Artificial Intelligence

Prof. Federico Bergenti

Artificial Intelligence Laboratory

www.ailab.unipr.it/bergenti

Class Details

- Class schedule
  - Tuesday 8:30–10:30, room A/1
  - Thursday 8:30–10:30, room A/1
- Classes are live streamed and recorded
  - The course page on Elly (elly2022.smfi.unipr.it) mentions the live-streaming link, which is the same for all classes
  - The course page on Elly mentions the code to register to the team and access recordings
  - *It is strictly prohibited to download recordings*

Communications with the Teacher

- Few simple rules to communicate with the teacher
  - Communications and announcements from the teacher are posted via Elly
  - Meetings with the teacher are requested via e-mail
  - Before requesting a meeting, send an e-mail describing the the reasons for the meeting request
    - Clearly describe the problems with sufficient level of detail
    - Attach the exercises and/or the programs that originated the meeting request

federico.bergenti@unipr.it

Exam Details

- Exam schedule
  - 3 sessions in January–March
  - 3 sessions in June–July
  - 1 session in (late) August–September
- Exam sessions are composed of
  - A written exam and an oral exam
  - The oral exam follows a successful written exam
  - The oral exam is upon explicit request of the teacher or of the student
  - The oral exam is part of the process, and it is not supposed to improve grades only
  - Written and oral exams must be passed in the same session
  - Written and oral exams cover the whole course program

Exam Details

- Exam enrollment
  - Enrollment is strictly needed to participate to an exam
  - Enrollment is performed using Esse3

unipr.esse3.cineca.it

  - Enrollment is possible only within the period associated with each session
  - Clearing the enrollment for an exam is possible only within the period associated with each session
- Students are not supposed to enroll to all sessions

Didactic Materials

- Classes are the official reference for the course
  - The slides used for classes will be available via Elly throughout the academic year
  - Class recordings will be available via Elly throughout the academic year
- Additional materials will be provided via Elly
  - To expand and complement slides
  - To allow for deeper investigations on selected topics
- Further reading
  - S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*, 4th edition, Pearson, 2020

What is Artificial Intelligence?

"In the 1980s, Minsky and Good had shown how neural networks could be generated automatically—self replicated—in accordance with any arbitrary learning program. Artificial brains could be grown by a process strikingly analogous to the development of a human brain."

Arthur C. Clarke, *2001: A Space Odyssey*

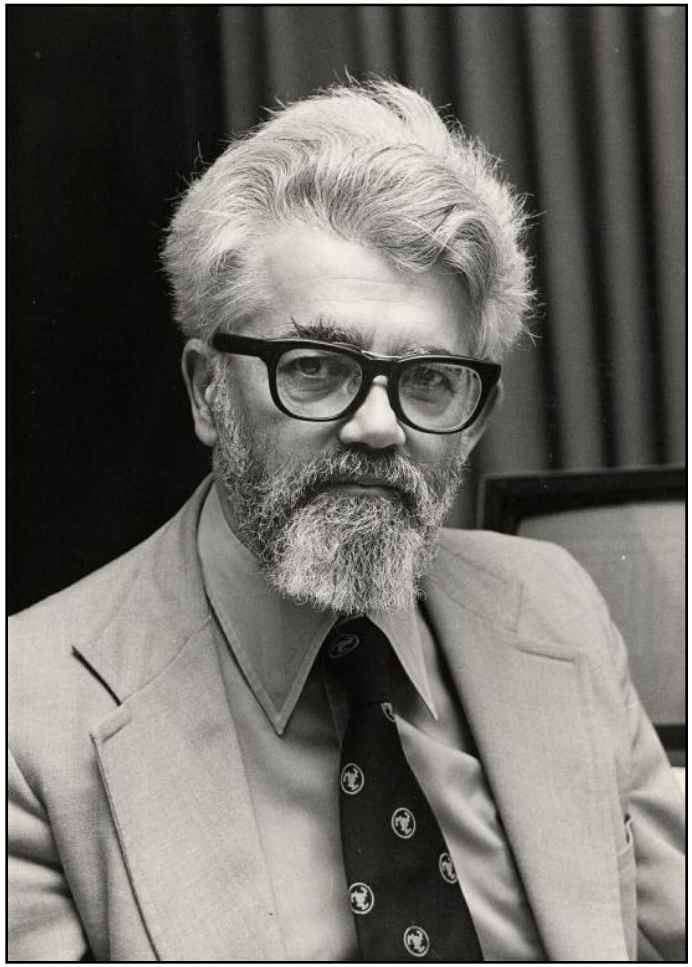
Artificial Intelligence

- An ambitious project that started together with Computer Science
- The goal of the project was to design and develop
  - Machines with intelligent behaviors
  - Machines that can effectively interact with the real world (robots)

- Recently, Artificial Intelligence (AI) often targets
  - Machines that can solve complex problems
  - Machines that exhibit rational behaviors
  - Machines that interact with complex and dynamic worlds (the Internet, the Web, social networks, ...)

The Dartmouth Meeting

- Prof. John McCarthy (1927-2011) invents the name *artificial intelligence* in 1955
  - In a letter to propose a meeting at the Dartmouth College to be held in 1956
- The Dartmouth meeting hosted discussions on the problems that AI have *not yet* solved
  - What is intelligence?
  - What is rationality?
  - Can machine think?
    - If so, how?
    - If not, why not?
  - …



A Characterization of AI

- From a published interview to John McCarthy
  - *Q: What is artificial intelligence?*
  - A: It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.
  - *Q: Yes, but what is intelligence?*
  - A: Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people […]

Other Characterizations

- AI is meant to deal with complex problems that can be (easily) solved by humans and other living beings, but that are not described as algorithms
  - E.g., understand natural languages, invent proofs of theorems, play chess
- Several problems were targeted by AI in the past, but they are no longer considered relevant
  - E.g., compile a Fortran program (~1955), compute the symbolic indefinite integral of a function (~1965), find a known structure in an image (~1970)

What is Easy? What is Hard?

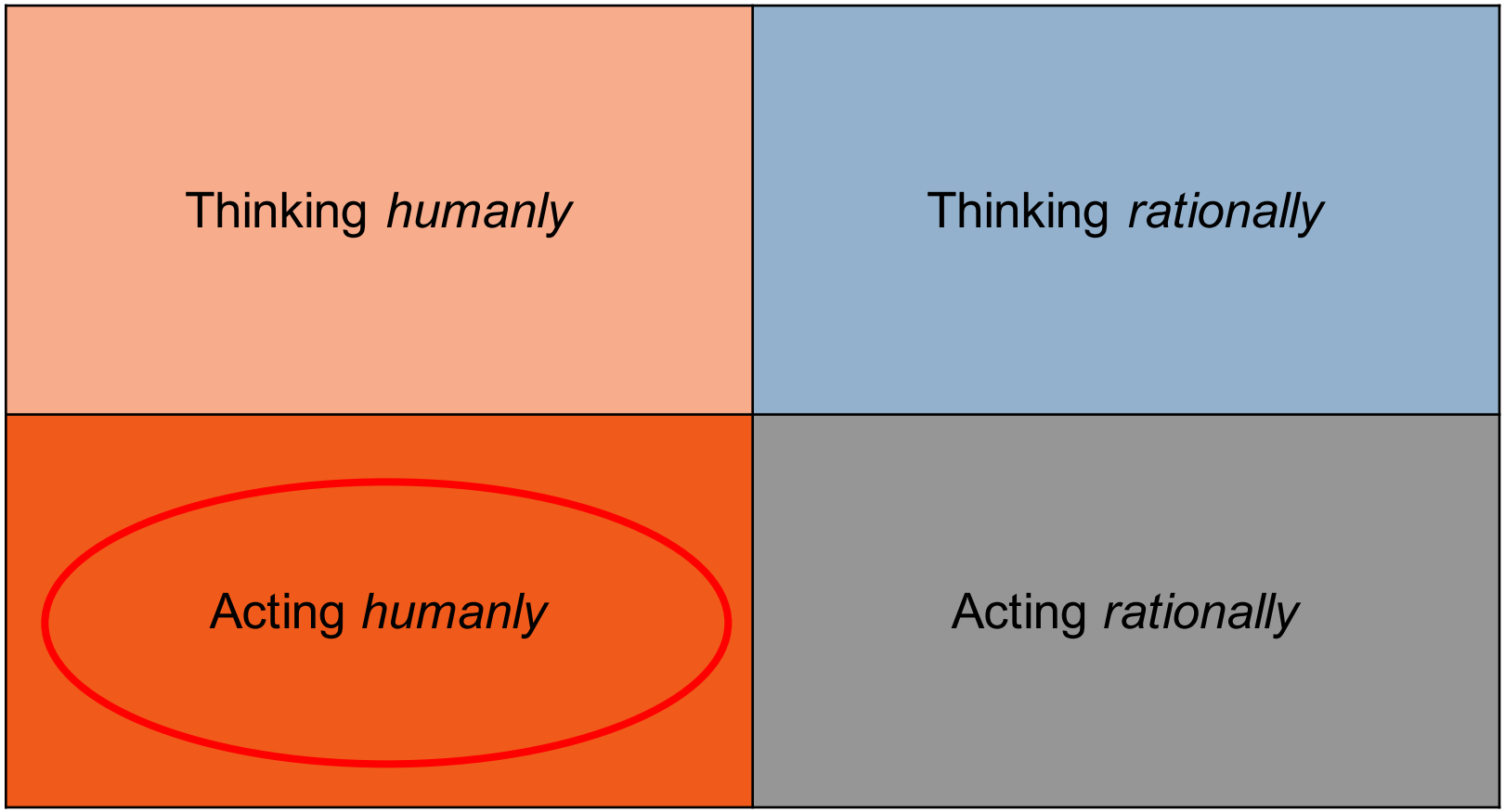
- Several activities that are based on *cognitive capabilities* and that are normally associated with intelligence have already been automatized algorithmically, and they are no longer relevant for AI
- Several activities that are easily and routinely performed by animals have not yet been automatized algorithmically
  - Walk without bumping into obstacles
  - Fuse information from senses (touch, sight, smell, …)
  - Effectively work together without a centralized control (like bees and ants do)

Long Term Goals of AI

- AI can be strong
  - A computer executing an appropriate program can exhibit an intelligence that is indistinguishable from human intelligence
  - The empiricist philosophers Thomas Hobbes stated *"By ratiocination, I mean computation* […] *Ratiocination, therefore, is the same with addition and subtraction."* (Elements of Philosophy, 1656)

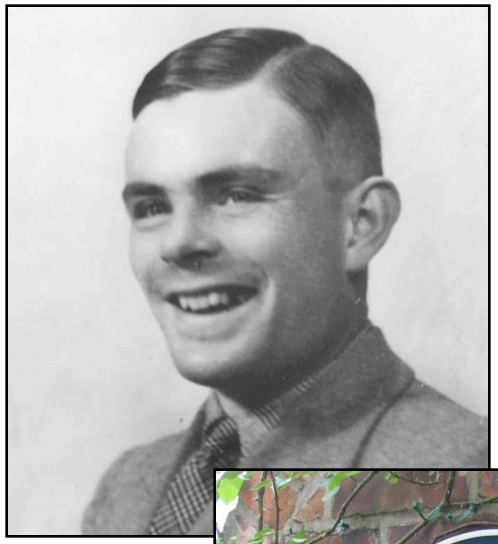
- AI can be weak
  - A computer will never have sufficient resources to exhibit an intelligence that is indistinguishable from human intelligence
  - A computer will never be so complex
    - Computers will simulate selected processes of the human mind, but they will never simulate all needed processes

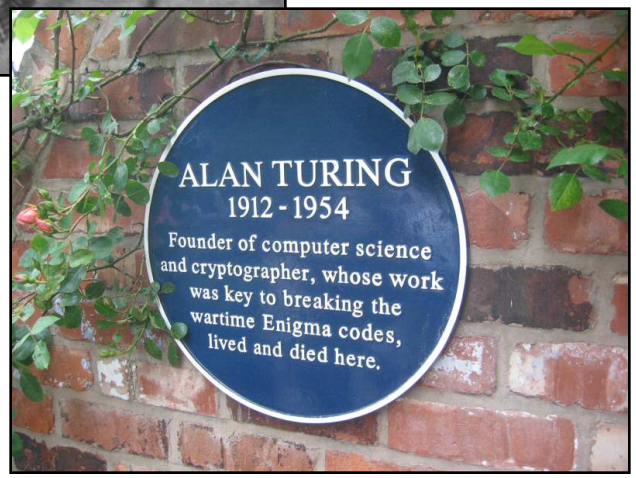
Four Viewpoints



Turing Test (I)

- What are the essential characteristics of intelligent behavior?
- Dr. Alan Turing (1912-1954) proposed an empirical test in 1950





- The Turing test is still relevant to understand the goals of AI

Turing Test (II)

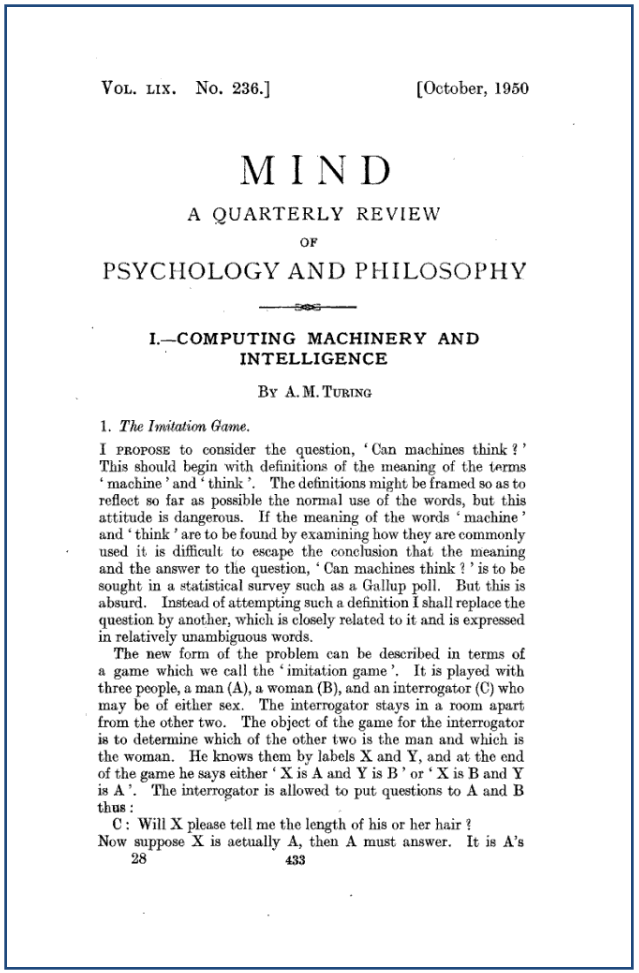
- The article Computing Machinery and Intelligence (Mind, 1950) starts with a section entitled

  The Imitation Game

- The section starts as follows

  I propose to consider the question,
  "Can machines think?"

