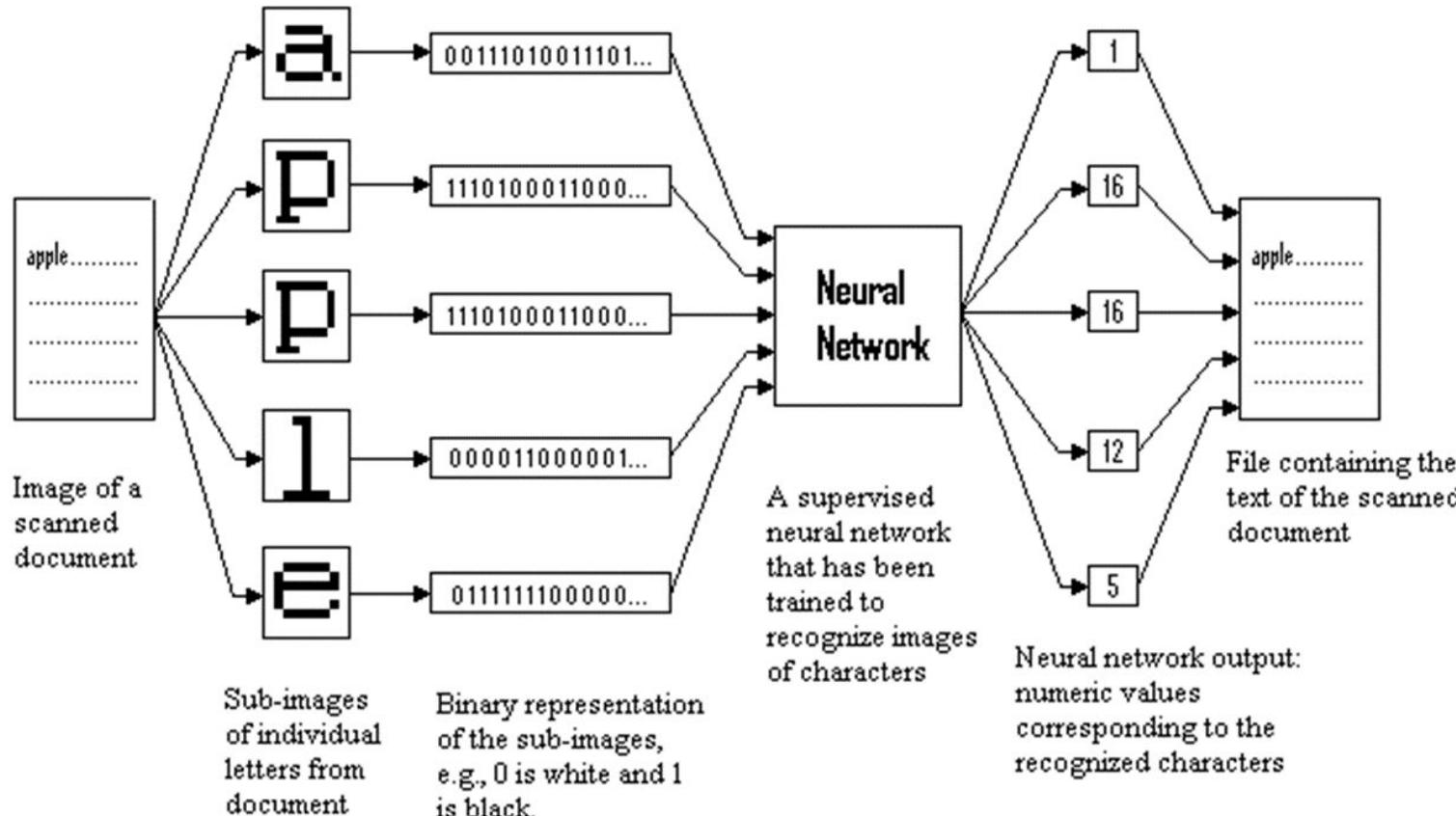
$$X_2 = 1 \quad W_2 = 0.4$$

$$X_3 = 2 \quad W_3 = 0.1$$

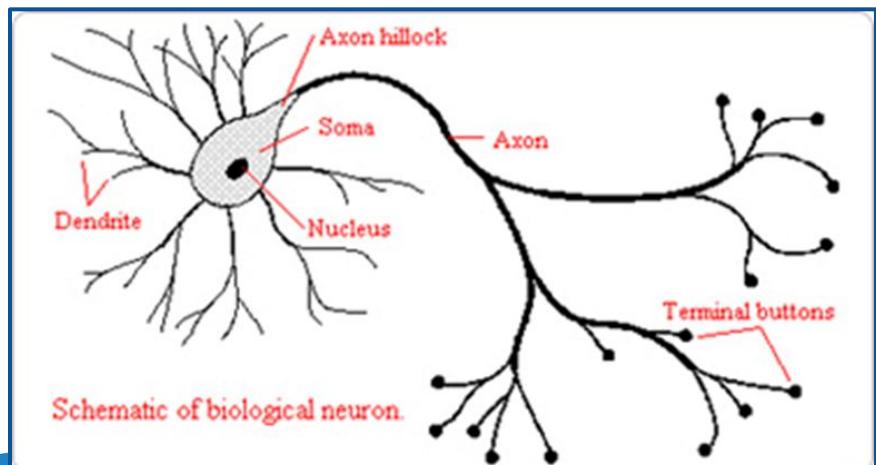
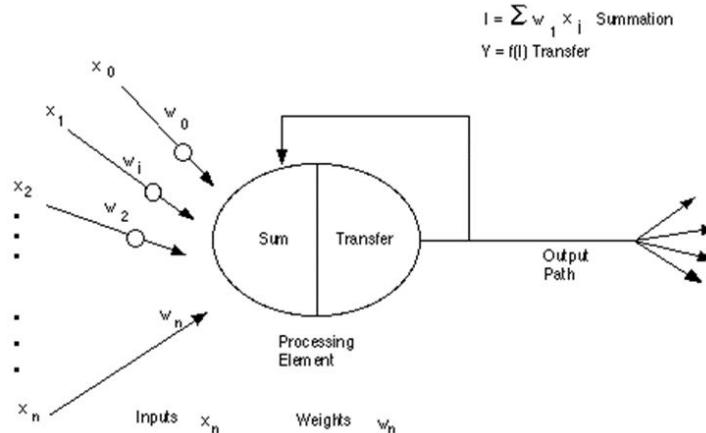
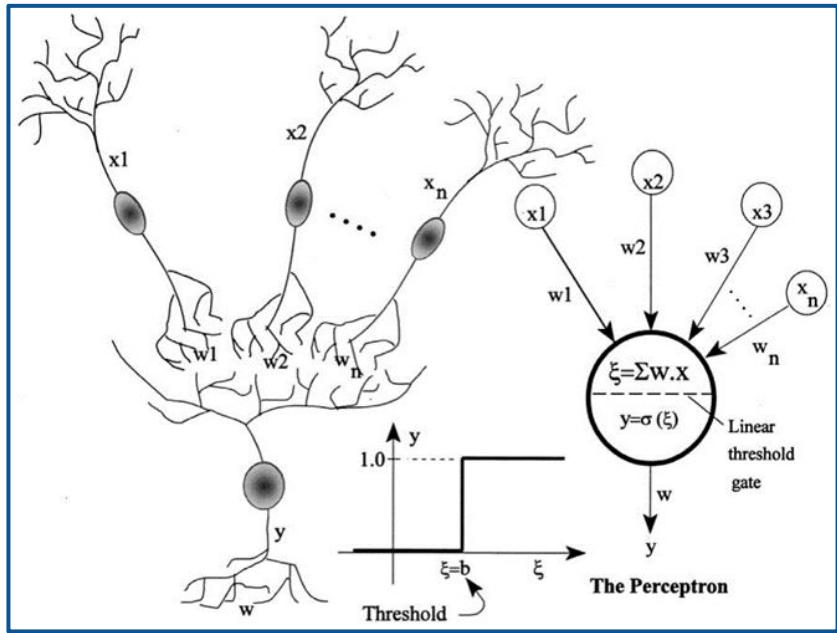
$$Y_T = 1/(1 + e^{-Y})$$