- The article describes a test to assess if a machine can be considered intelligent
  - It provides an empirical definition of intelligence



VOL. LIX. No. 236.] [October, 1950

MIND

A QUARTERLY REVIEW
OF
PSYCHOLOGY AND PHILOSOPHY

I.—COMPUTING MACHINERY AND INTELLIGENCE

BY A. M. TURING

1. *The Imitation Game.*

I PROPOSE to consider the question, 'Can machines think?' This should begin with definitions of the meaning of the terms 'machine' and 'think'. The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous. If the meaning of the words 'machine' and 'think' are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, 'Can machines think?' is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

The new form of the problem can be described in terms of a game which we call the 'imitation game'. It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart from the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either 'X is A and Y is B' or 'X is B and Y is A'. The interrogator is allowed to put questions to A and B thus:

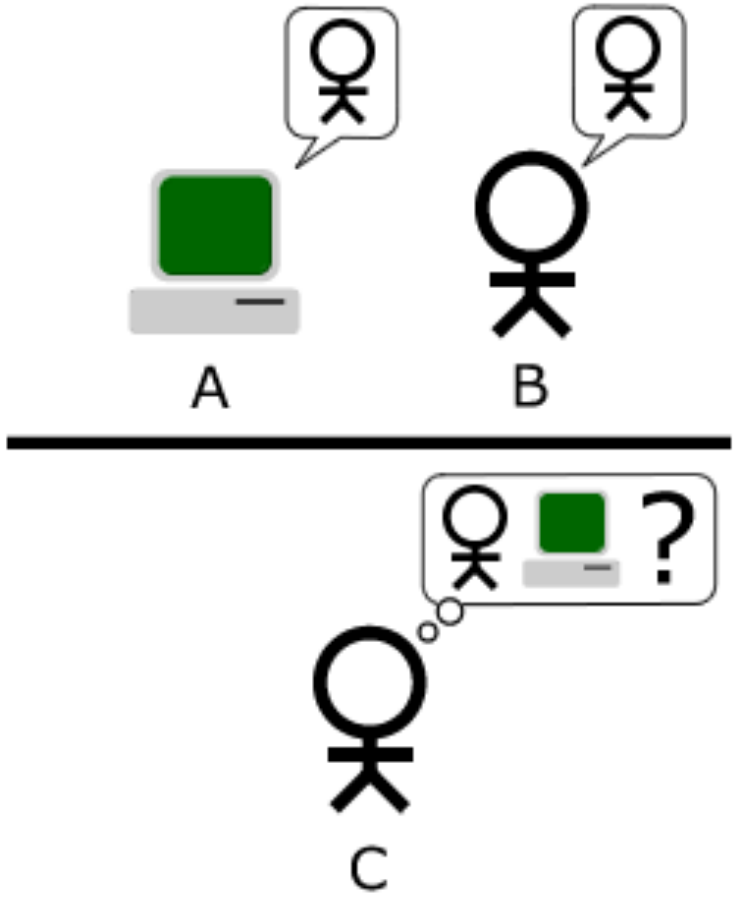
C: Will X please tell me the length of his or her hair?

Now suppose X is actually A, then A must answer. It is A's

28 433

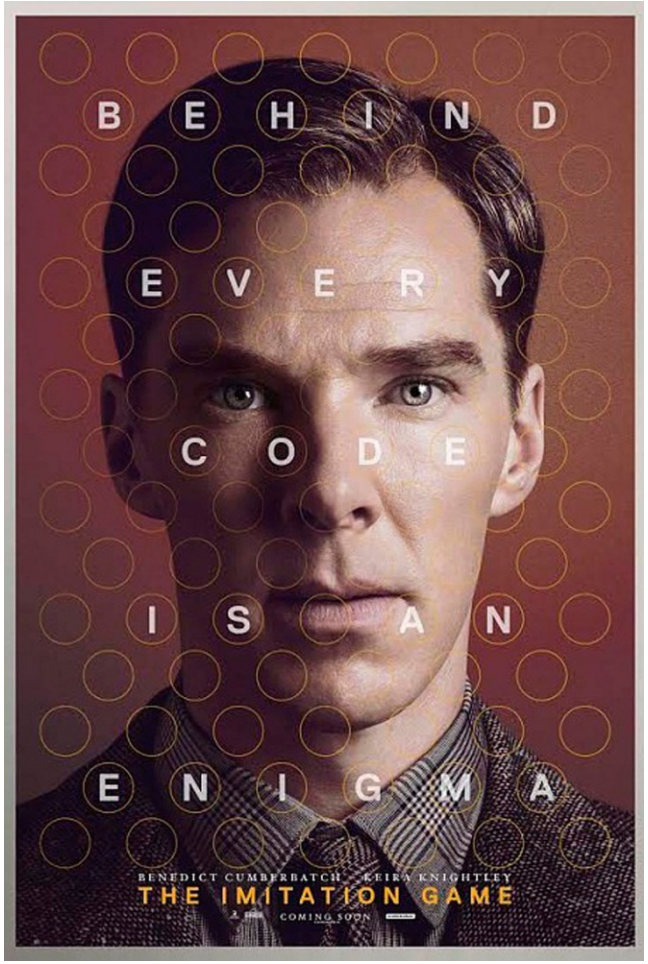
Turing Test (III)

- *A* is a machine designed and developed to play chess
  - E.g., IBM *Deep Blue*
- *B* and *C* are two chess players (not newbies, not champions)
- *C* is not in the same room where *A* and *B* are
- *C* does not known if the current opponent is either *A* or *B*
- *A* can be considered *as intelligent as B* if *C* cannot decide which is the current opponent



Acting Humanly

- The Turing test provides an empirical definition for the term *intelligent behavior*
  - In the original test, Turing did not talk about chess players, which inherently require rationality
- The Turing test advocates an anthropocentric approach
  - The goal is to act as a person
  - Intelligent machines play *imitation games*

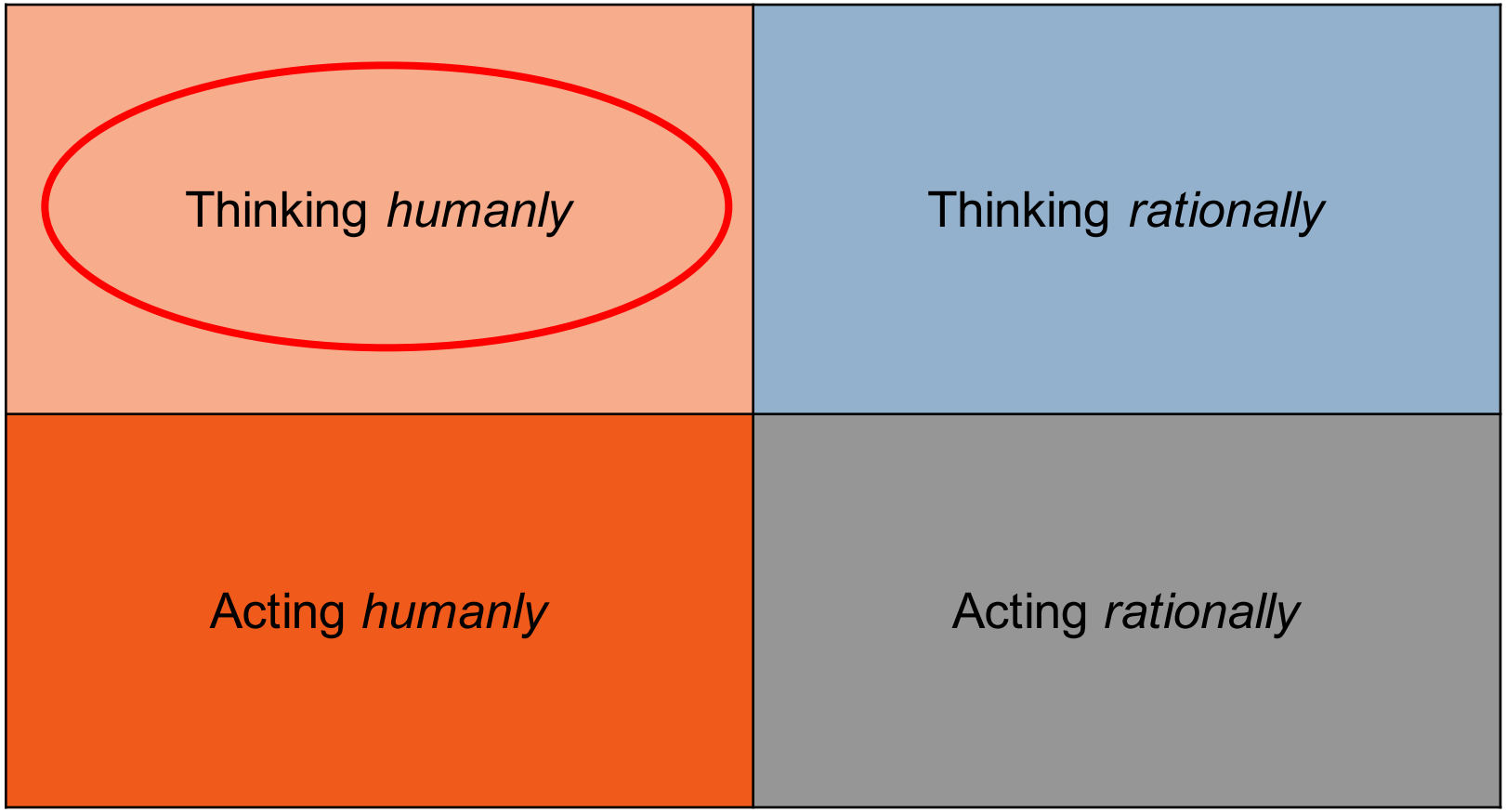


Eliza (J. Weizenbaum, 1966)

- The Turing test has been approached several times by several computer programs
- The computer program Eliza behaves as a Rogerian psychotherapist
- Eliza simply
  - Identifies relevant parts of input sentences
  - Rephrases input sentences to continue the conversation

```
> Hello, I am Eliza.
* Hi
> How are you today...What would you
like to discuss?
* I have problems at school
> Can you elaborate on that?
* I don't pass exams
> Do you wish to be able to pass
exams?
* Yes, sure
> I see.
* Math exams are too difficult for
me
> Come, come, elucidate your
thoughts.
* I don't see a meaning in math
> Do you wish to be able to see a
meaning in math?
```

Four Viewpoints

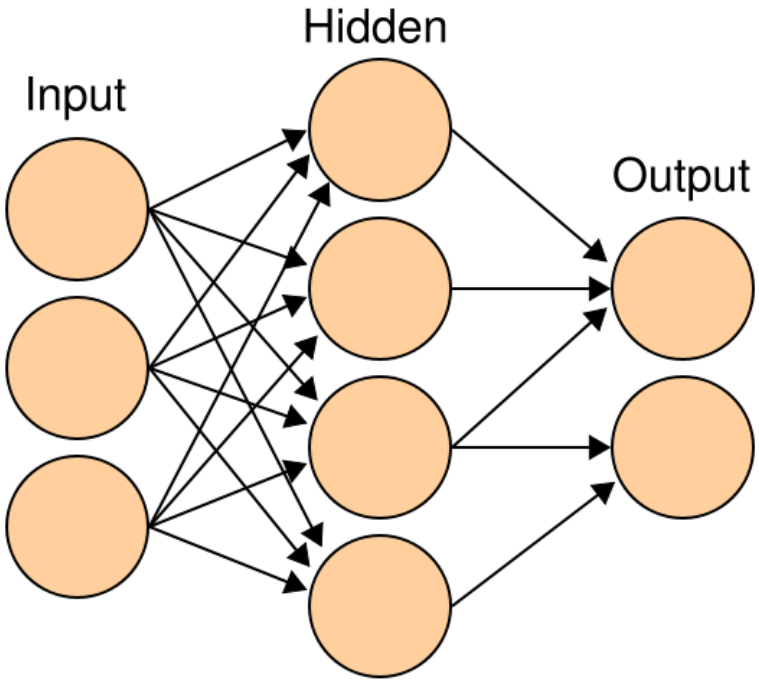


Thinking Humanly

- A possible approach to imitate human behavior is based on the possibility to simulate the organ that originates thoughts
  - The brain is the organ where thoughts originate
- Using this approach, the goal is to build *electronic brains* to simulate human brains
  - The simulation is at the cell level
  - The simulation includes simulated neurons, axons, soma, …

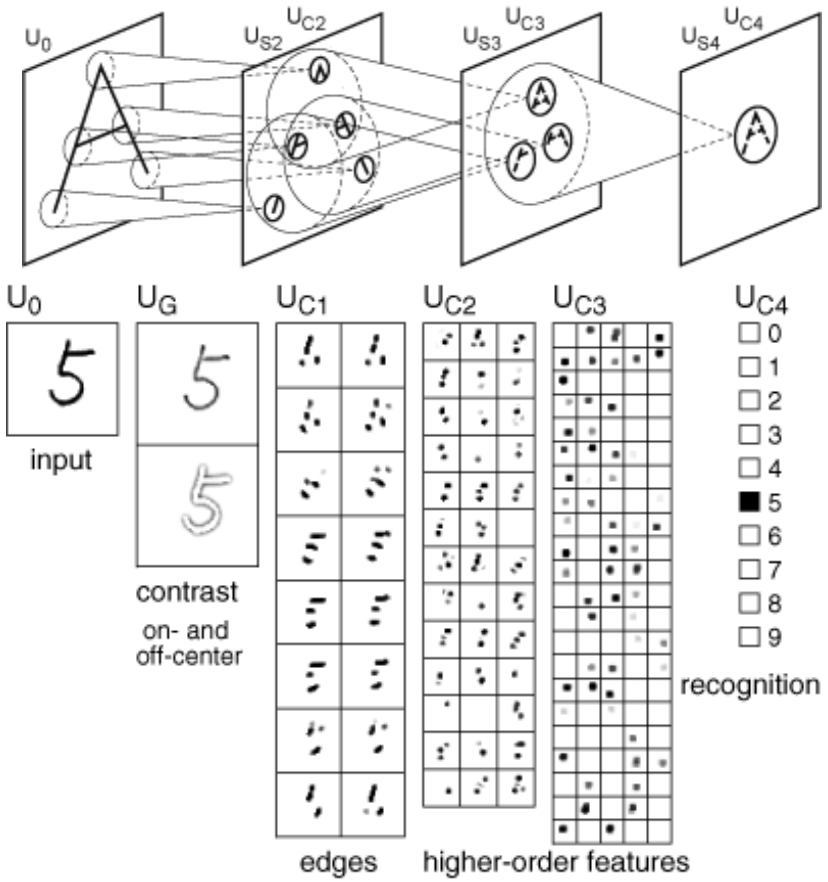
Neural Networks (I)

- Each *unit* simulates a neuron
- Units are connected in a *network*
  - That receives *percepts* from *sensors*
  - That provides *stimuli* to *actuators*
- The network is trained to learn the desired behavior
  - Normally, it is too complex to be explicitly programmed

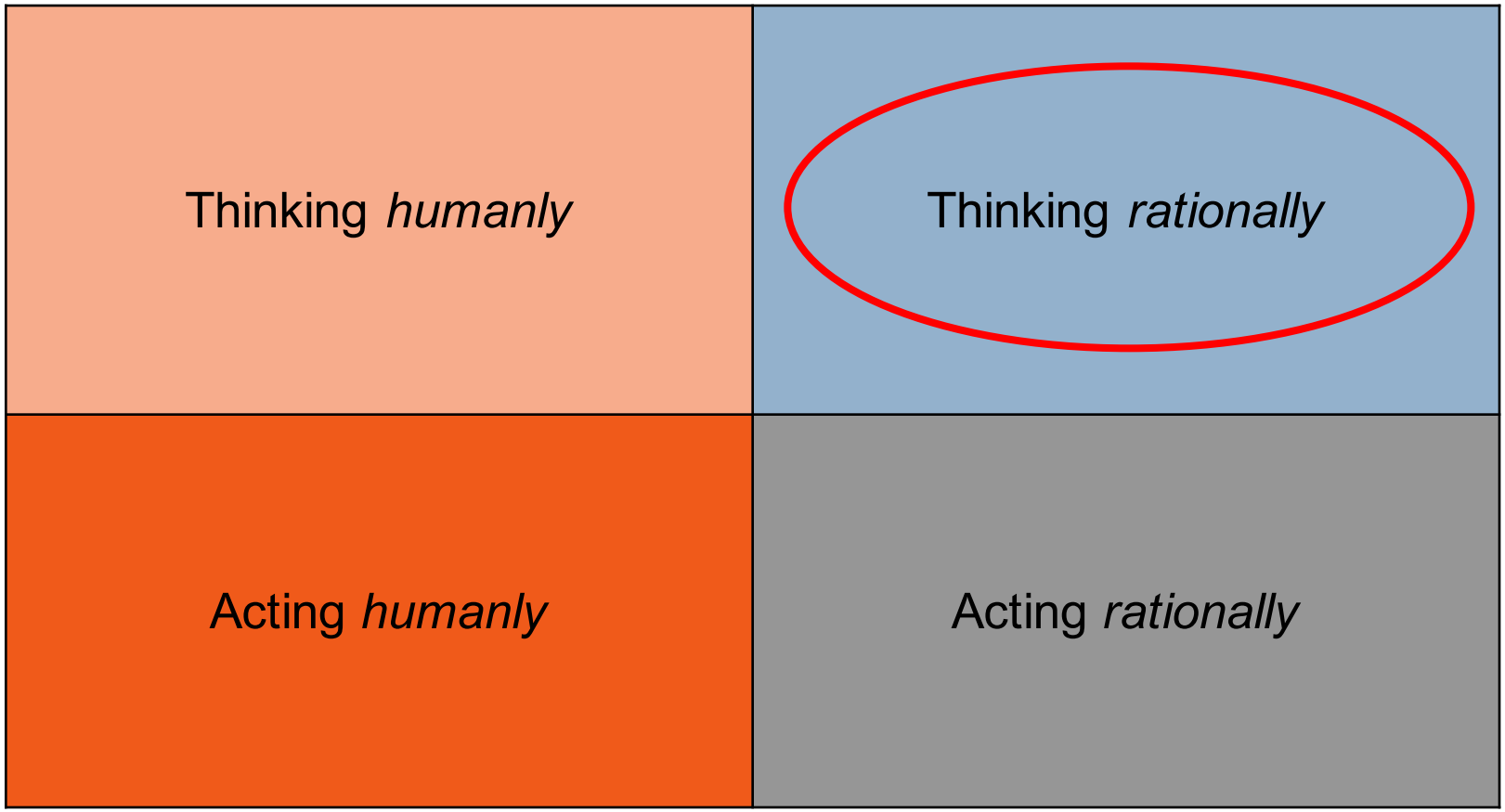


Neural Networks (II)

- *Neocognitron* (K. Fukushima, 1980)
  - Neural network trained to recognize handwritten characters
- It is an example of a
  - *Convolutional Neural Network* (*CNN*)
  - A *deep network* (a network with several layers) for deep learning



Four Viewpoints



Thinking Rationally

- Human beings are not always rational
  - They are influenced by habits, hopes, unreachable goals, unsolvable problems, …
- Rationality is described by logic languages or equivalent formalisms
  - Assume that $A \rightarrow B$ is considered true
  - B must be considered true as soon as A is known to be true
- Logic formalisms are used for deductions

Expert Systems

- A computer program written in terms of a *knowledge base*, which is a (possibly dynamic) set of *facts* and *inference rules*
  - Facts:

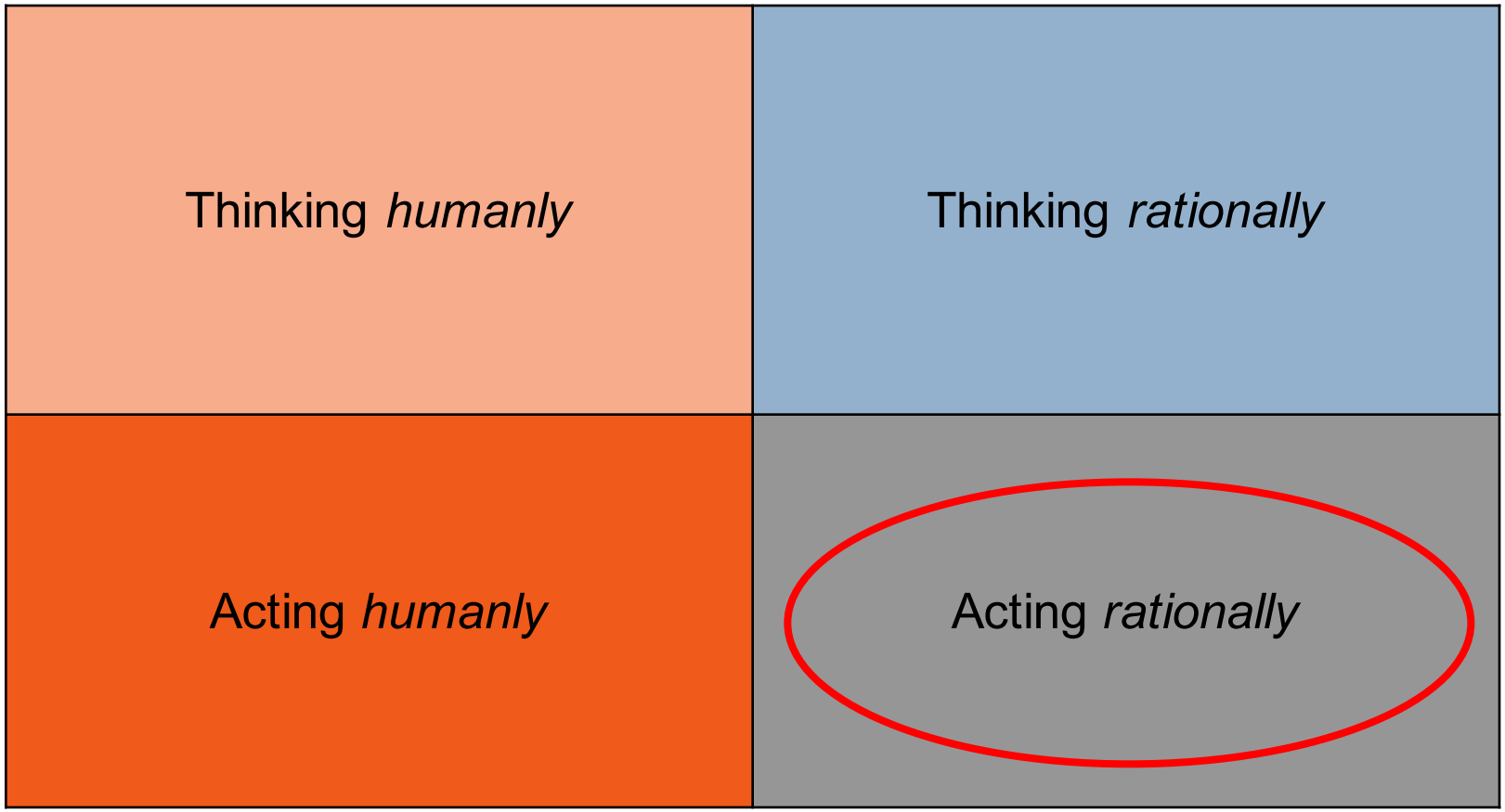
    $mother\_of(ann,bob),\ sister\_of(claire,ann)$
  - Inference rules:

    $mother\_of(X,Y) \wedge sister\_of(Z,X) \rightarrow aunt\_of(Z,Y)$
- The knowledge base of an expert system can be used to infer new facts from known facts

MYCIN (E. Shortliffe, 1970s)

- MYCIN is an expert system for medical diagnosis that
  - Contains more than 600 inference rules
  - Can ask yes/no questions
  - Provides a list of diagnosis and prescriptions
- An extensive experimental campaign measured that MYCIN was correct in 69% of the cases
  - It was correct more often than the doctors that supplied the list of used inference rules

Four Viewpoints



Acting Rationally

- The machines that act rationally are known as intelligent agents
  - They are often used to design and develop AI applications
- A rational agent is characterized by its behavior and the internal machinery used to obtain the behavior is not relevant
  - They are often based on logic languages
  - They are sometimes based on neural networks
  - They act rationally in their environment
  - They are not particularly good at imitating human behaviors

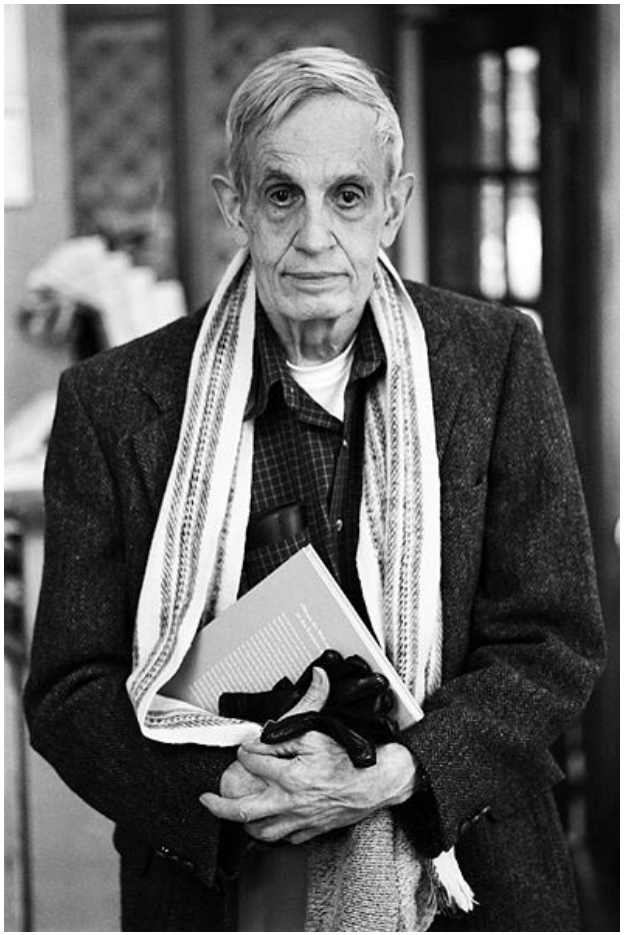
SHRDLU (T. Winograd, 1971)