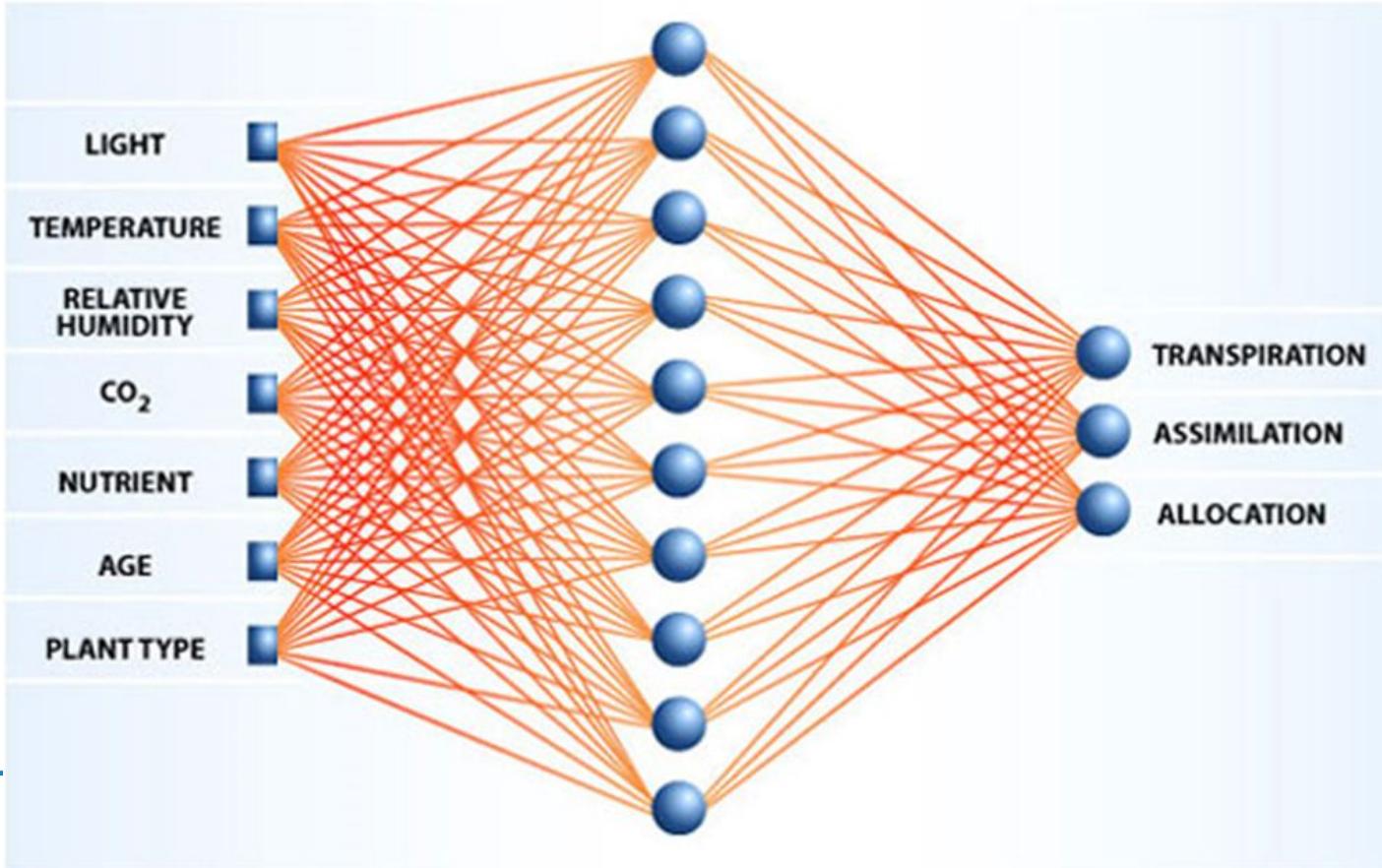
Một số ví dụ



Một số ví dụ



Một số ví dụ

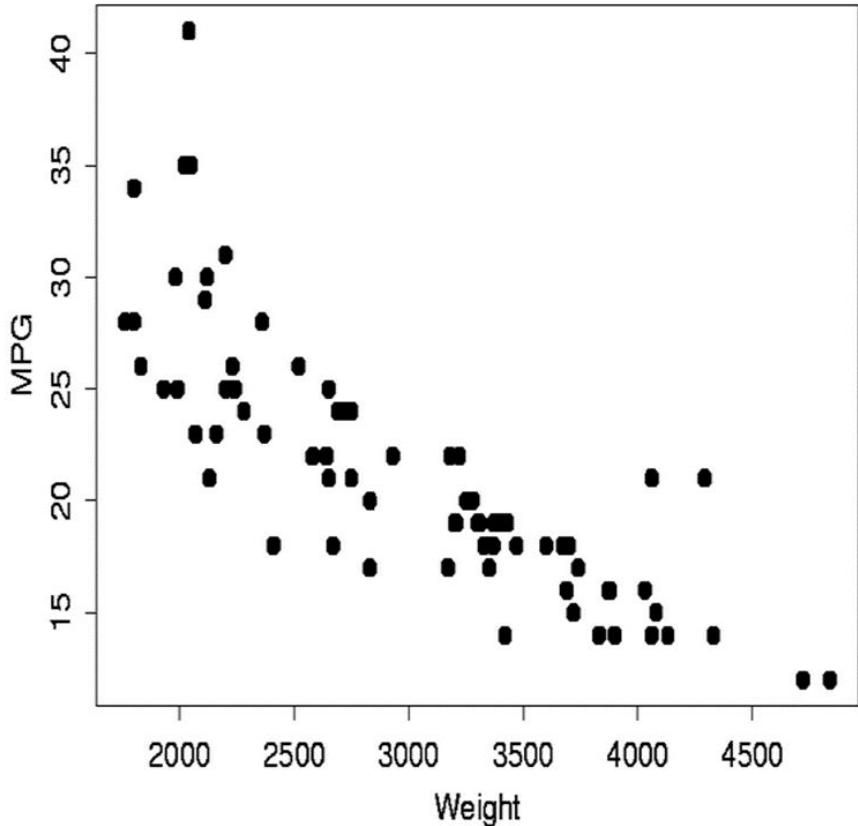


Neural Network



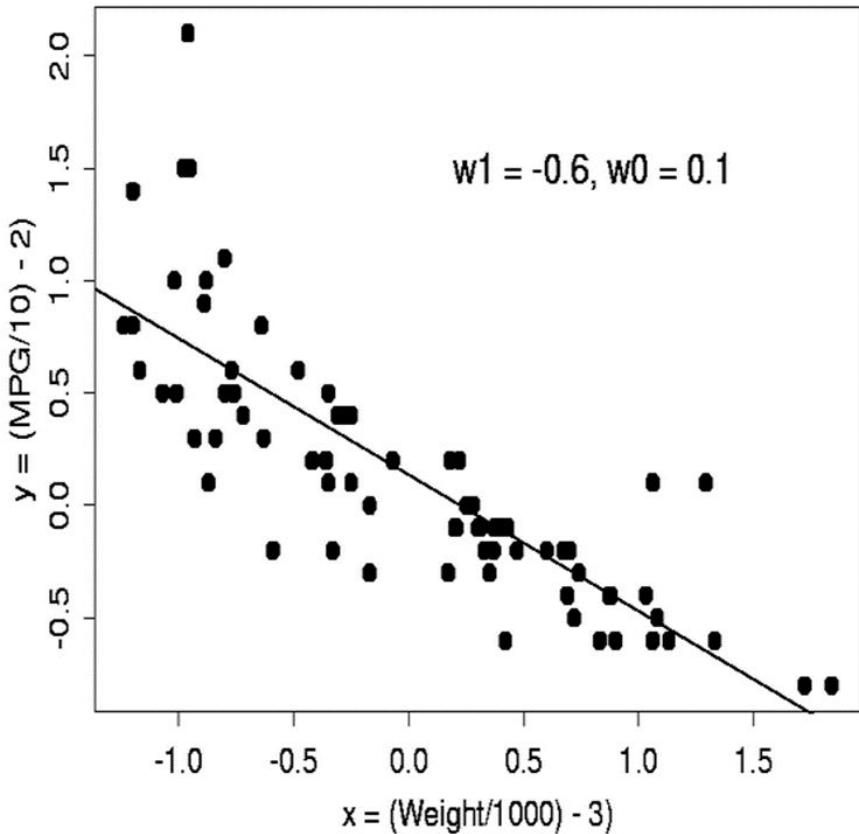
Tổng quan về
neural network

Linear Regression



- ❑ Each dot provides the information about the weight and fuel consumption (mile per gallon) for one of 74 cars.
- ❑ Goal: given the 75th car. How to predict how much fuel it will use.
- ❑ Model: $y=w_1x+w_0$

Minimize the loss function

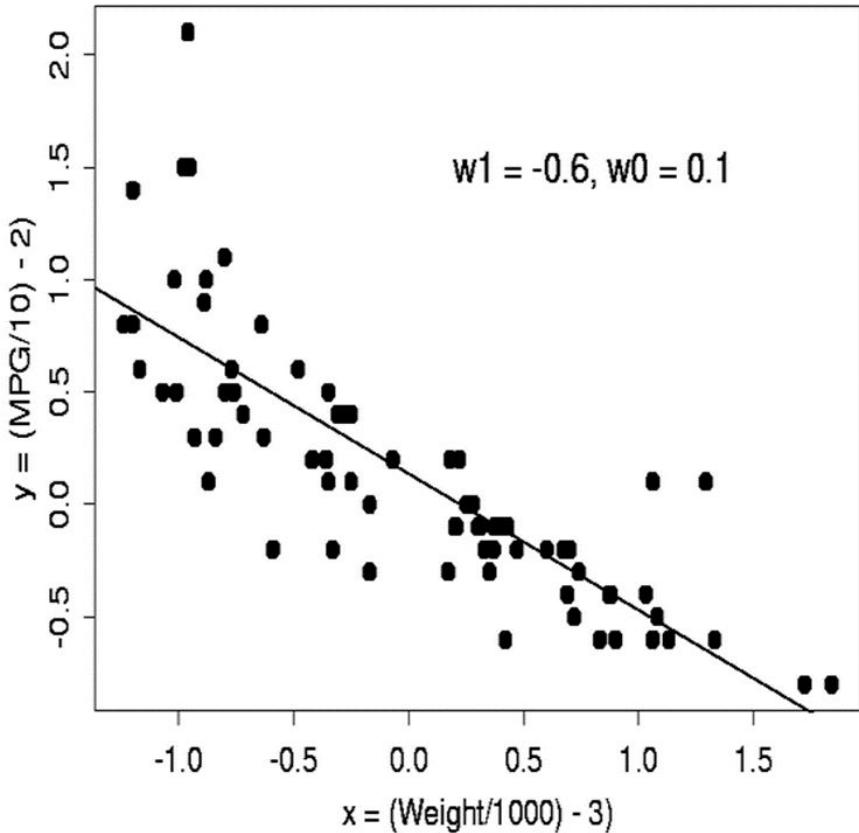


- For linear models, one can use linear regression to minimize the loss function or sum-squared error function:

$$E = \frac{1}{2} \sum_p (t_p - y_p)^2$$

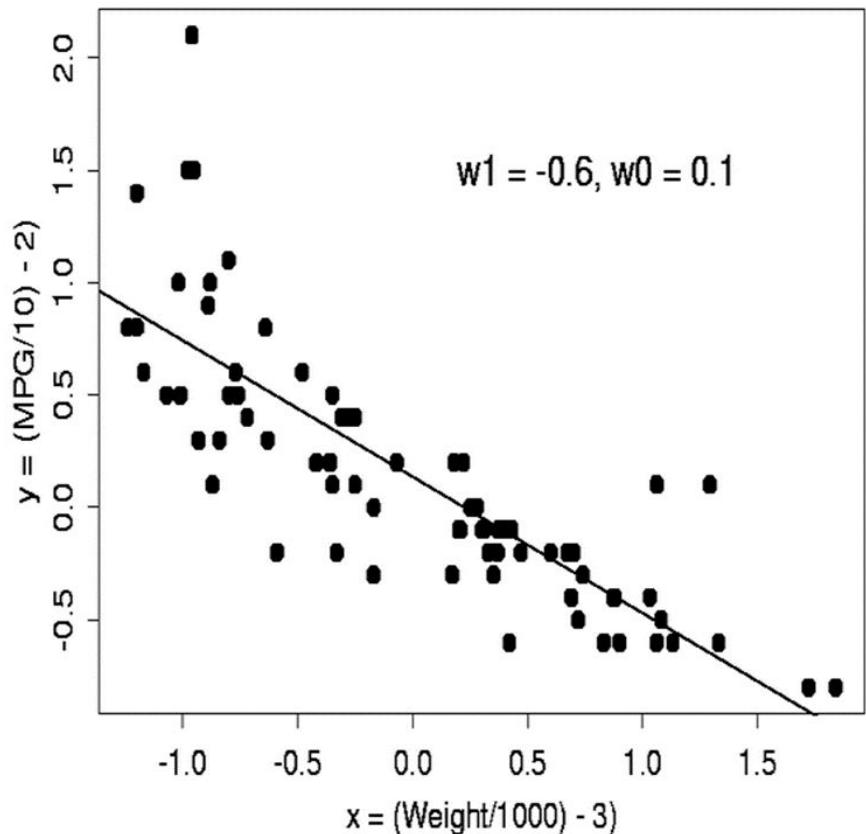
- Here, t_p is a target value (actual fuel consumption and

Minimize the loss function

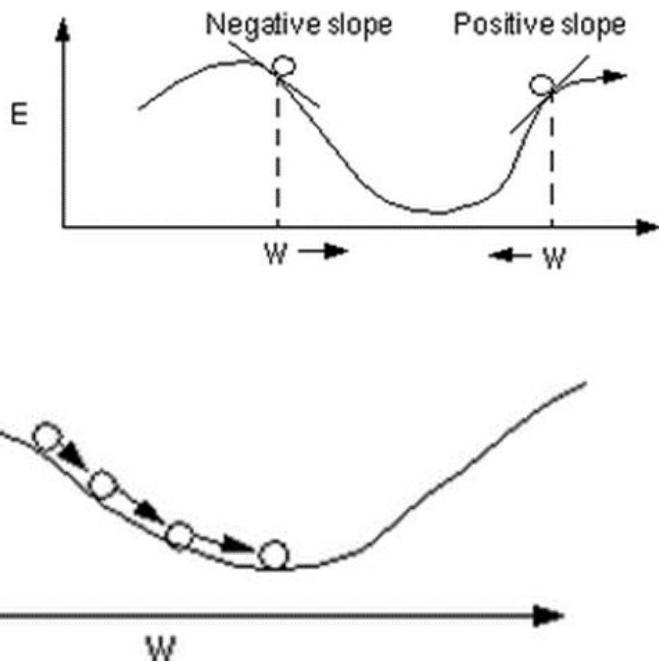


- The linear regression can be solved by an iterative numerical technique, named gradient descent:
 - Step 1: choose some initial values for the model parameters.
 - Step 2: Calculate the gradient G of the loss function with respect to each model parameter.
 - Step 3: change the model parameters so that we move a short distance in the direction of the greatest rate of decrease of the error.
 - Step 4: repeat step 2 and 3 until G gets close to zero.

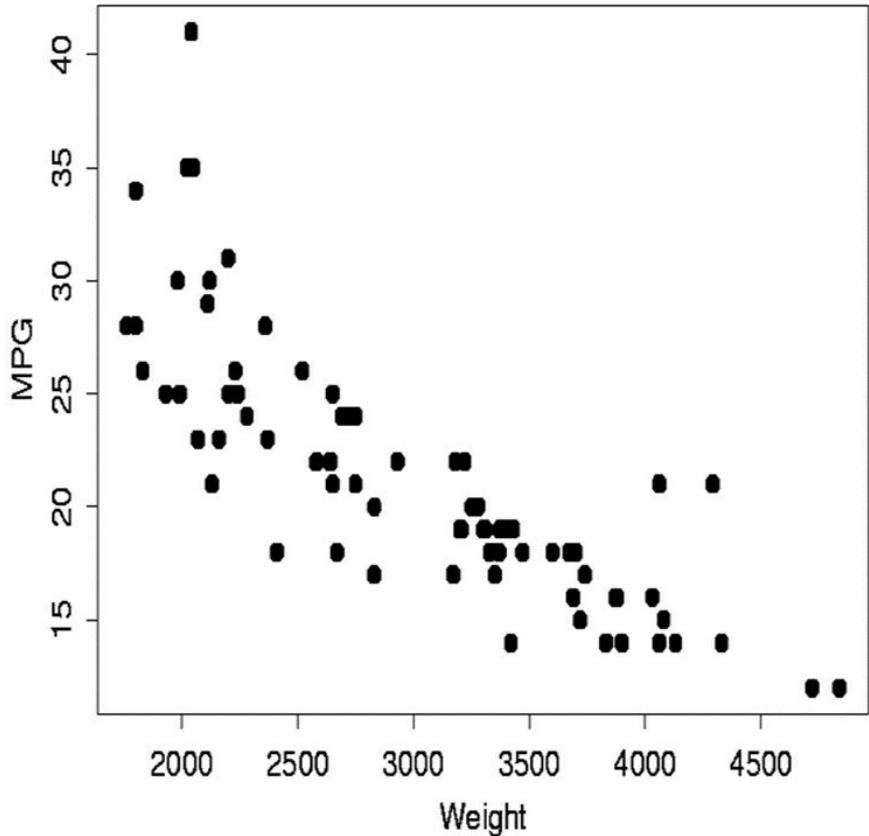
Minimize the loss function



Slope of E positive
=> decrease W
Slope of E negative
=> increase W



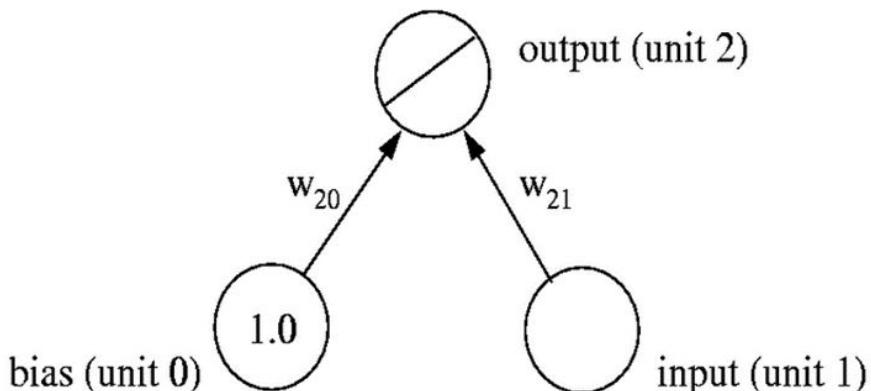
Using neural networks



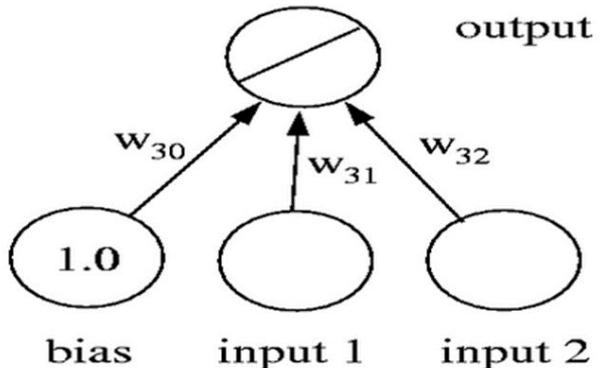
- ❑ Model:

$$y_2 = y_1 w_{21} + 1.0 w_{20}$$

- ❑ Here, it consists of a bias unit, an input unit and a linear output unit:

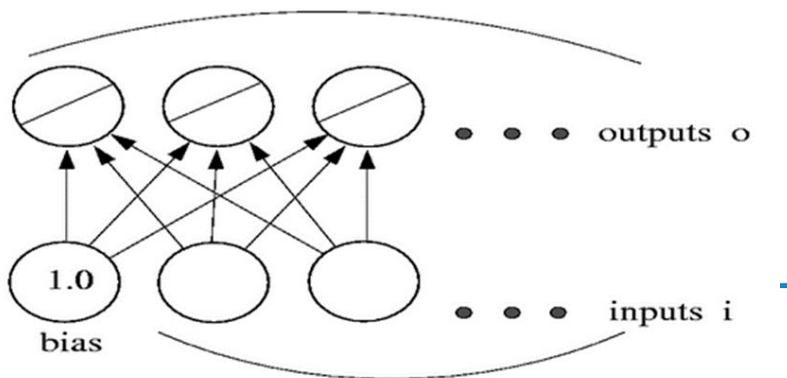


Linear neural networks



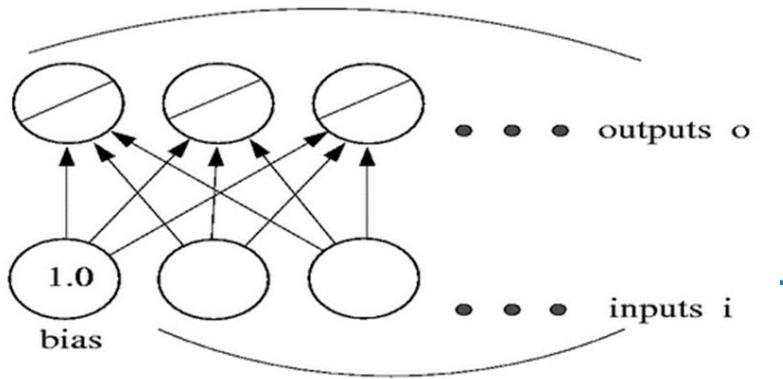
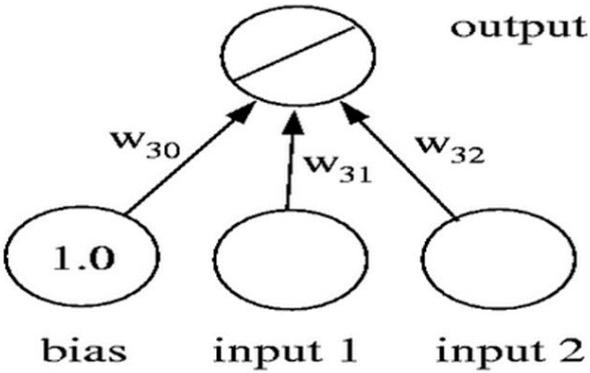
- ❑ The neural network has a typical layered structure: a layer of input units (including the bias), connected by a layer of weights to a layer of output units.
- ❑ The corresponding loss function is:

$$E = \sum_p E^p, E^p = \frac{1}{2} \sum_p (t^{p,0} - y^{p,0})^2$$



for each point p in the training data.

The gradient descent algorithm

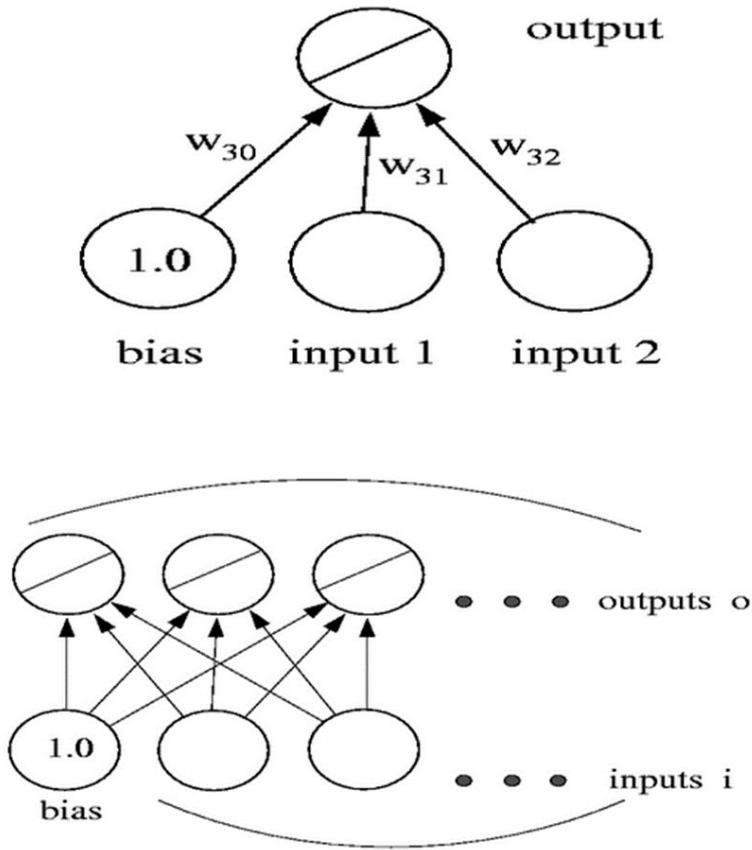


- ❑ Initialize all weights to a small random values.
- ❑ REPEAT until done
 - ❑ For each weight w_{ij} , set $\Delta w_{ij} = 0$.
 - ❑ For each data point $(x,t)^p$:
 - ❑ Set input units to x .
 - ❑ Compute the value of output units.
 - ❑ For each weight w_{ij} , set:

$$\Delta w_{ij} = \Delta w_{ij} + (t_i - y_i) y_j$$
 - ❑ For each weight w_{ij} , set:

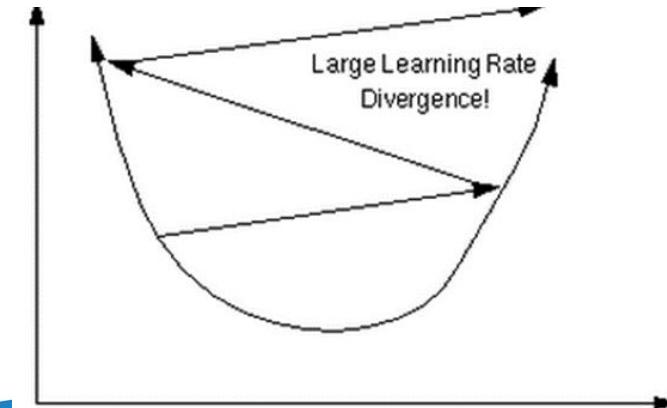
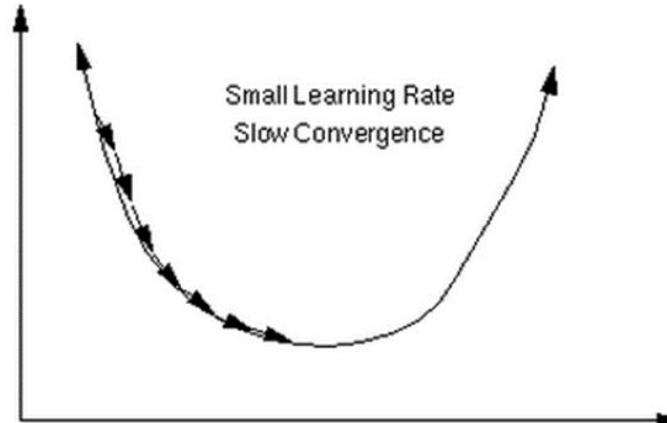
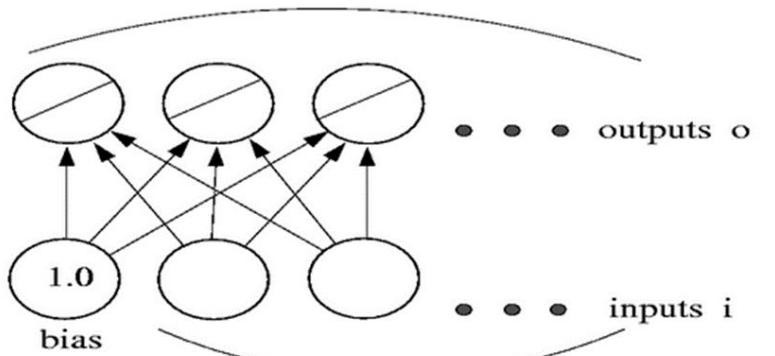
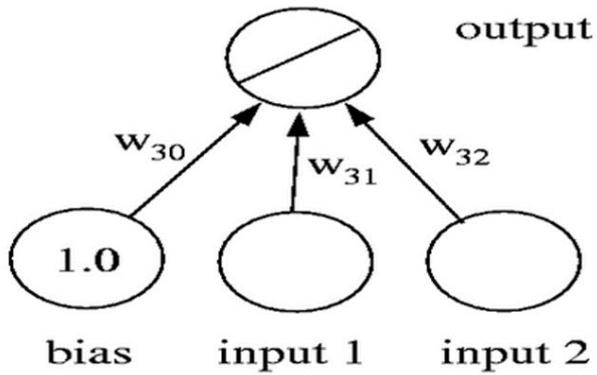
$$w_{ij} = w_{ij} + \mu \Delta w_{ij}$$
- Here μ is the learning rate.