- The computer program SHRDLU controls an emulated robot arm to interact in the blocks world
- The user can interact with SHRDLU using a natural language (English)
  - SHRDLU associates a meaning with user sentences
  - SHRDLU can ask questions to better understand the user
- SHRDLU has been the considered the prototypical AI program for several years

```
* Pick up a big red block.
> OK.
* Find a block which is taller than
the one you are holding and put it
into the box.
> BY "IT", I ASSUME YOU MEAN THE
BLOCK WHICH IS TALLER THAN THE ONE I
AM HOLDING.
> OK.
* What does the box contain?
> THE BLUE PYRAMID AND THE BLUE
BLOCK.
* What is the pyramid supported by?
> THE BOX.
* How many blocks are not in the
box?
> FOUR OF THEM.
* Is at least one of them narrower
than the one which i told you to
pick up?
> YES, THE RED CUBE.
* Is it supported?
> YES, BY THE TABLE.
```

Game Theory

- Game theory is used to design and develop intelligent agents
  - It was popularized by the works of John Nash (1928-2015)
  - It is studied and used, differently, by AI and Economics
- Game theory can be used to reason on rational behavior by describing the rational behavior in terms of the earned utility



Games Theory and AI

- AI often restricts its scope to games with the following characteristics
  - The game includes two players that play alternate turns
  - The game has simple and formalized rules
  - The game is played in a completely accessible environments, and therefore, players have *perfect knowledge* of the state of the game
  - The game (often) have strict time constraints

- For example
  - Checkers, chess, … are often considered in AI
  - Poker, bridge, … are not often considered in AI

“Real” Players (I)

- *Chinook* won the *Man-Machine Checkers Championship*
- It is still possible to play against Chinook

www.cs.ualberta.ca/~chinook

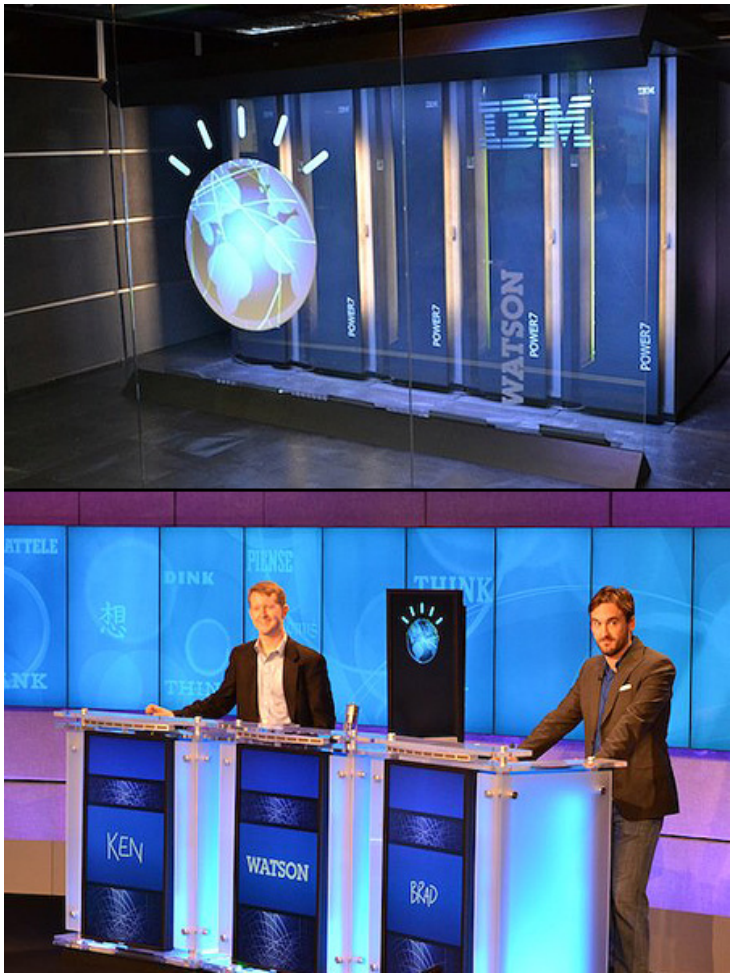
- *IBM Deep Blue* won against Garry Kasparov in 1996

“Real” Players (II)

- Go was considered unfeasible for computers
  - Chess: *branching factor* > 40, more than 50 turns per game
  - Go: branching factor > 250, more than 350 turns per game
- *AlphaGo* is an AI player for Go
  - In October 2015, AlphaGo won 5-0 against the European (human) champion
- AlphaGo uses neural networks to support a game-theoretic algorithm to decide next move

“Real” Players (III)

- *Watson* is a computer program that can answer questions expressed in English
  - It stores ~$200 \cdot 10^6$ documents including a dump of Wikipedia
- In 2011, Watson participated to three special episodes of *Jeopardy!* winning against several opponents
  - Watson was not connected to the Internet during the show recordings



Agents and Multi-Agent Systems

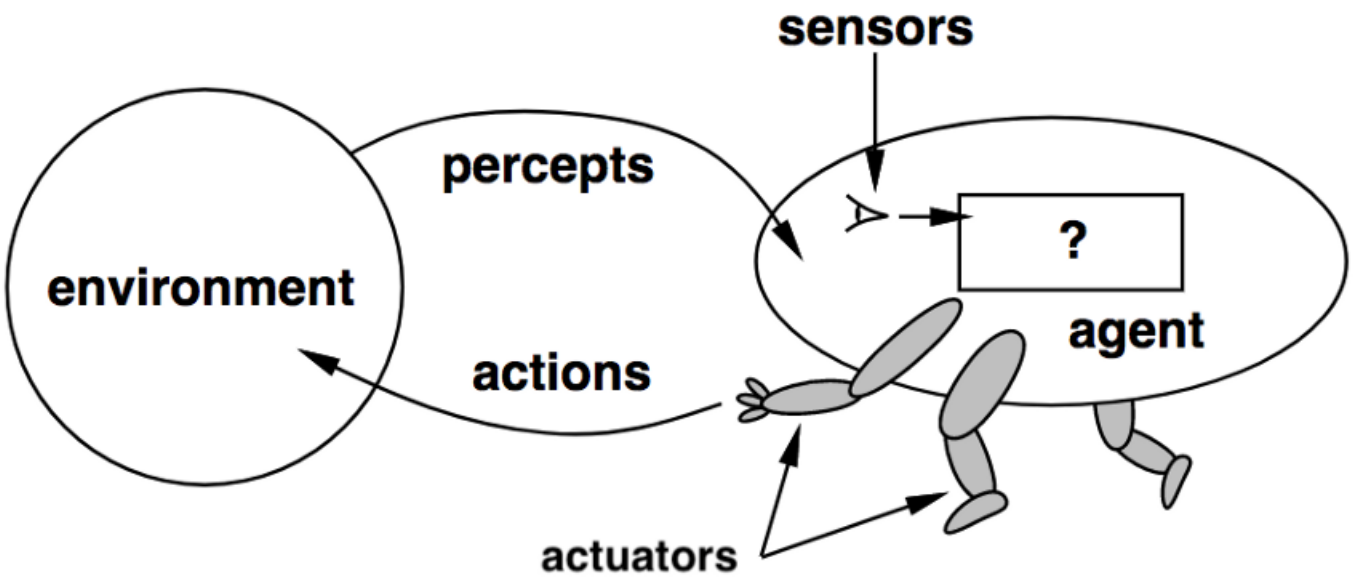
"I am about to propose the existence of something called the knowledge level*, within which knowledge is to be defined. To state this clearly, requires first reviewing the notion of computer systems levels."*

A. Newell. The Knowledge Level. *Artificial Intelligence*, 18(1):87–127, 1982

So, What is An Agent? (I)

- In AI, an agent can be broadly characterized as *anything* that
  - Perceives its environment
  - Takes actions autonomously to react to changes in the environment and/or to bring about its goals
  - May improve its performance using learning
  - May use knowledge to achieve *rationality*
- The debate on what should be *really* considered an agent is deeply rooted in the research on AI
  - Different authors provide different, often inconsistent, definitions
  - Commonly adopted definitions change over time to reflect mainstream research interests

So, What is An Agent? (II)



- ăgo, ăgis, egi, actum, ăgĕre
  - Latin verb meaning *to act*, *to do*, *to lead*
  - Common root for *actors* and *agents*

Agents and Environments

- A broad characterization of agents is based Percepts, Actions, Goals, and the Environment (PAGE) [1]
- The features of the environment are crucial to such a characterization of agents
  - Fully observable vs partially observable
  - Deterministic vs stochastic
  - Static vs dynamic
  - Discrete vs continuous
  - Episodic vs sequential
  - Single-agent vs multi-agent

[1] S.J. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003

Types of Environments (I)

- Fully observable (accessible) vs partially observable (or unobservable/inaccessible)
  - If the agent can sense the complete state of the environment at each moment in time, then the environment is fully observable
  - A fully observable environment is easy for the agent because there is no need to maintain an internal state to track the history of the world
- Deterministic vs non-deterministic (or stochastic)
  - If the current state of the agent, including the selected action to be performed next, completely determine the next state of the environment, then the environment is deterministic
  - In a deterministic, fully observable environment, the agent does not need to worry about uncertainty

Types of Environments (II)

- Static vs dynamic
  - If the environment can change while an agent is choosing the next action to perform, then the environment is dynamic
  - Static environments are easy for the agent because the agent does not need to sense the environment while choosing the next action
- Discrete vs continuous
  - An environment with a discrete number of percepts and a discrete number of actions is discrete
- Episodic vs sequential
  - In an episodic environment, only the current percept is required to choose the next action to perform
  - In a sequential environment, the agent needs memory of past actions to determine the next action to perform

Types of Environments (III)

- Single-agent vs multi-agent
  - If only one agent is in the environment, then the environment is single-agent
  - Multi-agent environments are normally more challenging than single-agent environments

- Multi-agent environments can be
  - *Cooperative*, when agents cooperate to bring about *joint goals*
  - *Competitive*, when agents bring about their goals with conflicting (or absent) relationships with other agents
  - *Strategic,* if the environment is deterministic except for the actions of other agents

Agents for Weiss

- Weiss [1] classifies agents on the basis of their internals
  - *Reactive agents*: the decision about which action to perform is described in terms of a direct mapping from conditions to actions
  - *Layered(-architecture) agents*: the decision about which action to perform uses layers, each of which is intended to reactively reason at a different level of abstraction
  - *Logic-based agents*: the decision about which action to perform is made using logical reasoning
  - *Belief-Desire-Intention (BDI) agents*: the decision about which action to perform depends on the processing of data representing the beliefs, the desires, and the intentions of the agent

[1] G. Weiss. *Multiagent Systems*. MIT Press, 2013

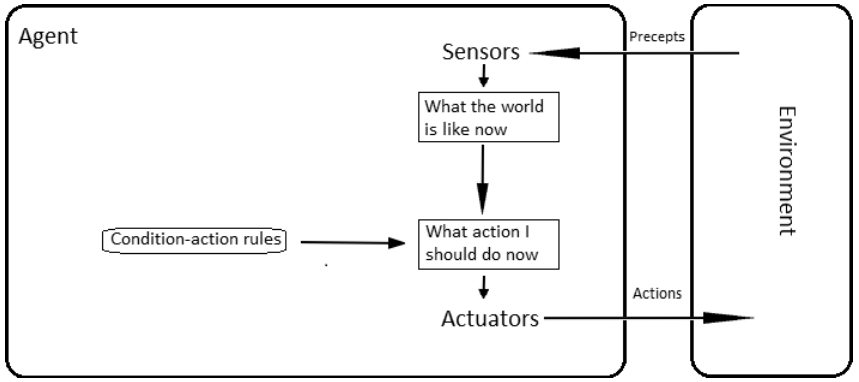
Agents for Russel & Norvig

- Russel & Norvig [1] classify agents on the basis of the relevance of some of the characteristics commonly associated with intelligence
  - Simple (reflex) agents
  - Model-based (reflex) agents
  - Goal-based agents
  - Utility-based agents
  - *Learning agents (not relevant here)*

[1] S.J. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003

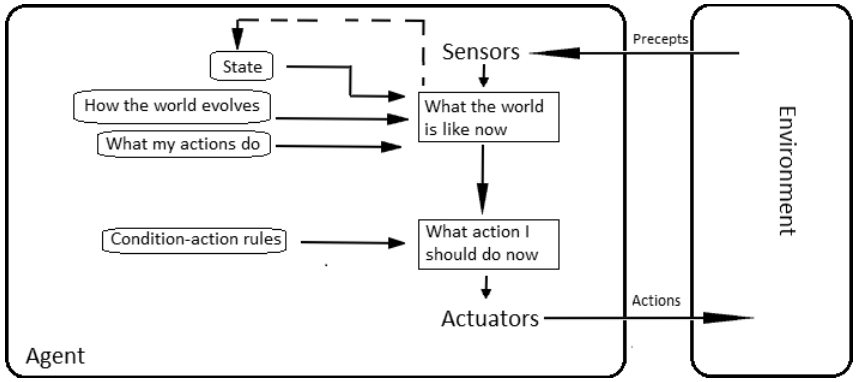
Simple (Reflex) Agents

- Simple agents act on the basis of the current percept (only)
  - They ignore the rest of the percept history
  - Their behaviors are based on *condition-action* rules
- They are usable only when the environment is fully observable