The gradient descent algorithm

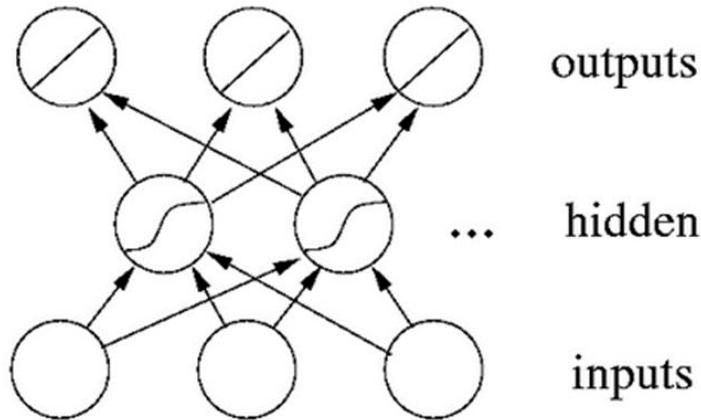


	general case	linear network
Training data	(x, t)	(x, t)
Model parameters	w	w
Model	$y = g(w, x)$	$y_o = \sum_j w_{oj} y_j$
Error function	$E(y, t)$	$E = \sum_p E^p, \quad E^p = \frac{1}{2} \sum_o (t_o^p - y_o^p)^2$
Gradient with respect to w_{ij}	$\frac{\partial E}{\partial w_{ij}}$	$- (t_i - y_i) y_j$
Weight update rule	$\Delta w_{ij} = -\mu \frac{\partial E}{\partial w_{ij}}$	$\Delta w_{oi} = \mu (t_o - y_o) y_i$

The gradient descent algorithm

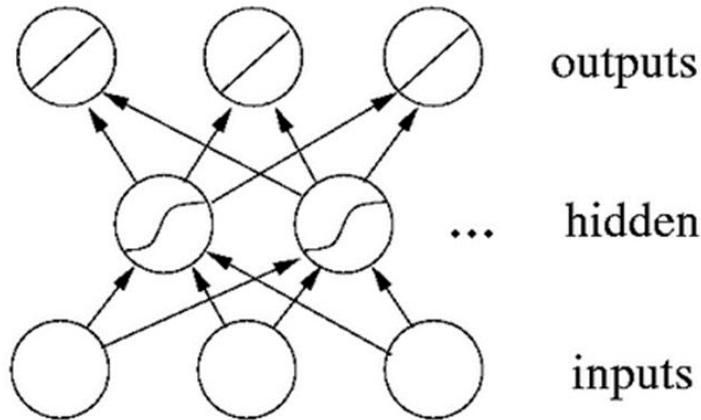


Error back-propagation algorithm



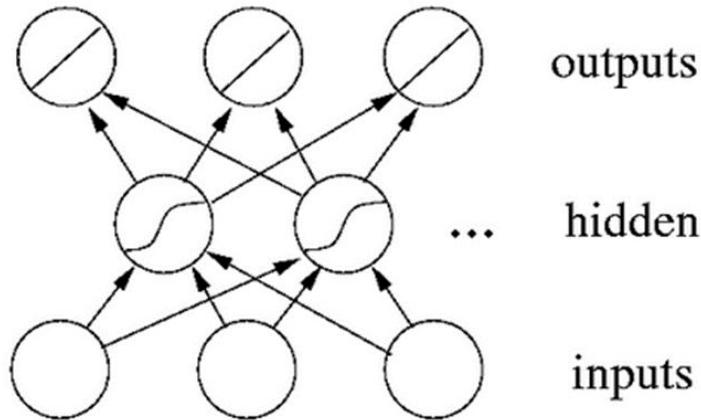
- ❑ We already trained linear networks by gradient descent method.
- ❑ Could we do similarly for multi-layer networks?
- ❑ Difficulties:
 - ❑ Do not have the target values for the hidden units!!!
- ❑ That is an unsolved problems in 1950s.
- ❑ 30 years later, the error back-propagation algorithm was proposed to train hidden units, leading to a new wave of neural network research and applications.

Error back-propagation algorithm



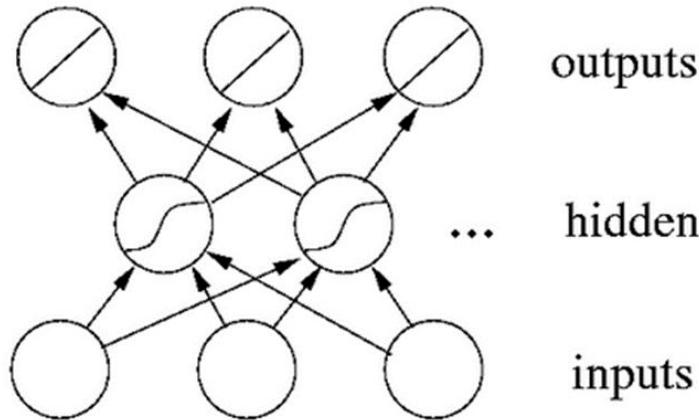
- The algorithm provides a way to train networks with any number of hidden units arranged in any number of layers.
- The network does not have to be organized in layers.
- There must be a way to order the units such that all connections go from “earlier” (closer to the input) to “later” ones (closer to the output).
- Their connection pattern must not contain any cycles.
- The network that respect this constraint are called feed-forward networks and their connection pattern forms a directed acyclic graph.

Error back-propagation algorithm



- ❑ We want to train a multi-layer feed forward network by gradient descent to approximate an unknown function, based on some training data consisting of pairs (x, t) .
- ❑ The vector x represents a pattern input to the network and the vector t the corresponding target (desired output).
- ❑ The overall gradient with respect to the entire training set is just the sum of the gradients for each pattern.

Error back-propagation algorithm



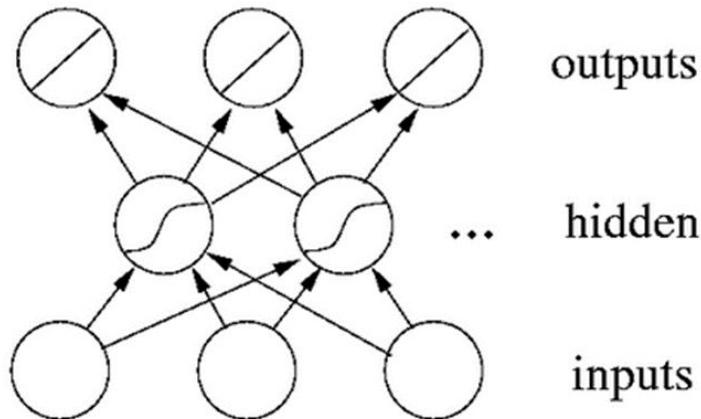
- We will describe how to compute the gradient for just a single training pattern.
- We denote the weight from unit j to unit i by w_{ij} .
- Some definitions:
 - The error signal for unit j:

$$\partial_j = -\partial E / \partial \text{net}_j$$

- The gradient for weight w_{ij} :

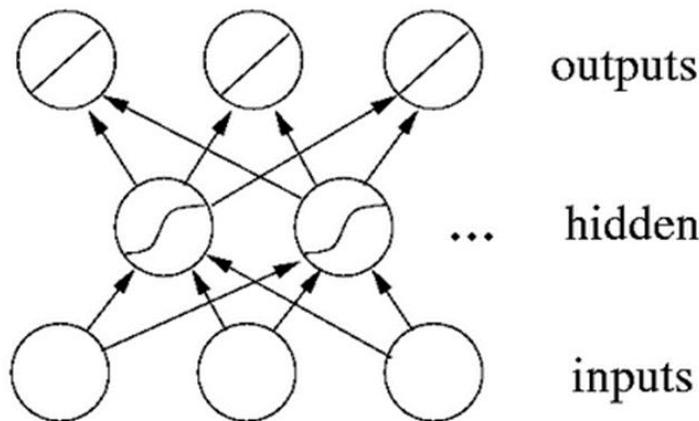
$$\Delta w_{ij} = -\partial E / \partial w_{ij}$$

Error back-propagation algorithm



- ❑ Some definitions:
 - ❑ The set of nodes anterior to unit i:
 $A_i = \{j: \exists w_{ij}\}$
 - ❑ The set of nodes posterior to unit j:
 $P_j = \{i: \exists w_{ij}\}$
 - ❑ The gradient: we expand the gradient into two factors by using the chain rule:
 - ❑ Here:
 - ❑ To compute this gradient, we need to know the activity and the error for all relevant nodes in the network.
- $$\Delta w_{ij} = -\frac{\partial E}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial w_{ij}}$$
- $$\frac{\partial \text{net}_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{k \in A_i} w_{ik} y_k = y_j$$

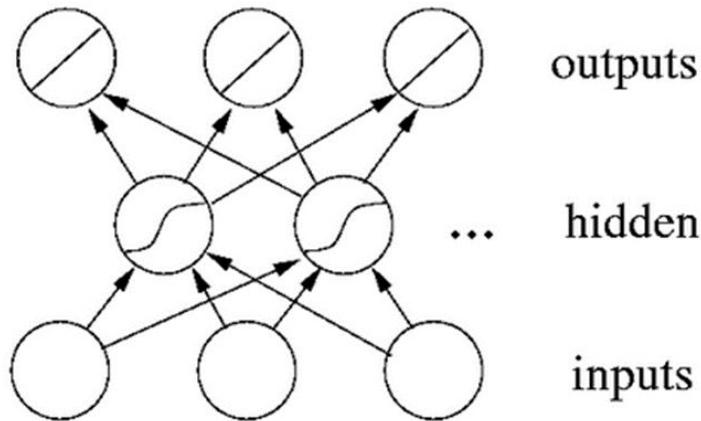
Error back-propagation algorithm



- ❑ Forward activation:
 - ❑ the activity of the input units is determined by the network's external input x .
 - ❑ For all other units, the activity are propagated forward:

$$y_i = f_i(\sum_{j \in A_i} w_{ij} y_j)$$

Error back-propagation algorithm



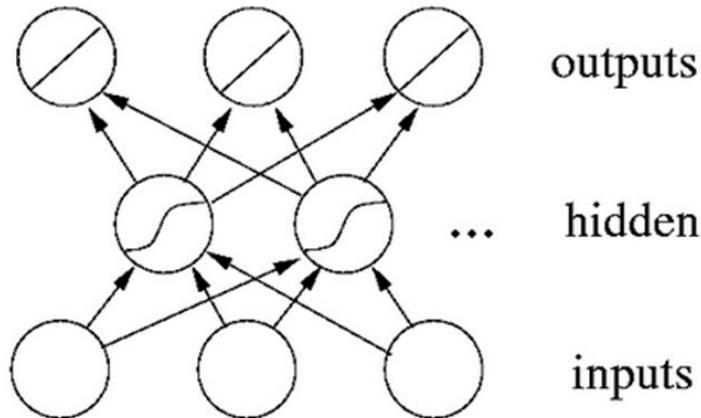
- Calculating output error: assuming that we are using the sum-squared loss function:

$$E = \frac{1}{2} \sum_o (t_o - y_o)^2$$

- The error for the output unit is as following:

$$\delta_o = t_o - y_o$$

Error back-propagation algorithm



- ❑ Error back-propagation: for hidden units, we must propagate the error back from the output nodes. Again, using the chain rule, we can expand the error of a hidden unit in terms of its posterior nodes:

$$\delta_j = \sum_{i \in P_j} \frac{\partial E}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial y_j} \frac{\partial y_j}{\partial \text{net}_j}$$

- ❑ Here: the first is just the error of node i. The second is:

$$\frac{\partial \text{net}_i}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_{k \in A_i} w_{ik} y_k = w_{ij}$$

While the third term is the derivative of node j's activation function:

So: $\delta_j = f'_j(\text{net}_j) \sum_{i \in P_j} \delta_i w_{ij}$

$$\frac{\partial y_j}{\partial \text{net}_j} = f'_j(\text{net}_j)$$

Fuzzy expert systems: Fuzzy logic

- Introduction, or what is fuzzy thinking?
- Fuzzy sets
- Linguistic variables and hedges
- Operations of fuzzy sets
- Fuzzy rules
- Summary

Introduction, or what is fuzzy thinking?

- ❑ Experts rely on **common sense** when they solve problems.
- ❑ **How can we represent expert knowledge that uses vague and ambiguous terms in a computer?**
- ❑ Fuzzy logic is not logic that is fuzzy, but logic that is used to describe fuzziness. Fuzzy logic is the theory of fuzzy sets, sets that calibrate vagueness.
- ❑ Fuzzy logic is based on the idea that all things admit of degrees. Temperature, height, speed, distance, beauty – all come on a sliding scale. The motor is running **really hot**. Tom is a **very tall** guy.

Introduction, or what is fuzzy thinking?

- ❑ Boolean logic uses sharp distinctions. It forces us to draw lines between members of a class and non-members. For instance, we may say, Tom is tall because his height is 181 cm. If we drew a line at 180 cm, we would find that David, who is 179 cm, is small. Is David really a small man or we have just drawn an arbitrary line in the sand?
- ❑ Fuzzy logic reflects how people think. It attempts to model our sense of words, our decision making and our common sense. As a result, it is leading to new, more human, intelligent systems.

Introduction, or what is fuzzy thinking?

- ❑ Fuzzy, or multi-valued logic was introduced **in the 1930s by Jan Lukasiewicz**, a Polish philosopher. While classical logic operates with only two values 1 (true) and 0 (false), Lukasiewicz introduced logic that extended the range of truth values to all real numbers in the interval between 0 and 1. He used a number in this interval to represent the possibility that a given statement was true or false. For example, the possibility that a man 181 cm tall is really tall might be set to a value of 0.86. It is likely that the man is tall. This work led to an inexact reasoning technique often called **possibility theory**.

Introduction, or what is fuzzy thinking?

- Later, in 1937, **Max Black** published a paper called “Vagueness: an exercise in logical analysis”. In this paper, he argued that a continuum implies degrees. Imagine, he said, a line of countless “chairs”. At one end is a Chippendale. Next to it is a near-Chippendale, in fact indistinguishable from the first item. Succeeding “chairs” are less and less chair-like, until the line ends with a log. When does a chair become a log? Max Black stated that if a continuum is discrete, a number can be allocated to each element. He accepted **vagueness as a matter of probability**.

Introduction, or what is fuzzy thinking?

- In 1965 **Lotfi Zadeh**, published his famous paper “Fuzzy sets”. Zadeh extended the work on possibility theory into a formal system of mathematical logic, and introduced a new concept for applying natural language terms. This new logic for representing and manipulating fuzzy terms was called **fuzzy logic**, and Zadeh became the Master of fuzzy logic.

Introduction, or what is fuzzy thinking?

□ Why fuzzy?

As Zadeh said, the term is concrete, immediate and descriptive; we all know what it means. However, many people in the West were repelled by the word fuzzy , because it is usually used in a negative sense.

□ Why logic?

Fuzziness rests on fuzzy set theory, and fuzzy logic is just a small part of that theory.

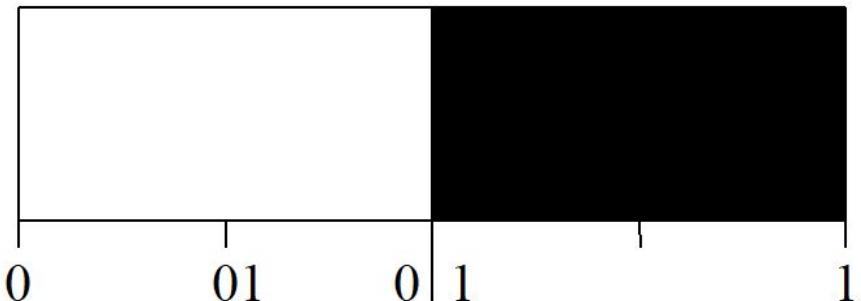
Introduction, or what is fuzzy thinking?

- ❑ Fuzzy logic is a set of mathematical principles for knowledge representation based on degrees of membership.

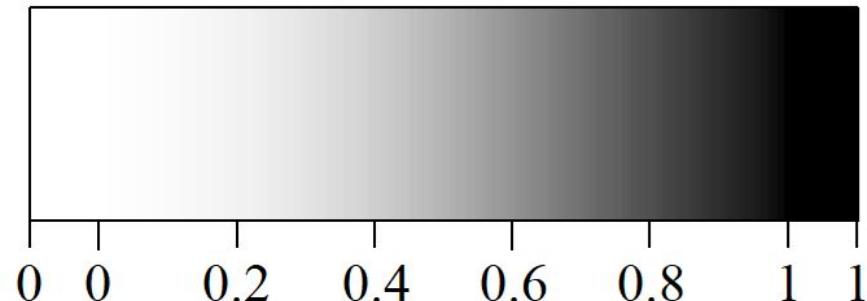
- ❑ Unlike two-valued Boolean logic, fuzzy logic is **multi-valued**. It deals with **degrees of membership** and **degrees of truth**. Fuzzy logic uses the continuum of logical values between 0 (completely false) and 1 (completely true). Instead of just black and white, it employs the spectrum of colours, accepting that things can be partly true and partly false at the same time

Introduction, or what is fuzzy thinking?

❑ Range of logical values in Boolean and fuzzy logic



(a) Boolean Logic.



(b) Multi-valued Logic

Fuzzy sets

- ❑ The concept of a **set** is fundamental to mathematics.
- ❑ However, our own language is also the supreme expression of sets. For example, **car** indicates the **set of cars**. When we say a car , we mean one out of the set of cars.

Fuzzy sets

- The classical example in fuzzy sets is tall men. The elements of the fuzzy set “tall men” are all men, but their degrees of membership depend on their height.

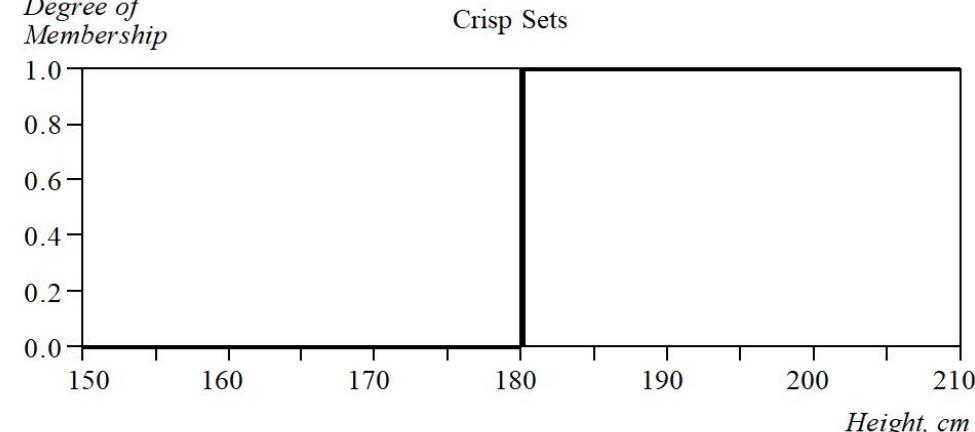
Name	Height, cm	Degree of Membership	
		Crisp	Fuzzy
Chris	208	1	1.00
Mark	205	1	1.00
John	198	1	0.98
Tom	181	1	0.82
David	179	0	0.78
Mike	172	0	0.24
Bob	167	0	0.15
Steven	158	0	0.06
Bill	155	0	0.01
Peter	152	0	0.00

Fuzzy sets

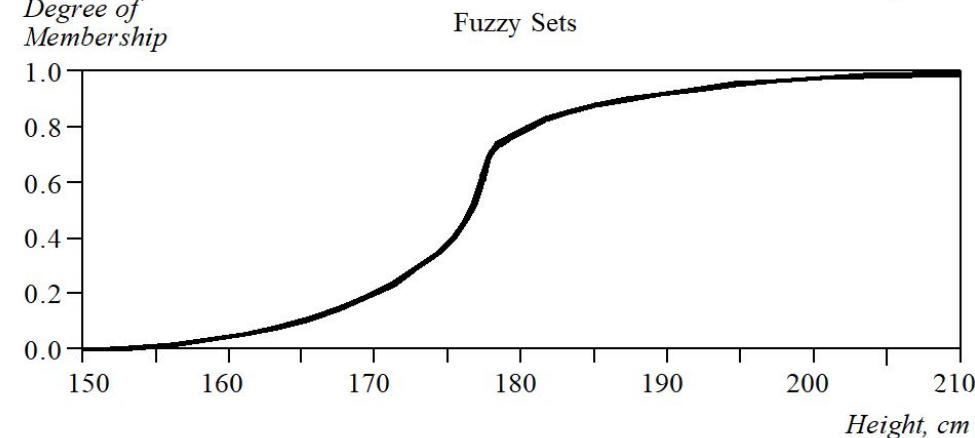
❑ Crisp and fuzzy sets of “tall men”

The x-axis represents the **universe of discourse** – the range of all possible values applicable to a chosen variable. In our case, the variable is the man height. According to this representation, the universe of men's heights consists of all tall men.

Degree of Membership



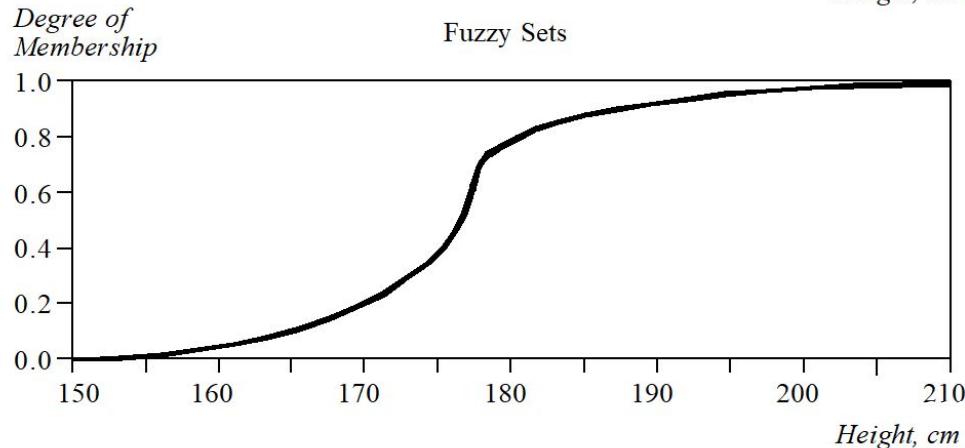
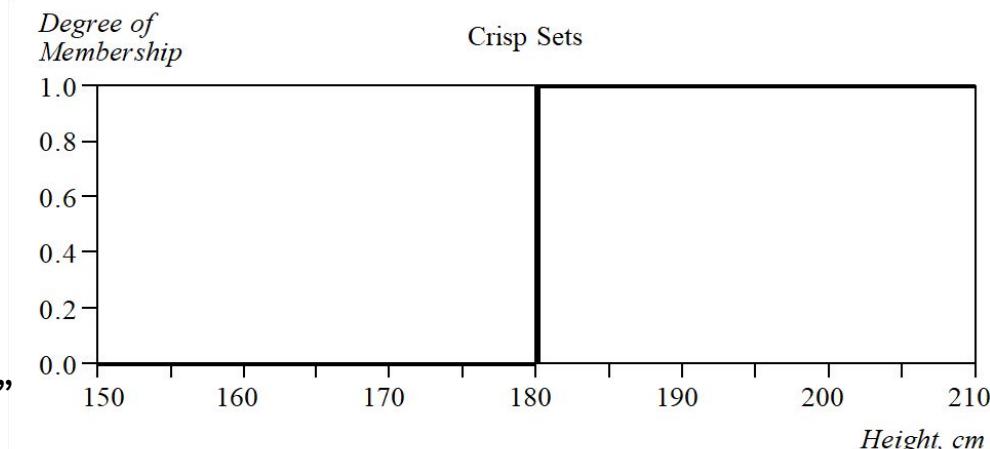
Degree of Membership



Fuzzy sets

- Crisp and fuzzy sets of “tall men”

The y-axis represents the **membership value of the fuzzy set.**
In our case, the fuzzy set of “tall men” maps height values into corresponding membership values.