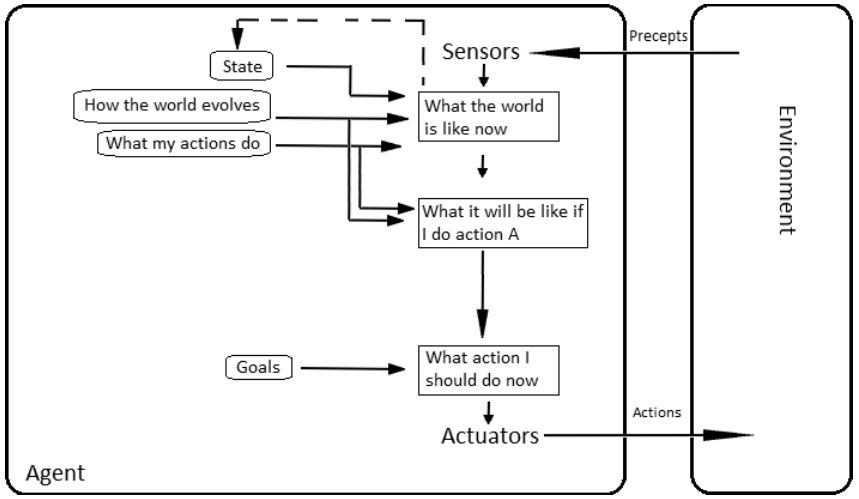
Model-Based (Reflex) Agents

- A model-based agent
  - Maintains an internal model that depends on the percept history
  - Uses the model to predict the impact of future actions
- They are usable also in partially observable environments



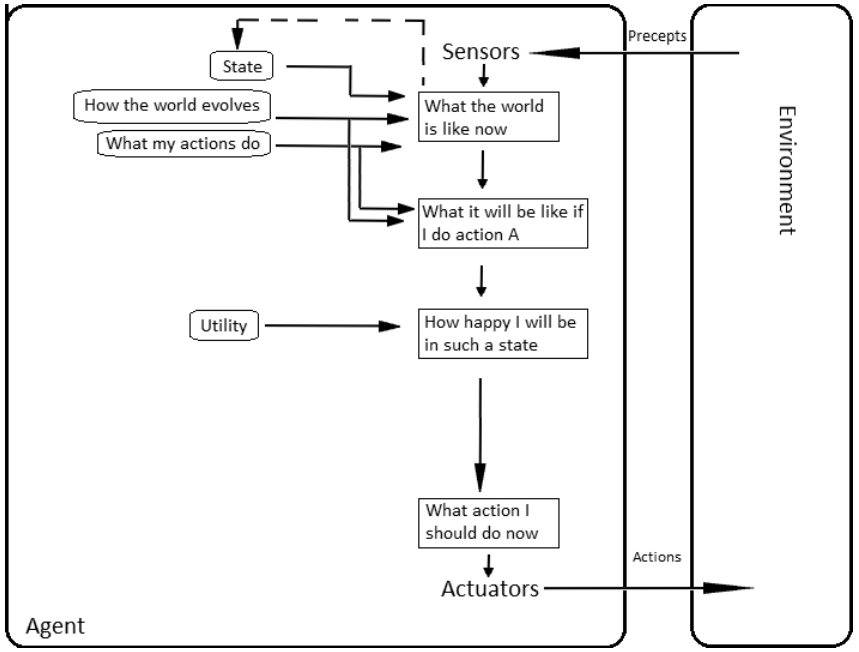
Goal-Based Agents

- Goal-based agents are model-based agents with explicit goals
  - They choose what to do to bring about their current goals, which describe situations that are currently considered desirable
  - Multiple, non-conflicting goals are allowed



Utility-Based Agents

- Utility-based agents are goal-based agents that
  - Have a *utility function* to associate *utilities (or payoffs)* with goals
  - Actions are chosen to bring about the goals with maximum expected utility
  - Multiple, non-conflicting goals are allowed



Types of Software Agents

- Despite the differences in definitions, the following broad classes of agents are commonly accepted
  - *Software agents*: agents that are implemented as software systems
  - *Distributed agents*: software agents that execute in distributed computing environments
  - *Mobile agents*: distributed agents that can move across the nodes of the distributed computing environment
  - *Autonomous agents*: software agents that act to achieve, possibly implicit, goals
  - *Intelligent (software) agents*: autonomous agents whose major characteristics are those related to rationality and learning

Multi-Agent Systems

- A Multi-Agent System (MAS) is a *software system* composed of multiple interacting software agents that share an environment
- Agents in a MAS communicate via
  - *Direct communication*, which is based on message passing
  - *Mediated communication*, which is based on the use of the shared environment for communication purposes
- Despite the open debate on agents and their characteristics, there is general consensus [1] on the definition of MASs

[1] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002

IEEE FIPA – A Standard for MASs

- In 1996, a worldwide group of companies teamed to establish the Foundation for Intelligent Physical Agents (FIPA) [1]
  - The initiative was promoted by Leonardo Chiariglione
  - FIPA delivered interoperability specification for MASs
  - FIPA contributed to promote research and innovation on software agents and MASs
  - In 2006, FIPA became an *IEEE Standardization Committee*

[1] F. Bergenti et al. The first twenty years of agent-based software development with JADE. *Autonomous Agents and Multi-Agent Systems*, 34(2):1–19, 2020

Research on Agents and MASs

- Research results on agents and MASs are presented in virtually all conferences and journals that target *symbolic AI* (still different from *subsymbolic AI*)
- Agents and MASs are a topics on their own
  - The conference specifically devoted to the research on agents and MASs is the *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, a top conference since 2002
  - The journal specifically devoted to the research on agents and MASs is *Autonomous Agents and Multi-Agent Systems* (often called *JAAMAS*), a top journal since 1998

(Artificial) Neural Networks

"The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

The New York Times, July 8th, 1958

(Artificial) Neural Networks (I)

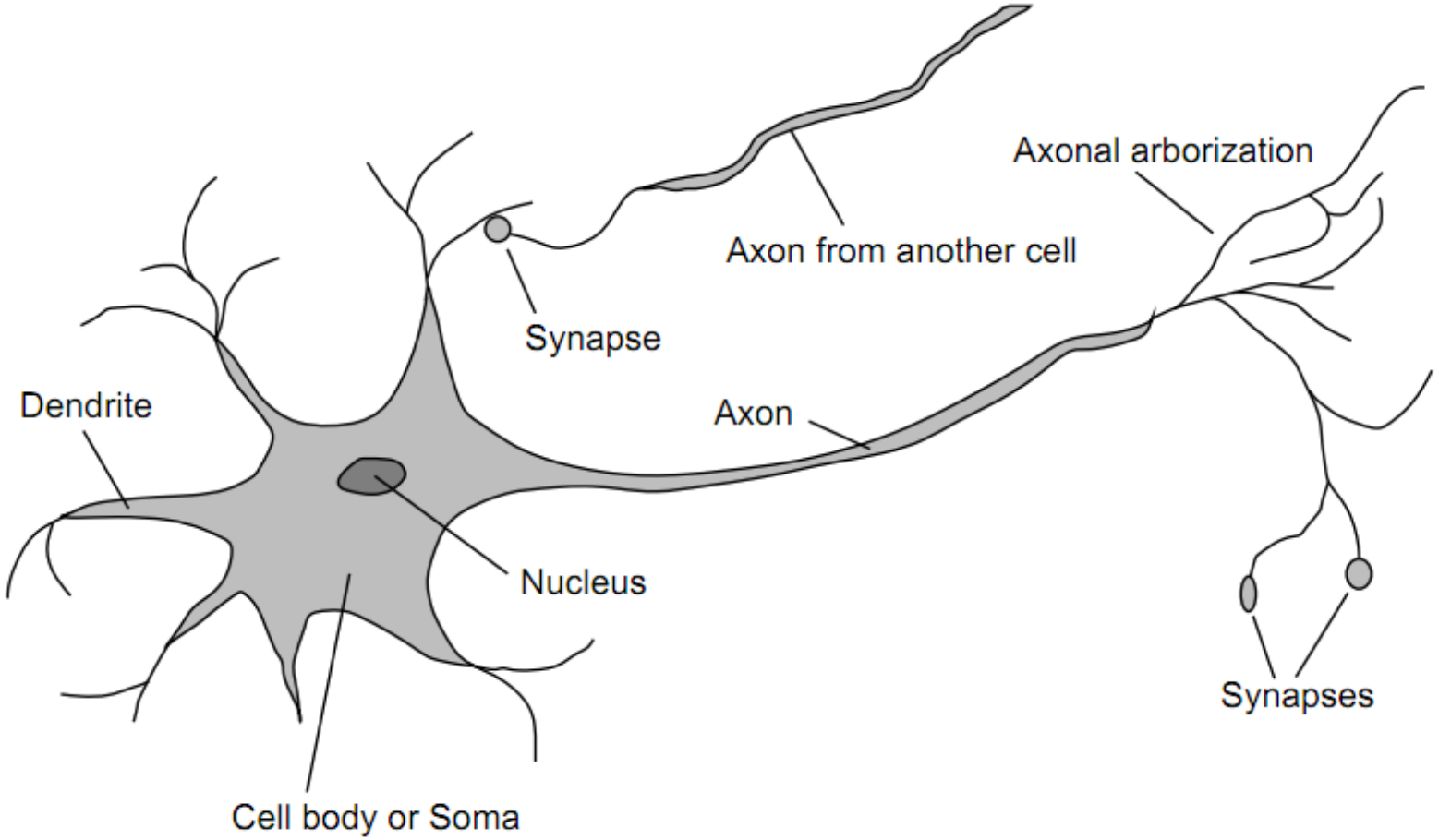
- An (Artificial) Neural Network (ANN) is a tool to approximate functions of several variables

- The basic ideas of ANNs are as follows
  - Only very simple units, called neurons, are used
  - Neurons are connected to form a (complex) network
  - The structure (*topology*) of the network primarily determines the approximation capability

(Artificial) Neural Networks (II)

- ANNs do not execute programs but they *react* to input
- ANNs are *trained* to
  - *Learn* expected behaviors for known input
  - *Generalize* to ensure adequate behaviors for unknown input
- ANNs cannot justify their behaviors
  - They are the core of *subsymbolic AI*
  - They are difficult to use for *eXplainable AI* (*XAI*)

Neurons (I)

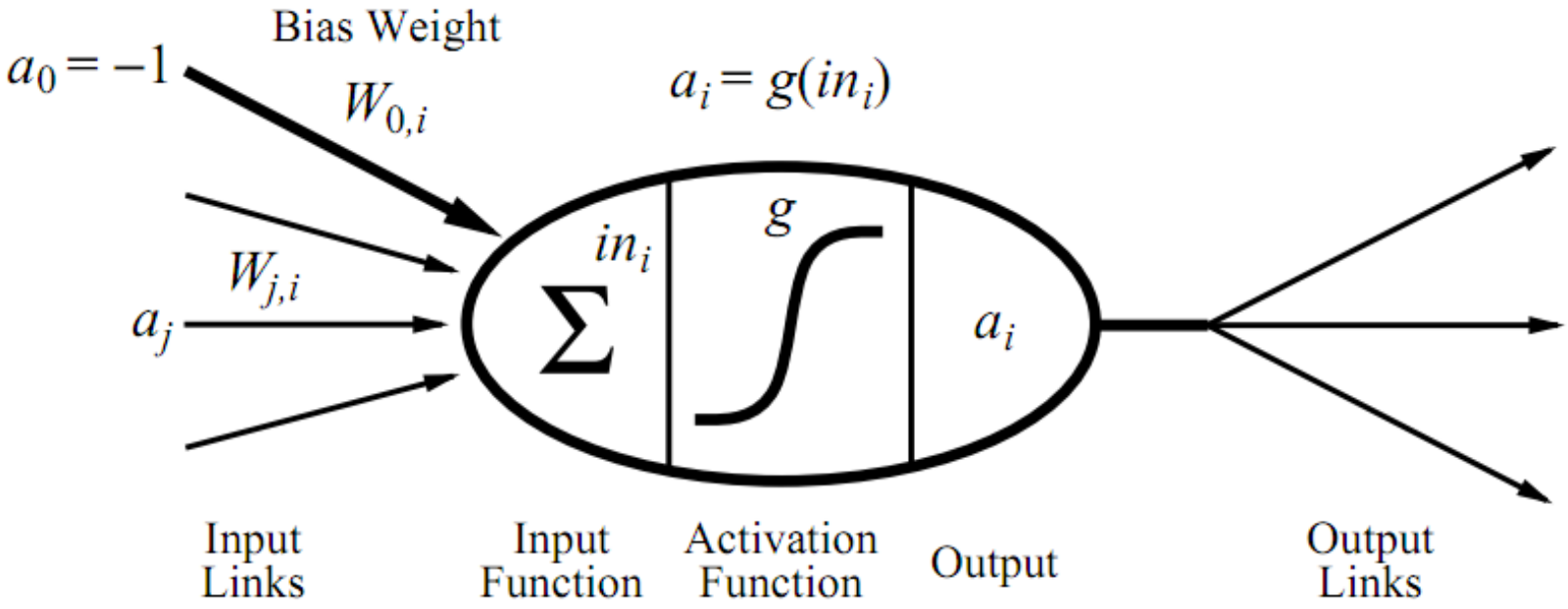

Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Neurons (II)

- Human brains normally contains (roughly) 10^{11} neurons
  - More than 20 types of neurons
  - Connected by 10^{14} synapses
  - Slowly reacting to changes in 1ms-10ms
- Human brains are *complex systems* that are very difficult to simulate
  - Mostly for the enormous number of connections in the network

The McCulloch-Pitts Unit (I)