Fuzzy sets

- ❑ A fuzzy set is a set with fuzzy boundaries.
- ❑ Let X be the universe of discourse and its elements be denoted as x . In the classical set theory, **crisp set A of X is defined as function $f_A(x)$ called the characteristic function of A**
- ❑ $f_A(x): X \rightarrow \{0, 1\}$, where

$$f_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$$

This set maps universe X to a set of two elements. For any element x of universe X , characteristic function $f_A(x)$ is equal to 1 if x is an element of set A , and is equal to 0 if x is not an element of A .

Fuzzy sets

- In the fuzzy theory, fuzzy set A of universe X is defined by function $m_A(x)$ called the **membership function** of set A

$\mu_A(x): X \rightarrow [0, 1]$, where $\mu_A(x) = 1$ if x is totally in A;

$\mu_A(x) = 0$ if x is not in A;

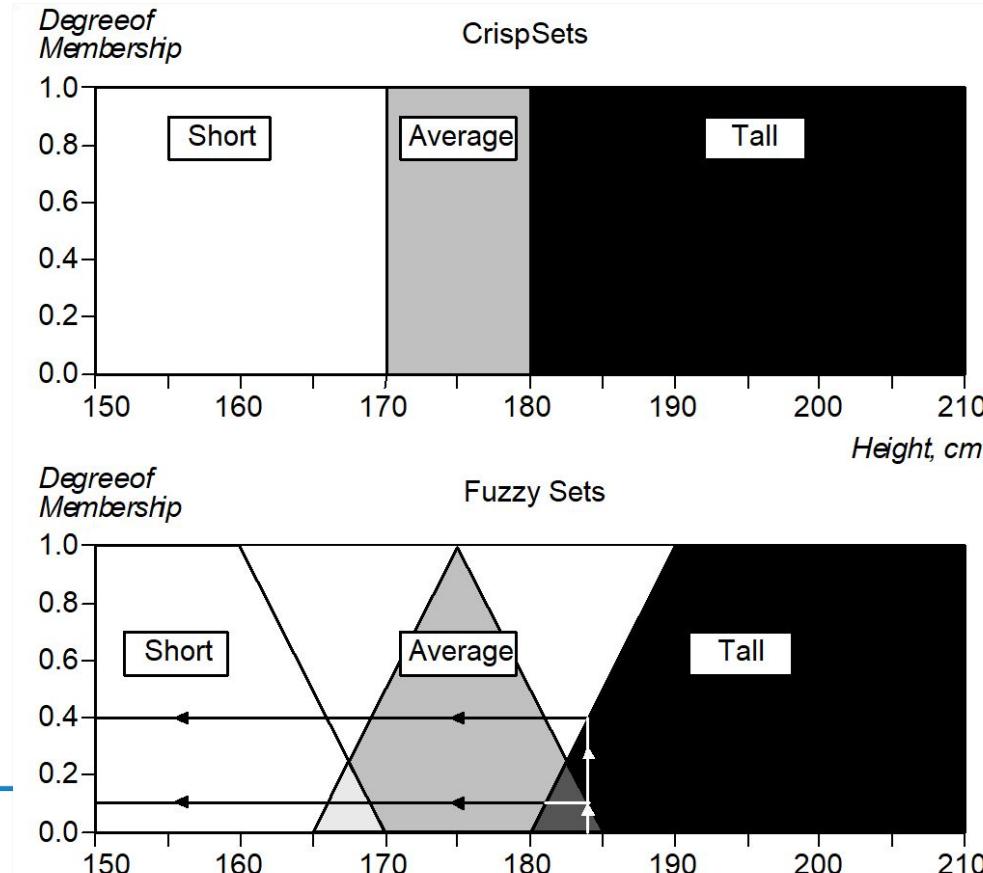
$0 < \mu_A(x) < 1$ if x is partly in A.

This set allows a continuum of possible choices. For any element x of universe X, membership function $m_A(x)$ equals the degree to which x is an element of set A. This degree, a value between 0 and 1, represents the **degree of membership**, also called **membership value**, of element x in set A.

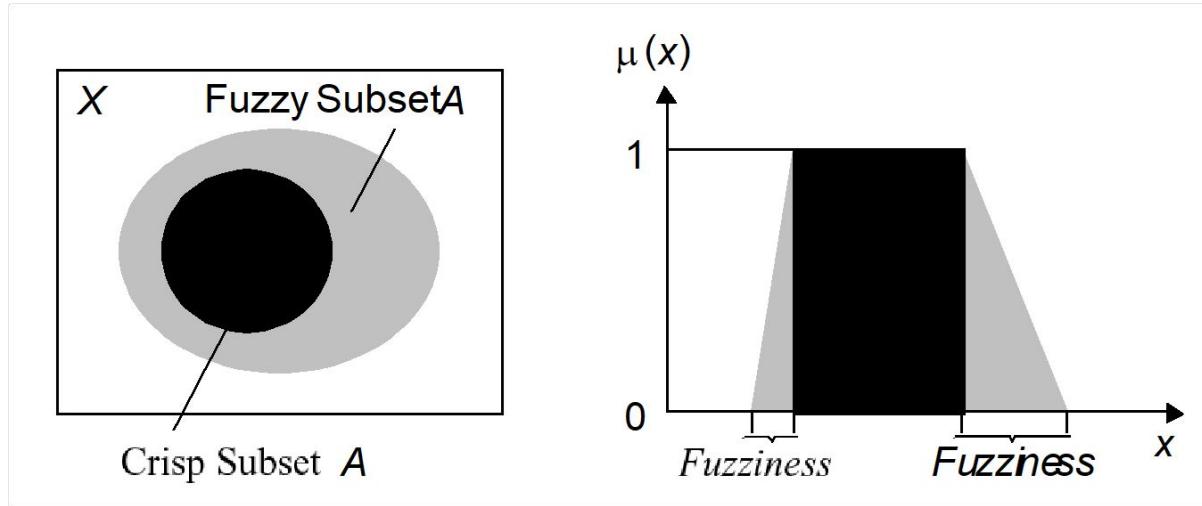
How to represent a fuzzy set in a computer?

- ❑ First, we determine the membership functions. In our “tall men” example, we can obtain fuzzy sets of tall, short and average men.
- ❑ The universe of discourse – the men’s heights – consists of three sets: short, average and tall men. As you will see, a man who is 184 cm tall is a member of the average men set with a degree of membership of 0.1, and at the same time, he is also a member of the tall men set with a degree of 0.4.

Crisp and fuzzy sets of short, average and tall men



Representation of crisp and fuzzy subsets



- Typical functions that can be used to represent a fuzzy set are sigmoid, gaussian and pi. However, these functions increase the time of computation. Therefore, in practice, most applications use **linear fit functions**.

Linguistic variables and hedges

- ❑ In fuzzy expert systems, linguistic variables are used in fuzzy rules. For example:

IF wind is strong
THEN sailing is good

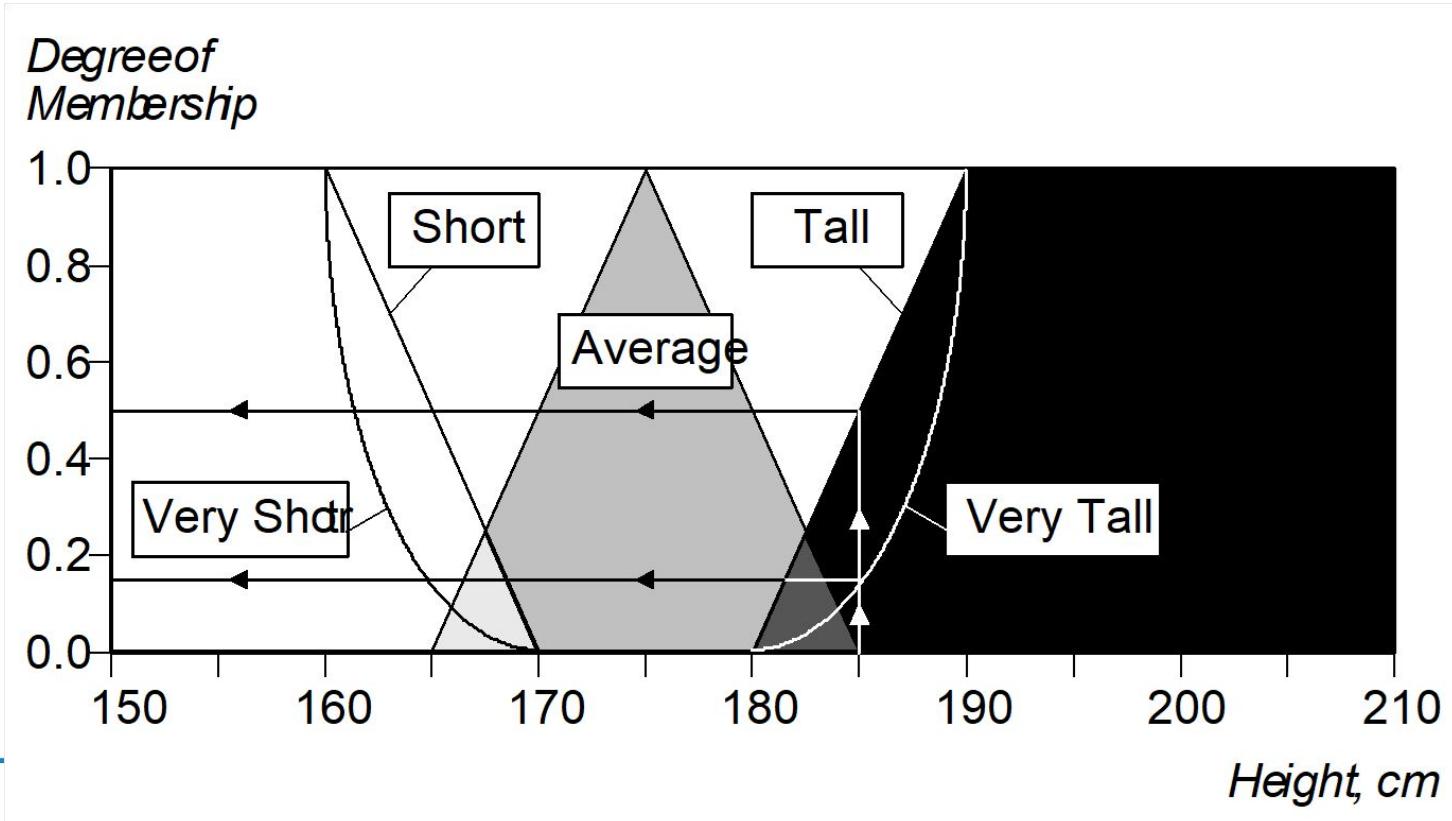
IF project_duration is long
THEN completion_risk is high

IF speed is slow
THEN stopping_distance is short

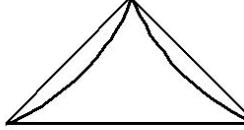
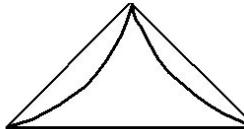
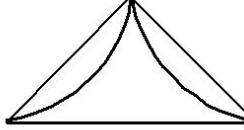
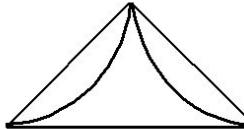
Linguistic variables and hedges

- ❑ The range of possible values of a linguistic variable represents the universe of discourse of that variable. For example, the universe of discourse of the linguistic variable speed might have the range between 0 and 220 km/h and may include such fuzzy subsets as very slow, slow, medium, fast, and very fast.
- ❑ A linguistic variable carries with it the concept of fuzzy set qualifiers, called **hedges**.
- ❑ **Hedges are terms that modify the shape of fuzzy sets. They include adverbs such as very, somewhat, quite, more or less and slightly.**

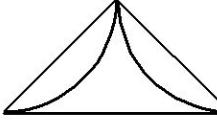
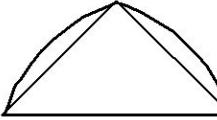
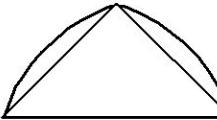
Fuzzy sets with the hedge very



Representation of hedges in fuzzy logic

Hedge	Mathematical Expression	Graphical Representation
A little	$[\mu_A(x)]^{1.3}$	
Slightly	$[\mu_A(x)]^{1.7}$	
Very	$[\mu_A(x)]^2$	
Extremely	$[\mu_A(x)]^3$	

Representation of hedges in fuzzy logic

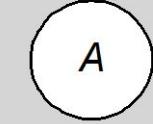
Hedge	Mathematical Expression	Graphical Representation
Very very	$[\mu_A(x)]^4$	
More or less	$\sqrt{\mu_A(x)}$	
Somewhat	$\sqrt{\mu_A(x)}$	
Indeed	$\begin{cases} 2 [\mu_A(x)]^2 & \text{if } 0 \leq \mu_A \leq 0.5 \\ 1 - 2 [1 - \mu_A(x)]^2 & \text{if } 0.5 < \mu_A \leq 1 \end{cases}$	

Operations of fuzzy sets

The classical set theory developed in the late 19th century by Georg Cantor describes how crisp sets can interact. These interactions are called **operations**.