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$

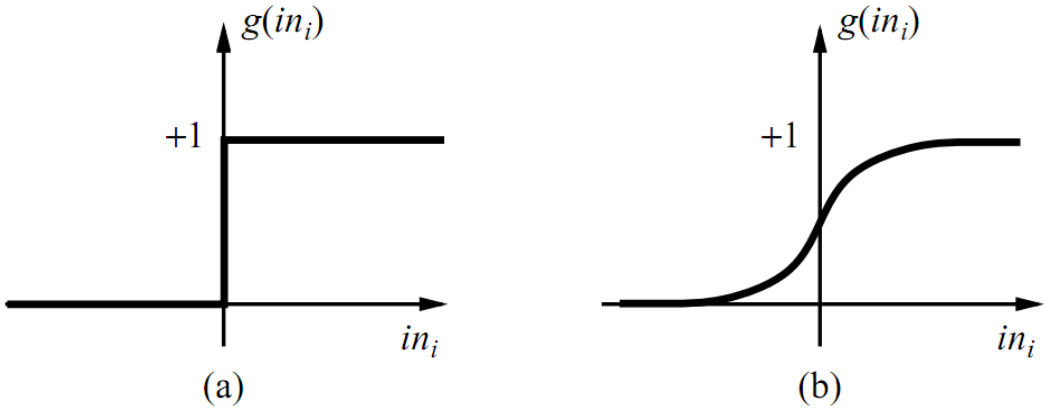


The McCulloch-Pitts Unit (II)

- Very simple and unrealistic model of neurons, they are the basic units of ANNs
  - Note that the power of ANNs does not follow from units but from the network
- The characterizing parameters of a McCulloch-Pitts unit are
  - The *weight* on each arc: $W_{j,i}$
  - The *bias weight*: $W_{0,i}$
  - The *activation function*: $g(.)$

The McCulloch-Pitts Unit (III)

- Several activation functions are used
  - *Linear function*: $L(x)=x$
  - *Rectified Linear Unit* (*ReLU*): $R(x)=\max\{0,x\}$
  - *Heaviside step function*: $H(x)$
  - *Logistic function*: $S(x)=1/(1+e^{-x})$



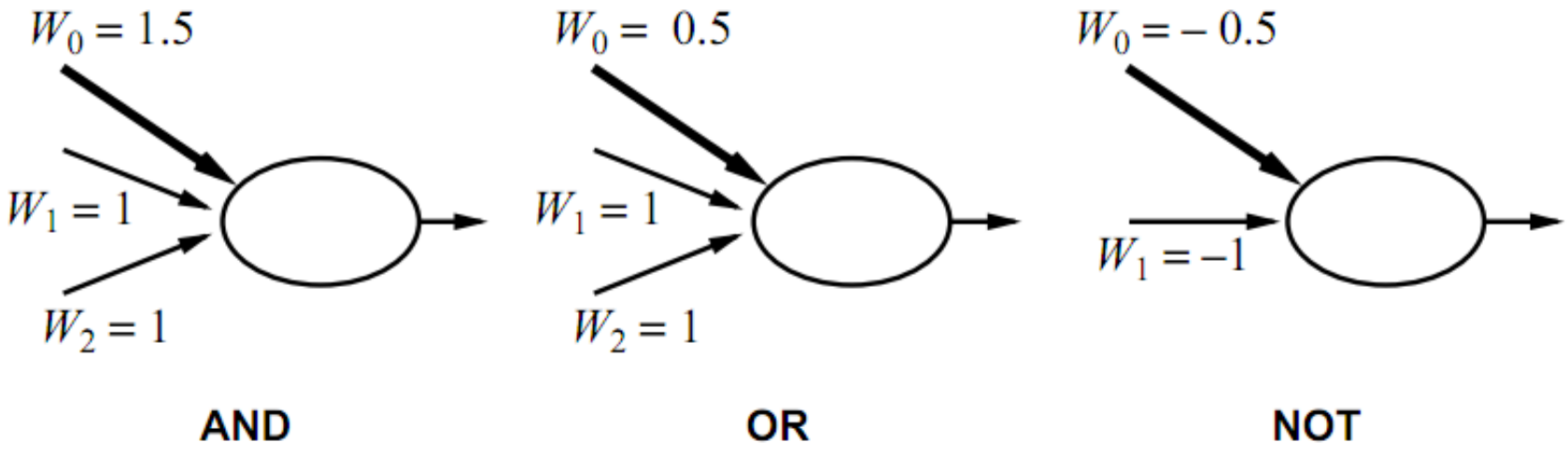
The McCulloch-Pitts Unit (IV)

- Normally, in an ANN
  - The activation function is fixed and shared among neurons
  - The activation function provides the needed nonlinearity
  - The network is known if a set of weights capable to obtain the expected behavior is known (bias weight included)

- The *learning process* is the process of identifying sufficiently good weights
  - Normally, it is the most difficult part of the construction of an ANN

Neurons as Logic Devices

- Neurons can be used as logic devices
  - Input and output in $\mathcal{B} = \{ 0, 1 \}$
  - The activation function is the Heaviside function



Types of Neural Networks

- *Feed-forward* neural networks
  - *Directed Acyclic Graph* (*DAG*) topology structured in terms of *layers*
  - No arcs connect a layer with previous layers
  - One input layer and one output layer
- *Recurrent* neural networks
  - *Directed graph* topology
  - No input and output layers
  - A network (sort of) memorizes *learned input*

Types of Training (I)

- *Training* phase
  - The ANN is provided with sample input from a *training set*, which is normally a set of real vectors
  - The training set (or a subset of it) is fed to the ANN for several *epochs*
  - During epochs, weights are modified to improve the behavior of the ANN on the training set
- *Test* (or *validation*) phase
  - The performance of the ANN is validated against a *test set*, which is normally a set of real vectors
  - Training and test sets are disjoint

Types of Training (II)

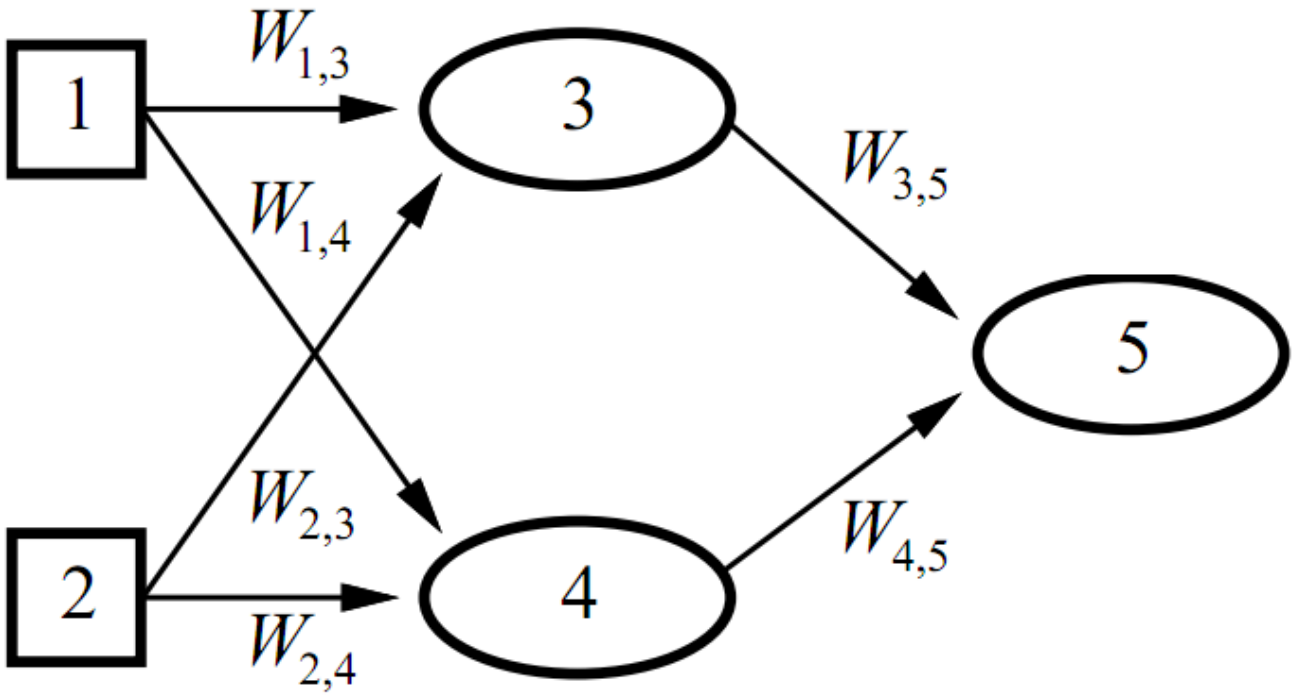
- *Supervised training*
  - The training set is composed of correctly associated input/output couples <*x*,*y*>
  - The error of the network can be computed by comparing actual output y_N with expected output *y*
  - Weights are corrected to reduce the computed error
  - Weights can be corrected
    - For each input/output couple in the training set
    - For each epoch (by gathering a cumulative error)
- Supervised training is often used for feed-forward networks

Types of Training (III)

- *Unsupervised training*
  - The training set is composed of input *x* for the ANN (no output is provided)
  - The rule to modify the weights to reduce the errors in the training set is fixed *a priori*
    - For example, weights are modified to make the ANN behave similarly for similar input vectors
  - The test phase assumes that the expected behavior of the ANN is provided together with input
- Unsupervised training is often used for recurrent networks

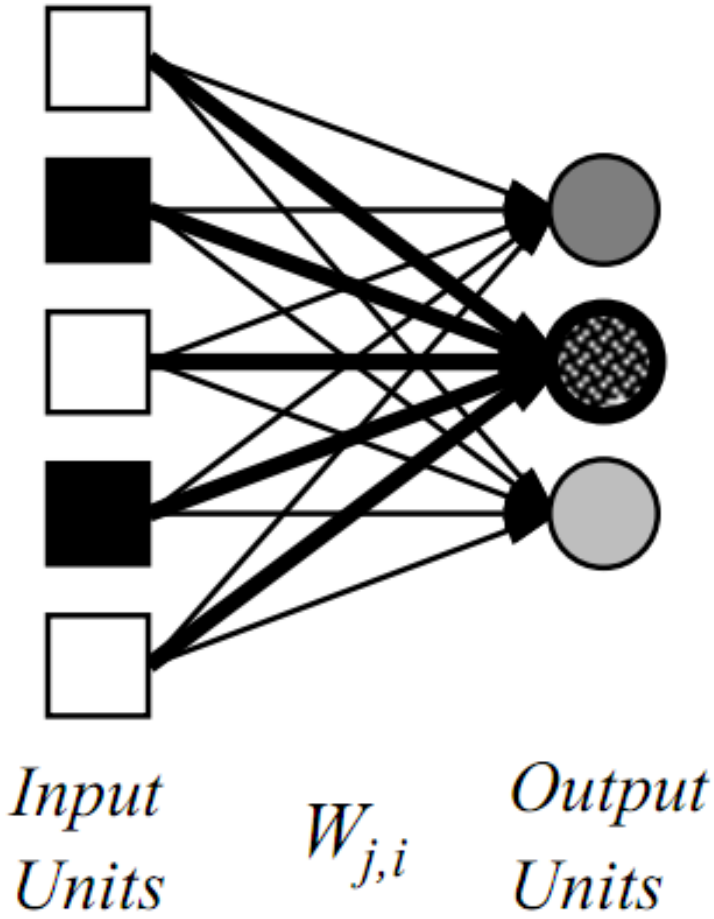
Feed-Forward Networks

- Feed-forward networks implement *nonlinear functions of several variables*



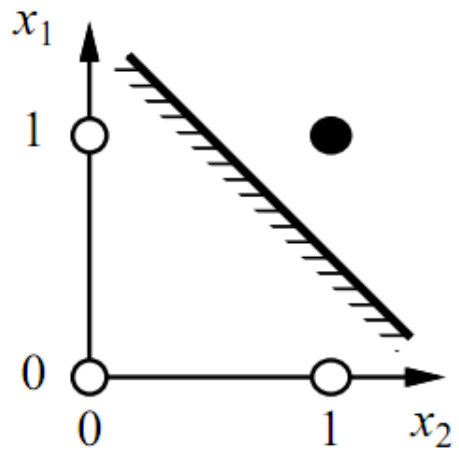
Single-Layer Perceptrons (I)

- *Single-Layer Perceptrons* (*SLPs*) are the simplest feed-forward networks
  - A weight vector is used to map input vectors to a component of the output vectors
  - The activation function is only used to compute output vectors

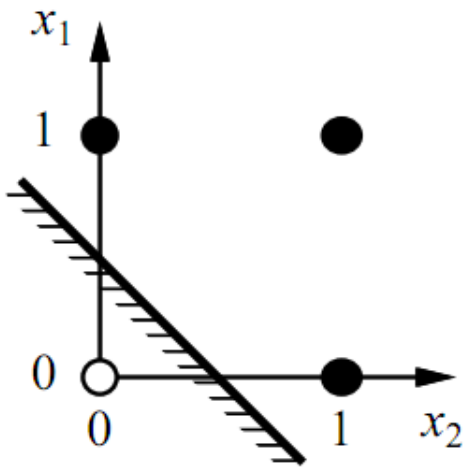


Single-Layer Perceptrons (II)

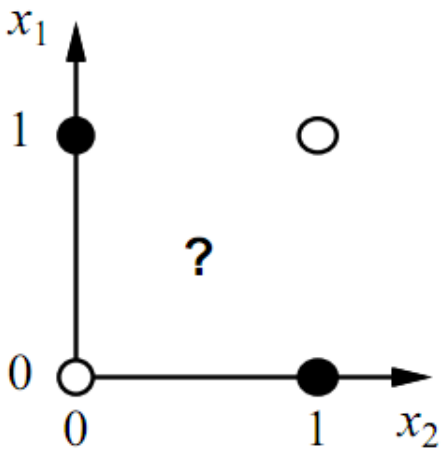
$\sum_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$



(a) x_1 **and** x_2



(b) x_1 **or** x_2



(c) x_1 **xor** x_2

Supervised Training for SLPs (I)