Cantor's sets

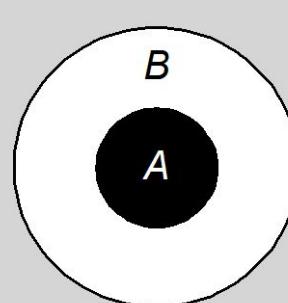
Not A



A

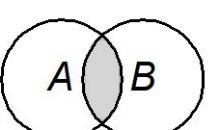
Complement

B



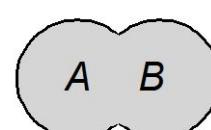
A

Containment



A B

Intersection



A B

Union

Complement

Crisp Sets: Who does not belong to the set?

Fuzzy Sets: How much do elements not belong to the set?

The complement of a set is an opposite of this set. For example, if we have the set of tall men, its complement is the set of NOT tall men. When we remove the tall men set from the universe of discourse, we obtain the complement. If A is the fuzzy set, its complement $\emptyset A$ can be found as follows:

$$\mu_{\neg A}(x) = 1 - \mu_A(x)$$

Containment

Crisp Sets: Which sets belong to which other sets?

Fuzzy Sets: Which sets belong to other sets?

Similar to a Chinese box, a set can contain other sets. The smaller set is called the **subset**. For example, the set of tall men contains all tall men; very tall men is a subset of tall men. However, the tall men set is just a subset of the set of men. In crisp sets, all elements of a subset entirely belong to a larger set. In fuzzy sets, however, each element can belong less to the subset than to the larger set. Elements of the fuzzy subset have smaller memberships in it than in the larger set.

Intersection

Crisp Sets: Which element belongs to both sets?

Fuzzy Sets: How much of the element is in both sets?

In classical set theory, an intersection between two sets contains the elements shared by these sets. For example, the intersection of the set of tall men and the set of fat men is the area where these sets overlap. In fuzzy sets, an element may partly belong to both sets with different memberships. A fuzzy intersection is the lower membership in both sets of each element. The fuzzy intersection of two fuzzy sets A and B on universe of discourse X:

$$\mu_{A \cap B}(x) = \min [\mu_A(x), \mu_B(x)] = \mu_A(x) \cap \mu_B(x),$$

where $x \in X$

Union

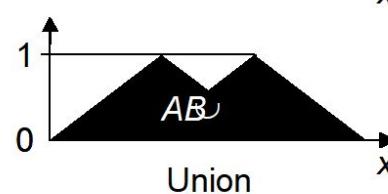
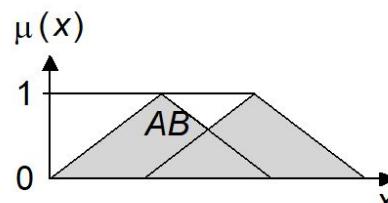
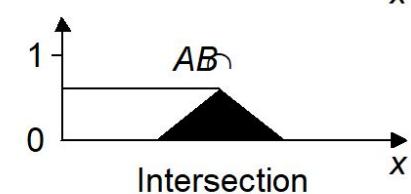
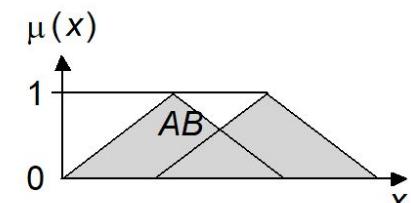
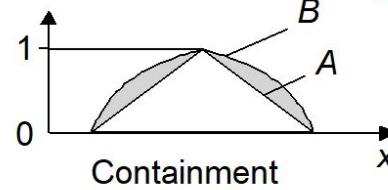
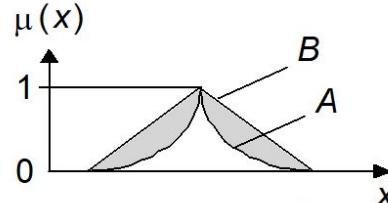
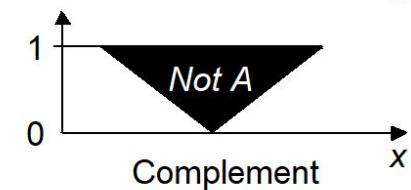
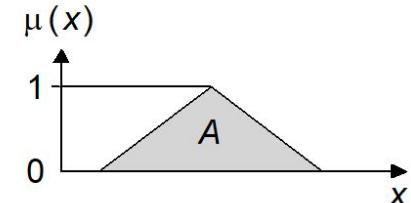
Crisp Sets: Which element belongs to either set?

Fuzzy Sets: How much of the element is in either set?

The union of two crisp sets consists of every element that falls into either set. For example, the union of tall men and fat men contains all men who are tall OR fat. In fuzzy sets, the union is the reverse of the intersection. That is, the union is the largest membership value of the element in either set. The fuzzy operation for forming the union of two fuzzy sets A and B on universe X can be given as:

$$\mu_{A \cup B}(x) = \max [\mu_A(x), \mu_B(x)] = \mu_A(x) \cup \mu_B(x),$$

where $x \in X$



In 1973, **Lotfi Zadeh** published his second most influential paper. This paper outlined a new approach to analysis of complex systems, in which Zadeh suggested capturing human knowledge in fuzzy rules.

A fuzzy rule can be defined as a conditional statement in the form:

**IF x is A
THEN y is B**

where x and y are linguistic variables; and A and B are linguistic values determined by fuzzy sets on the universe of discourses X and Y, respectively.

We can also represent the stopping distance rules in a fuzzy form:

Rule: 1

IF speed is fast

THEN stopping_distance is long

Rule: 2

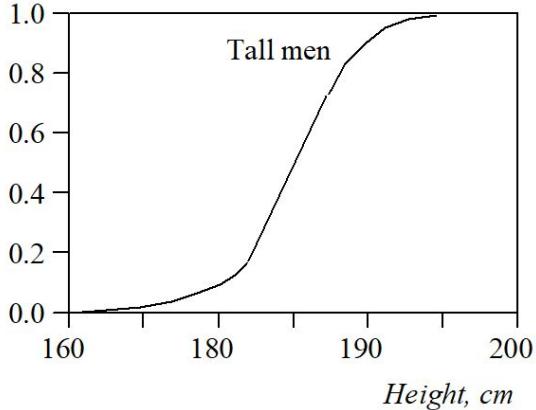
IF speed is slow

THEN stopping_distance is short

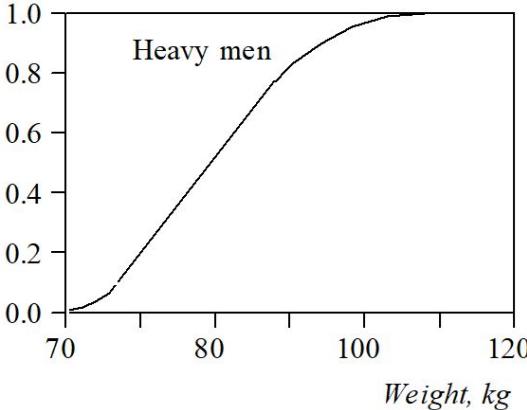
In fuzzy rules, the linguistic variable speed also has the range (the universe of discourse) between 0 and 220 km/h, but this range includes fuzzy sets, such as slow, medium and fast. The universe of discourse of the linguistic variable stopping_distance can be between 0 and 300 m and may include such fuzzy sets as short, medium and long.

- ❑ Fuzzy rules relate fuzzy sets.
- ❑ In a fuzzy system, all rules fire to some extent, or in other words they fire partially. If the antecedent is true to some degree of membership, then the consequent is also true to that same degree.

Degree of Membership



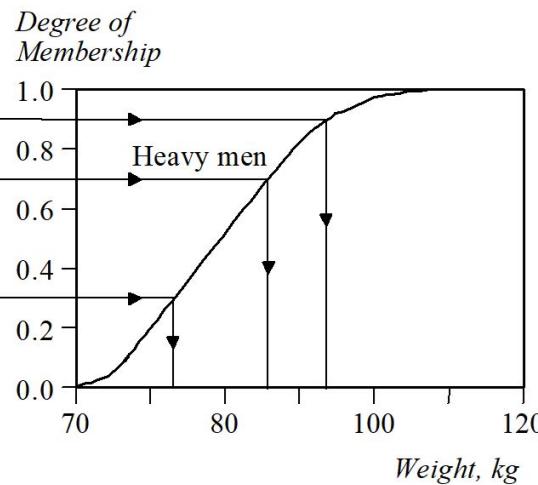
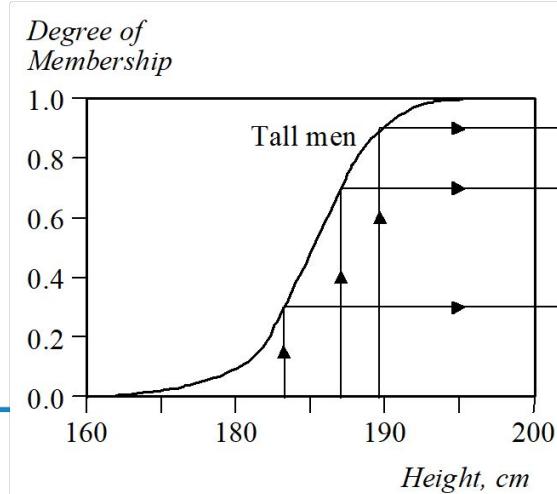
Degree of Membership



- These fuzzy sets provide the basis for a weight estimation model. The model is based on a relationship between a man's height and his weight:

**IF height is tall
THEN weight is heavy**

The value of the output or a truth membership grade of the rule consequent can be estimated directly from a corresponding truth membership grade in the antecedent. This form of fuzzy inference uses a method called **monotonic selection**.



A fuzzy rule can have multiple antecedents, for example:

**IF project_duration is long
AND project_staffing is large
AND project_funding is inadequate
THEN risk is high**

**IF service is excellent
OR food is delicious
THEN tip is generous**

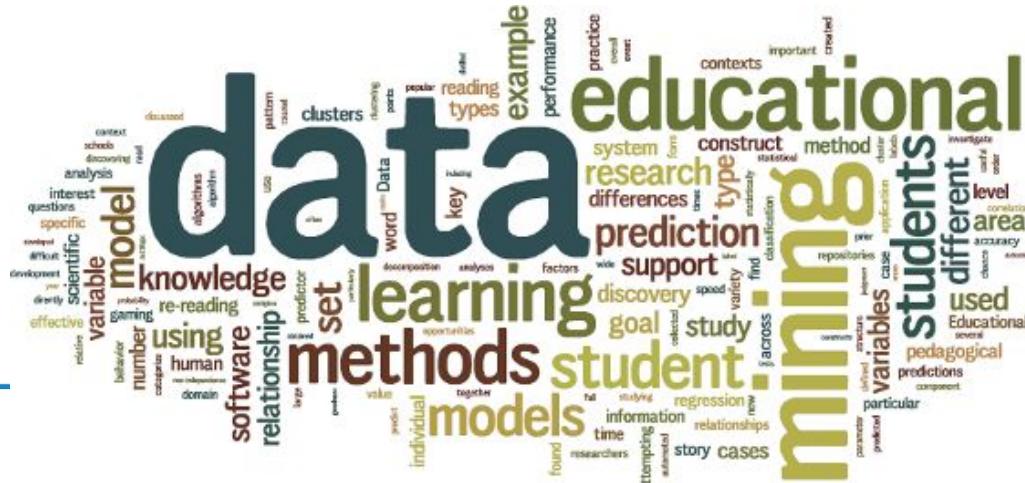
The consequent of a fuzzy rule can also include multiple parts, for instance:

IF **temperature is hot**
THEN **hot_water is reduced;**
 cold_water is increased

Data-mining process



- ❑ (Wikipedia) Extract information from a dataset and transform it into an understandable structure for further use.
- ❑ To discover the patterns and relationships in the data in order to help make better business decisions



❑ Databases today can range in size into the terabytes.