- It is worth recalling that
  - ANNs are used to approximate functions
  - The used activation function is fixed *a priori*
  - The approximation error changes with the weights of the network, and therefore, it is a function *Err*(*W*) of the weights called *error function*
- The supervised training of an ANN is meant to find a set of weights to minimizes the error function *Err*(*W*)
  - Ordinary algorithms to find minima of multivariate functions can be used

Supervised Training for SLPs (II)

- The supervised training of an SLP searches for a *weight vector* for each output neuron that minimizes

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2$$

- The *gradient descend algorithm* can be used to find the local minima of $E(W)$
  - The algorithm works on each output neuron
  - It starts from an appropriate weight vector (a random vector is often used)
  - It computes $E(W)$ and modifies the weight vector using $-\nabla E$

Supervised Training for SLPs (III)

- Note that

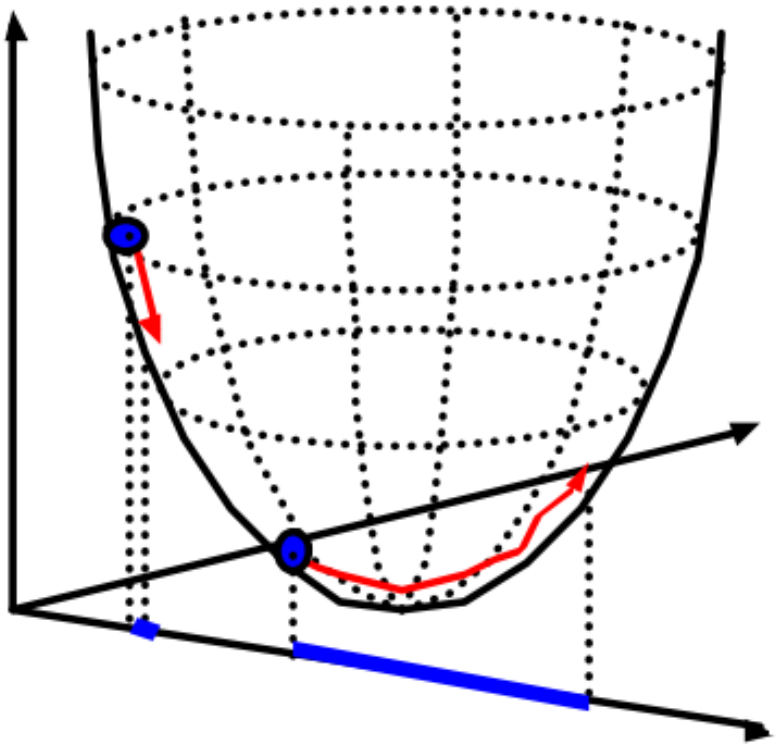
$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\Sigma_{j=0}^{n} W_j x_j)\right)$$
$$= -Err \times g'(in) \times x_j$$

- Therefore, the following update rule can be used to train an SLP for a given *learning rate* α

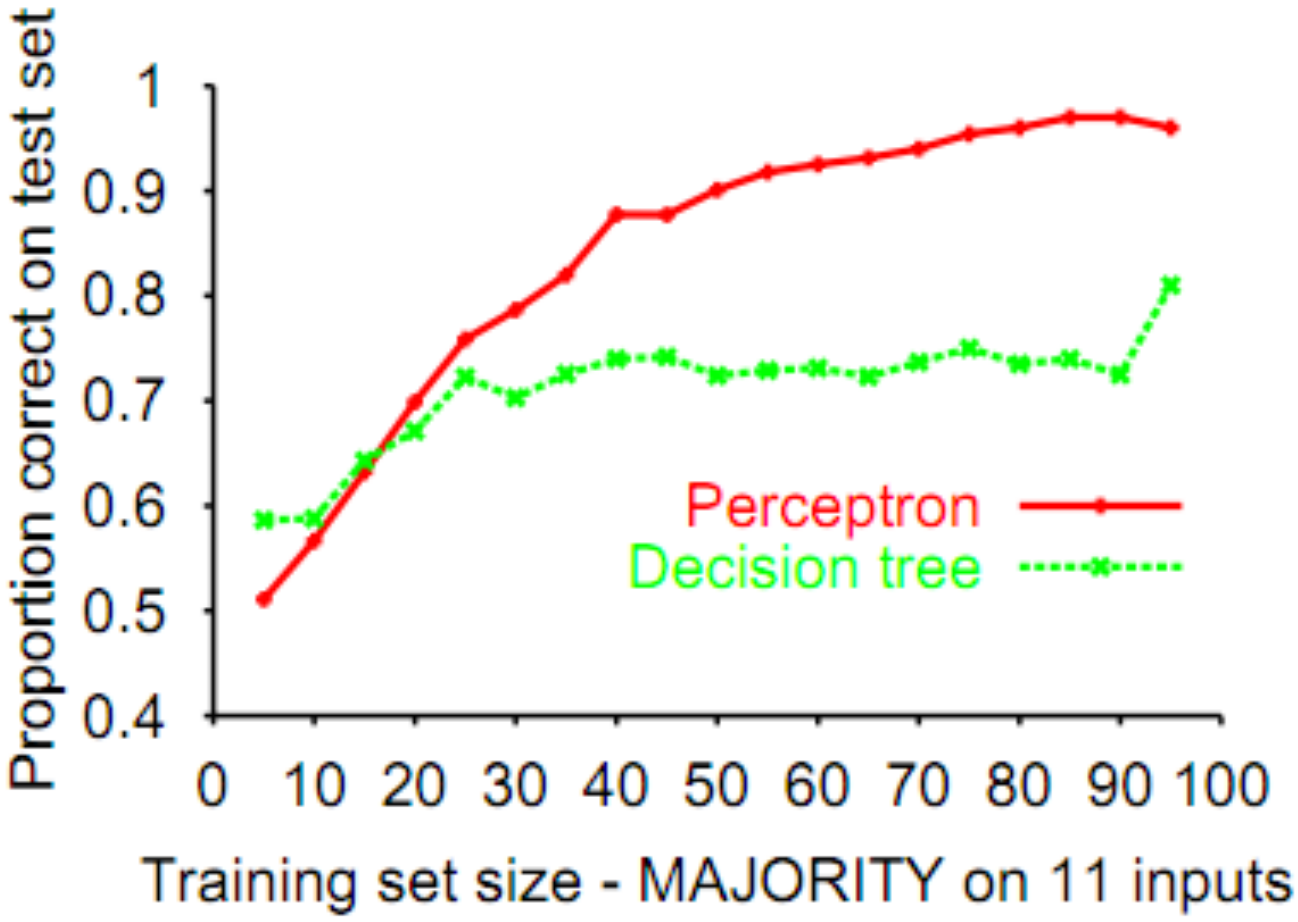
$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

Supervised Training for SLPs (IV)

- As usual, the gradient descend algorithm finds *local* (and not *global*) minima
- Smaller learning rates
  - Make the search more accurate
  - Increase the time needed to converge to a local minimum



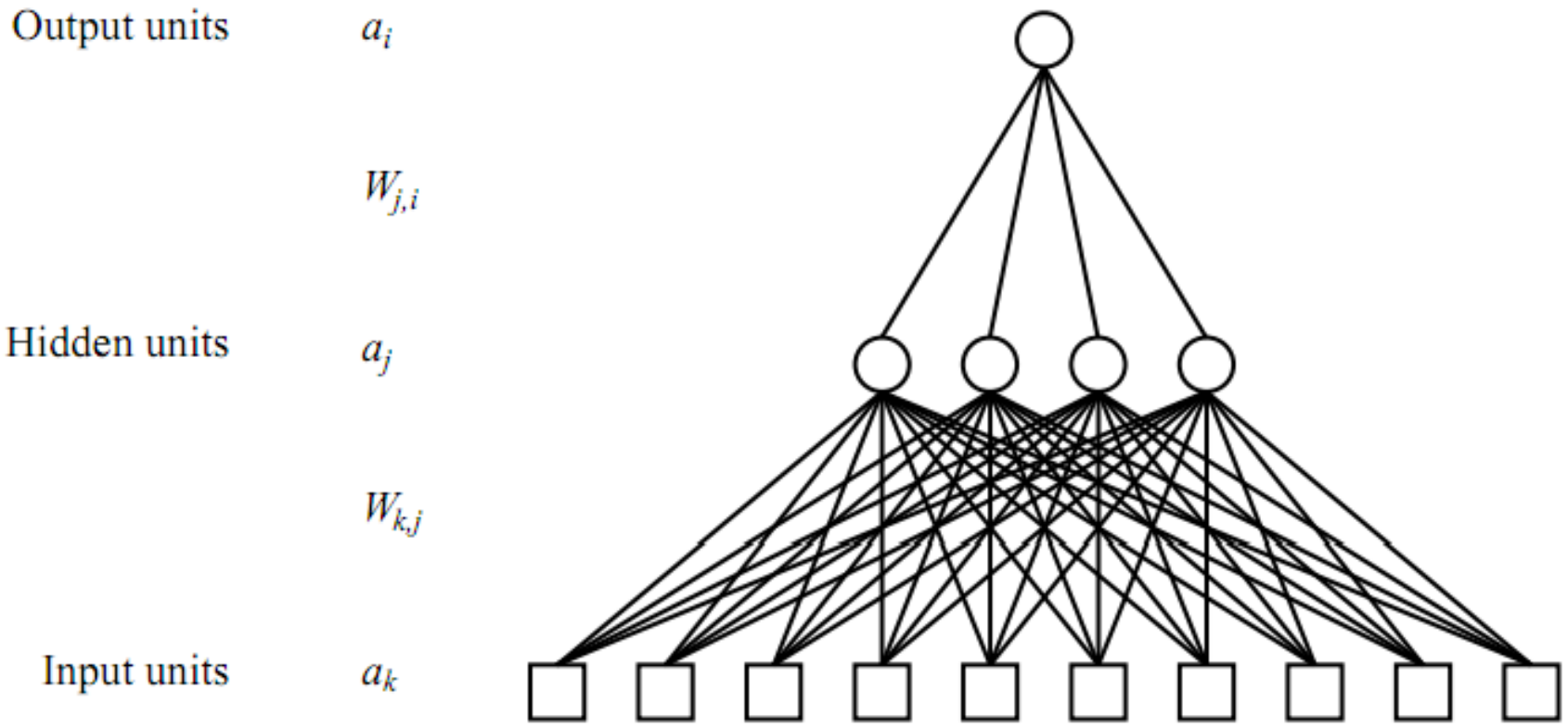
Supervised Training for SLPs (V)



Generalization and Overfitting

- A network is expected to
  - Approximate well the elements of the training set, as obtained with *accurate* training
  - Approximate well the elements of the test set, as obtained if the training set is *representative*
- A well designed training is expected to meet both needs, and a well trained network
  - Generalizes and provides good output for the elements of the test set
  - Does not overfit the elements of the training set

Multi-Layer Perceptrons (I)

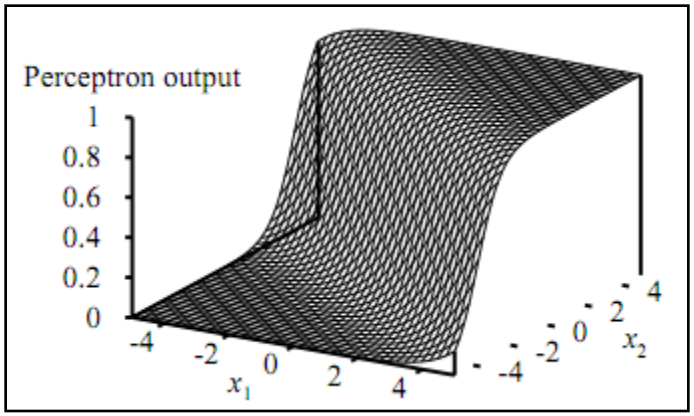


Multi-Layer Perceptrons (II)

- A Multi-Layer Perceptron (MLP) is a feed-forward ANN with at least one *hidden layer*
  - The number and topology of hidden layers is fixed *a priori*
  - The activation function is fixed *a priori* and shared among all neurons
- MLPs are inherently better than SLPs at approximating functions

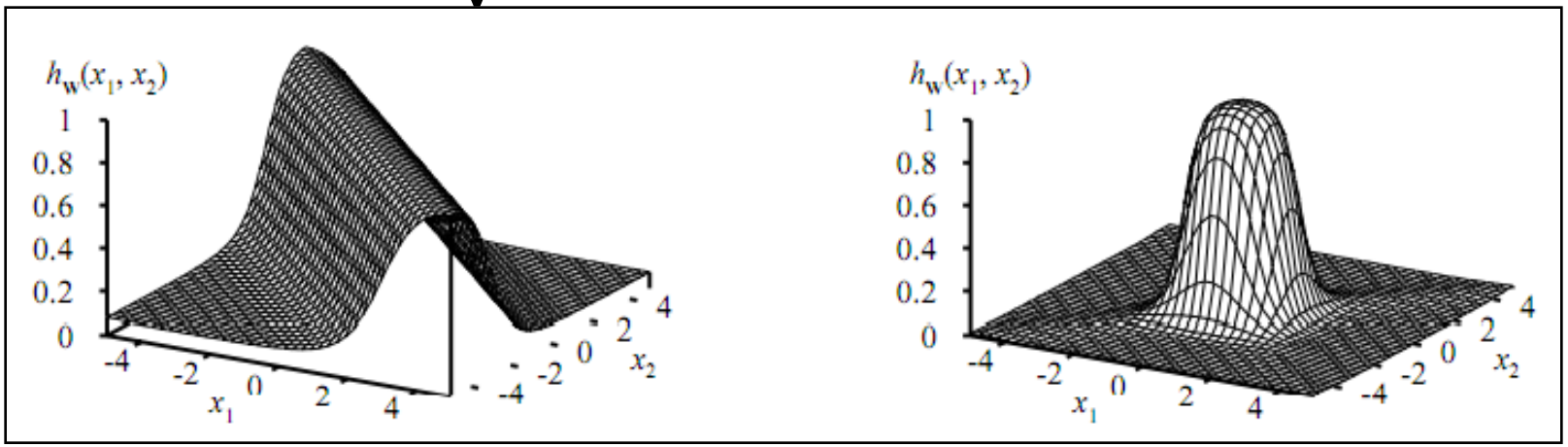
Multi-Layer Perceptrons (III)