❑ eBay

- 532 nodes cluster (8 * 532 cores, 5.3PB).
- Heavy usage of Java [MapReduce](#), Pig, Hive, HBase
- Using it for Search optimization and Research.



shine_production 16.1806640625 GB

5.3 PB = 5300 TB = 5300.000 GB

❑ Facebook

- We use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning.
- Currently we have 2 major clusters:
 - A 1100-machine cluster with 8800 cores and about 12 PB raw storage.
 - A 300-machine cluster with 2400 cores and about 3 PB raw storage.
 - Each (commodity) node has 8 cores and 12 TB of storage.
 - We are heavy users of both streaming as well as the Java APIs. We have built a higher level data warehousing framework (<http://hadoop.apache.org/hive/>). We have also developed a FUSE implementation over HDFS.



❑ Spotify

- We use Hadoop for content generation, data aggregation, reporting and analysis (see more: [Hadoop at Spotify](#))
- 690 node cluster = 8280 physical cores, 38TB RAM, 28 PB storage
- +7,500 daily Hadoop jobs (scheduled by Luigi, our home-grown and recently open-sourced job scheduler - [code](#) and [video](#))

❑ Within these mass of data lies lots of crucial and hidden information.

❑ **EBay**

- 532 nodes cluster (8 * 532 cores, 5.3PB).
- Heavy usage of Java [MapReduce](#), Pig, Hive, HBase
- Using it for Search optimization and Research.



shine_production 16.1806640625 GB

5.3 PB = 5300 TB = 5300.000 GB

❑ **Facebook**

- We use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning.
- Currently we have 2 major clusters:
 - A 1100-machine cluster with 8800 cores and about 12 PB raw storage.
 - A 300-machine cluster with 2400 cores and about 3 PB raw storage.
 - Each (commodity) node has 8 cores and 12 TB of storage.
 - We are heavy users of both streaming as well as the Java APIs. We have built a higher level data warehousing framework (<http://hadoop.apache.org/hive/>). We have also developed a FUSE implementation over HDFS.



❑ **Spotify**

- We use Hadoop for content generation, data aggregation, reporting and analysis (see more: [❑ Hadoop at Spotify](#))
- 690 node cluster = 8280 physical cores, 38TB RAM, 28 PB storage
- +7,500 daily Hadoop jobs (scheduled by Luigi, our home-grown and recently open-sourced job scheduler - [❑ code](#) and [❑ video](#))







- Market segmentation: identify the common characteristics of customers who buy the same products from our company.



- Customer churn: predict which customers are likely to leave our company and go to the competitor

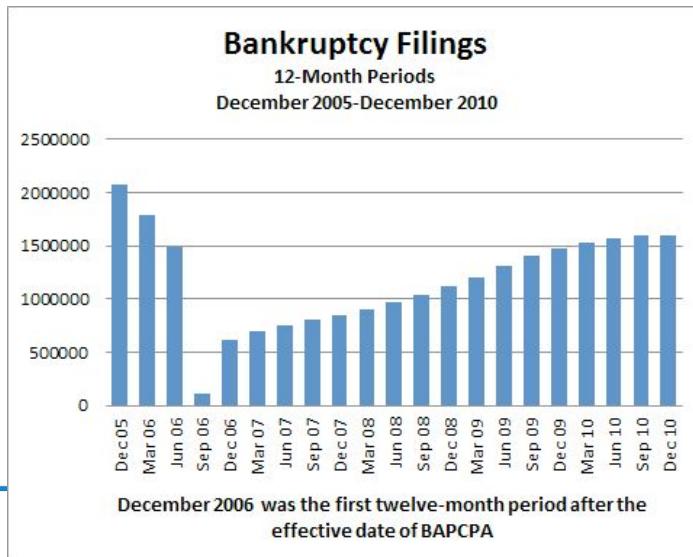


- ❑ Interactive marketing: predict what each individual accessing our Website is mostly likely interested in seeing



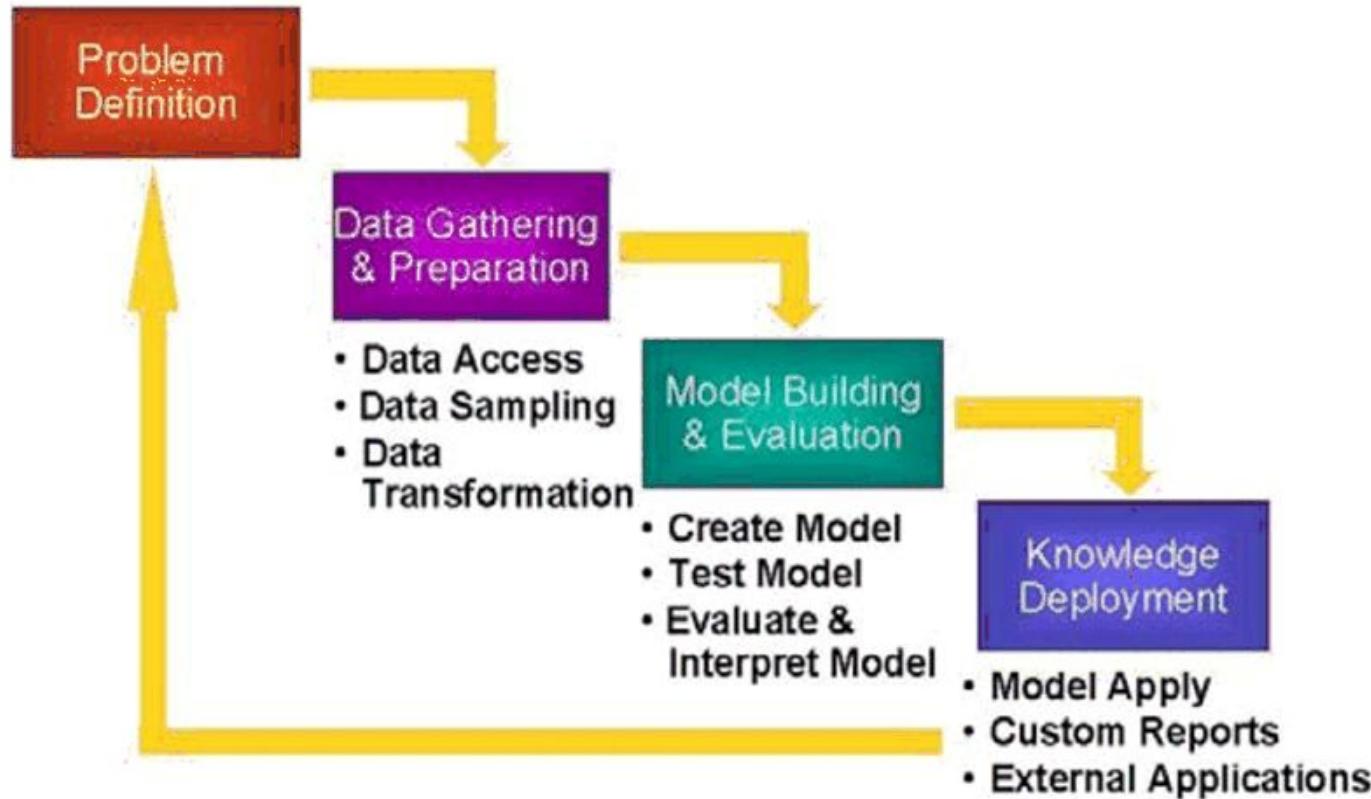
- ❑ Market basket analysis: understand what products or services are commonly purchased together.

- Automated prediction of trends and behaviors: data mining automates the process of finding predictive information in a large database



- ❑ Automated discovery of previously unknown patterns: data mining tools sweep through databases and identify previously hidden patterns.

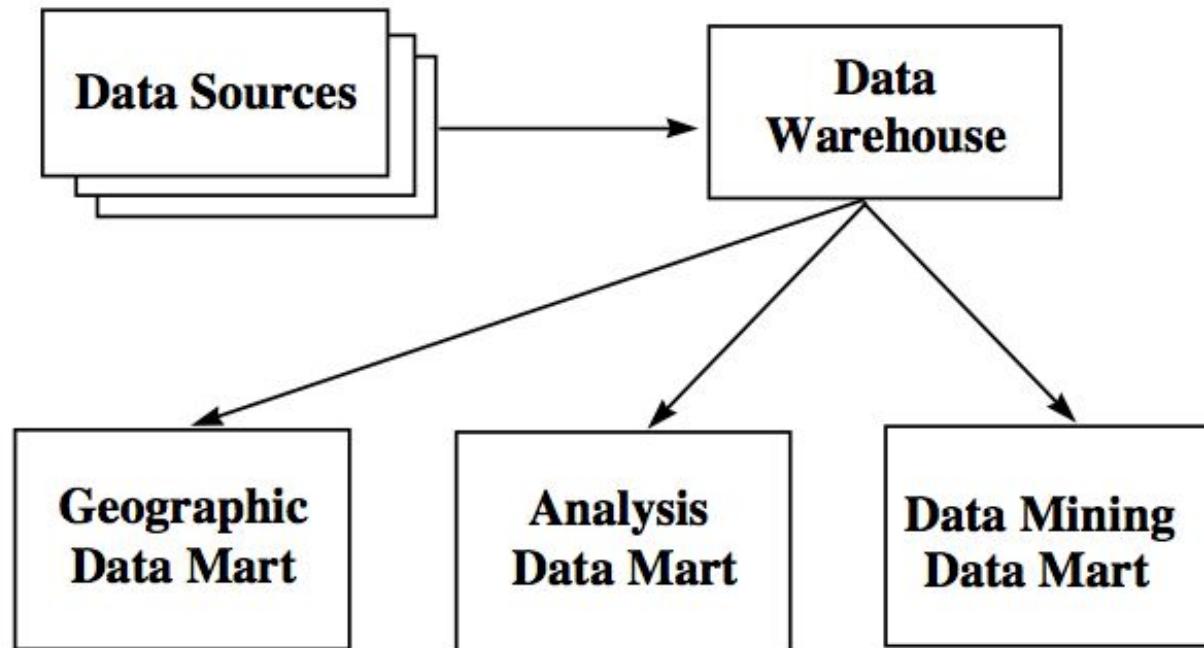
- ❑ For example: analyze the sales data in an company to identify seemingly unrelated products that are often purchased together.





- Understand our data and business
- Clear statements of our objectives.
- Examples: **Problem: increase the response of a direct mail campaign.**
 - Model 1: “increasing the response rate”
 - Model 2: “increase the value of a response”

- ❑ Usually take anywhere from 50% to 90% time and efforts of the entire knowledge discovery process



- ❑ Identify the most important fields in predicting an outcome and determine which derived values may be useful.



- ❑ This is the final data preparation step before building models.
- ❑ Have four main parts:
 - ❑ Select variables.
 - ❑ Select rows.
 - ❑ Construct new variables.
 - ❑ Transform variables.

- ❑ Explore alternative models to find the best one in solving our business problem.
- ❑ Choose a model type for making the prediction.
- ❑ Require a good training and validation protocol (supervised learning) for accurate and robust predictions

- ❑ Evaluate the model and interpret the significance of its results.
- ❑ The accuracy rate applies only to which the model was built.
- ❑ When a data mining model was built and validated, it can be used to recommend actions or to apply the model to various data sets.