SLP →



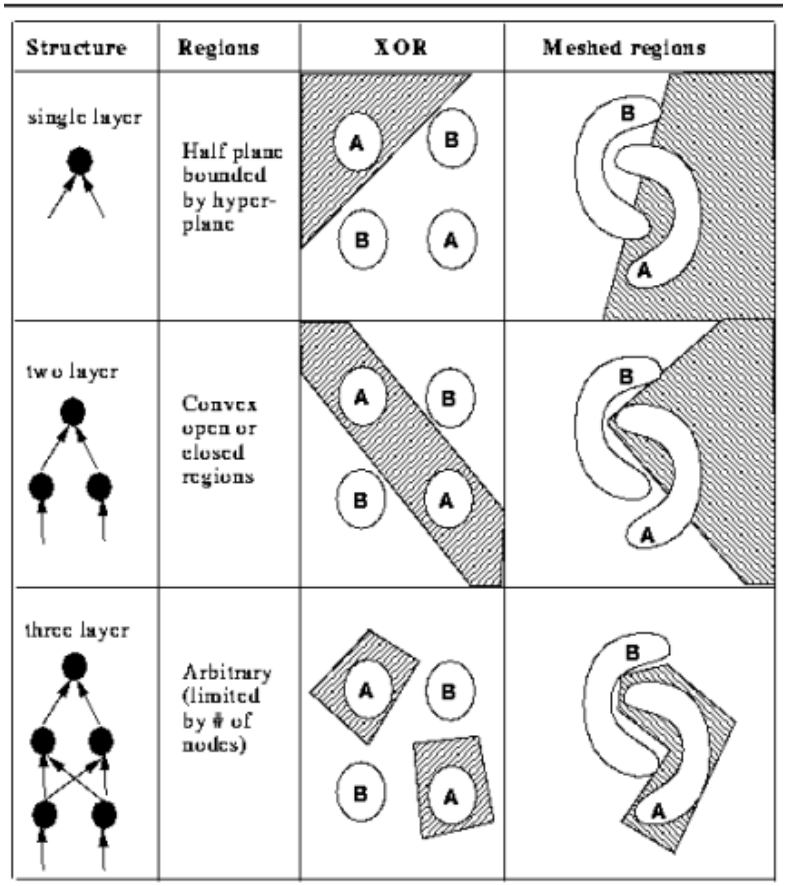
MLP

↓



Multi-Layer Perceptrons (IV)

- The addition of hidden layers increases the flexibility in approximating complex functions
- Two hidden layers are sufficient to approximate all functions (also discontinuous) for a fixed tolerance ε
  - Layers are fully connected
  - The number of neurons in each layer is not limited



The Back-Propagation Algorithm

```
// Back-propagation algorithm
procedure BackPropagation
    Initialize weights to small random values in (-1, 1)

    do
        Initialize the cumulative error to zero

        for each pattern in the training set do
            call ForwardPropagate
            call BackPropagate
            Update the cumulative error
        end
    while (number of epochs < max number of epochs) and
          (cumulative error > accepted tolerance)
end
```

The `ForwardPropagate` Procedure

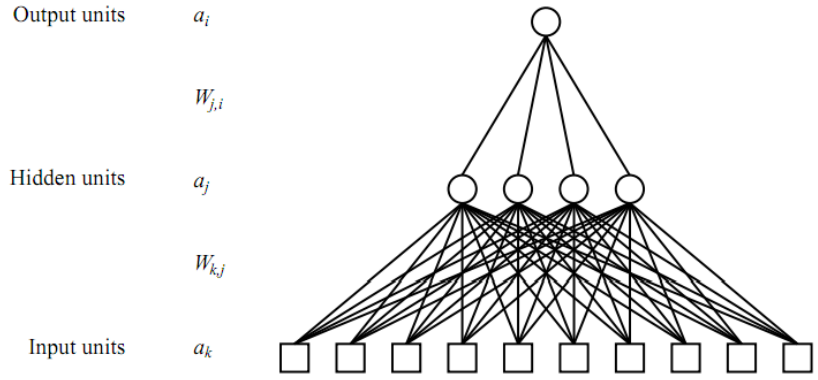
```
// Propagate input patterns through the network
procedure ForwardPropagate
    for each layer of the network do
        for each neuron in the current layer do
            1. Compute the sum of the weighted input values
            2. Add the weighted bias
            3. Compute the value of the activation function
        end
    end
end
```

The `BackPropagate` Procedure

```
// Back-propagate the error through the network
procedure BackPropagate
    for each neuron in the output layer do
        compute the actual error
    end

    for each hidden layer do
        for each neuron in the current layer do
            1. Compute the fraction of the error for the neuron
            2. Update weights using the computed fraction of
               the error
        end
    end
end
```

Update Rule (I)

- *Back-propagation*
  - The update rule of SLPs is used for the output layer



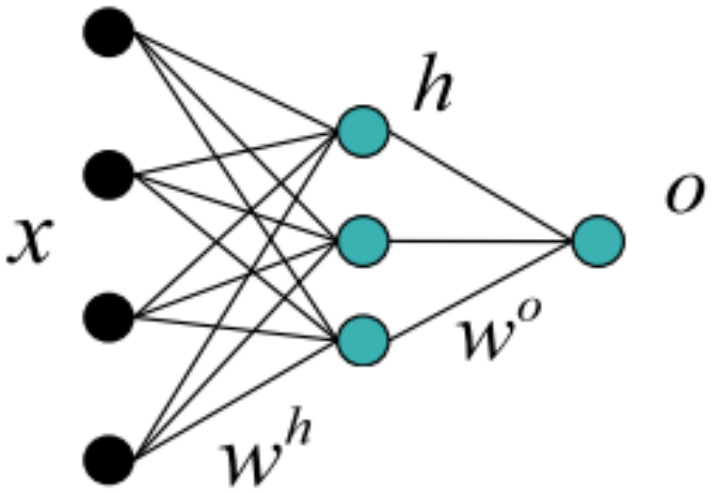
$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \qquad \Delta_i = Err_i \times g'(in_i)$$

  - The error is propagated backward

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \qquad W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

Update Rule (II)

- If only one hidden layer is used and the activation function is the logistic function

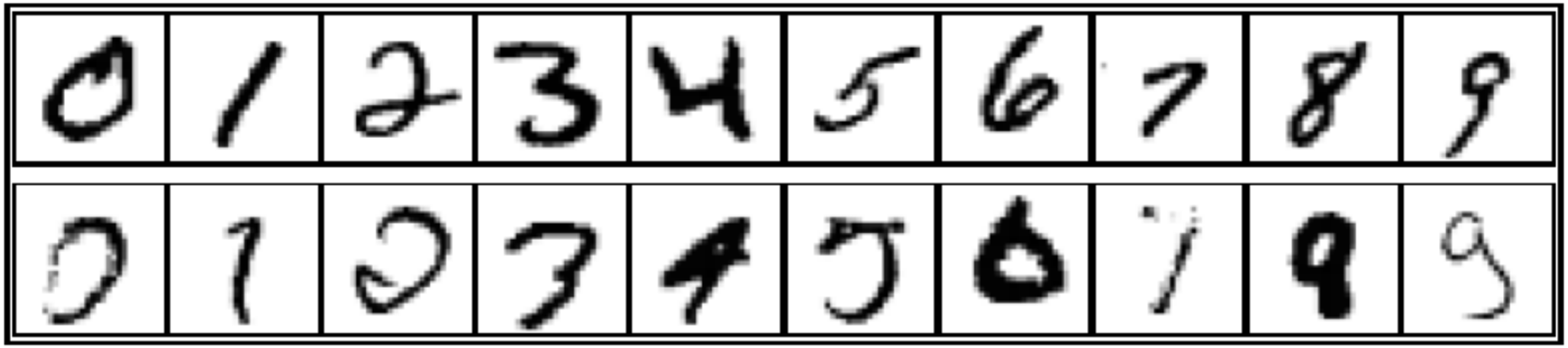


$$\Delta w_j^o = \alpha \delta^o h_j \qquad \delta^o = (y - o) \cdot o \cdot (1 - o)$$

$$\Delta w_{ij}^h = \alpha \delta_j^h x_i \qquad \delta_j^h = \delta^o \cdot w_j^o \cdot h_j \cdot (1 - h_j)$$

Handwritten Character Recognition

- MLPs can be effectively used to recognize handwritten characters



3-nearest-neighbor = 2.4% error

400–300–10 unit MLP = 1.6% error

Constraint Satisfaction Problems

"Were you to ask me which programming paradigm is likely to gain most in commercial significance over the next 5 years I'd have to pick Constraint Logic Programming (CLP)…"

BYTE, 1992

CSPs (I)

- Constraint Satisfaction Problems (CSPs) form a class of problems that is inherently related to the class of *search problems*
  - Search problems over trees
  - Search problems over graphs
- The class of CSPs can be understood as a specific class of search problems that is
  - *Sufficiently general* to include several interesting problems
  - *Sufficiently specific* to enable efficient solving algorithms and general-purpose *heuristics*

CSPs (II)

- A *CSP* is a triple
  - A finite and nonempty set of *variables*

$$V = \{ X_1 , X_2, \ldots, X_n \}$$

  - A finite and nonempty set of domains, one for each variable

$$D = \{ \text{dom}(X_1), \text{dom}(X_2), \ldots, \text{dom}(X_n) \}$$

  - A finite set of *constraints* over variables such that a constraint C is

$$C \subseteq \text{dom}(X_1) \times \text{dom}(X_2) \times \ldots \times \text{dom}(X_n)$$

CSPs (III)

- Normally, constraints are called
  - *Unary*, when they involve just one variable
  - *Binary*, when they involve two variables
  - *Global*, otherwise
- A *solution* to a CSP is a mapping that associates each variable X to a value in dom(X) such that all constraints are jointly *satisfied*
  - A CSP can have zero, one, or several solutions
  - Often, suitable restrictions of the domains of variables intended to include at least one solution are sufficient (*satisfiability testing*)

CSPs (IV)

- For example, consider the following CSP
  $V = \{ X_1, X_2 \}$
  $\text{dom}(X_1) = [1..100]$, $\text{dom}(X_2) = \text{dom}(X_1)$
  $X_1 < 5$, $X_1 + X_2 = 10$
- The first constraint ensures $\text{dom}(X_1) = \{ 1, 2, 3, 4 \}$
  But, for the second constratint $X_2 = 10 - X_1$
  And therefore, $\text{dom}(X_2) = \{ 6, 7, 8, 9 \}$
- The CSP has 4x4=16 possible solutions
  $(X_1=1,X_2=6)$, $(X_1=1,X_2=7)$, $(X_1=1,X_2=8)$, $(X_1=1,X_2=9)$
  $(X_1=2,X_2=6)$, $(X_1=2,X_2=7)$, $(X_1=2,X_2=8)$, $(X_1=2,X_2=9)$
  …

Solutions and Assignments

- An *assignment* is a mapping that associates some variables of the CSP with values taken from respective domains
- An assignment is
  - *Partial*, when it does not include all variables
  - *Inconsistent*, when at least one constraint is not satisfied
- A solution is a *complete* and *consistent* assignment
  - Therefore, solutions can be found by extending assignments in search for complete and consistent assignments

Types of CSPs

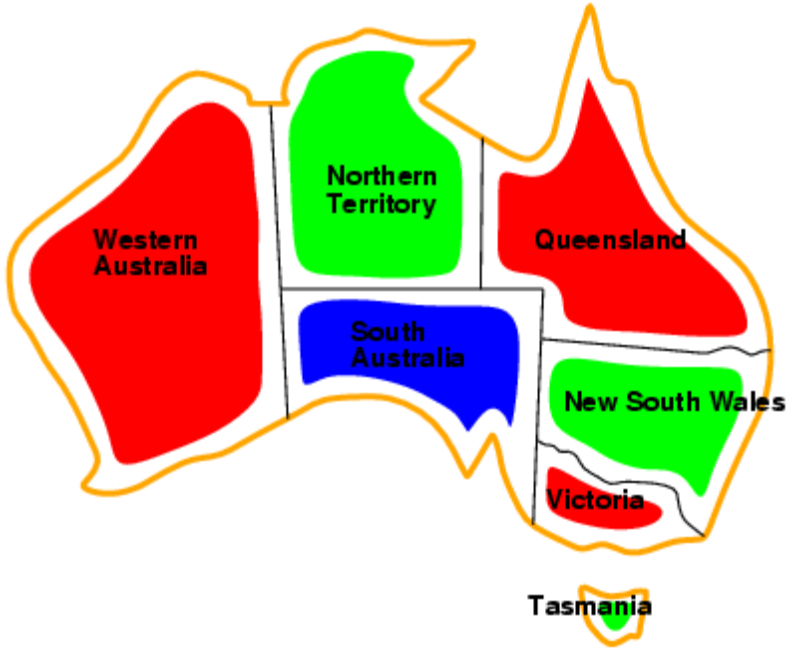
- CSPs with *discrete* domains
  - *Finite* domains with cardinality less that or equal to *d*
    - *n* variables, $O(d^n)$ complete assignments
  - *Countably infinite* domains
    - Natural numbers, integers, strings, …
- CSPs with *continuous* (*uncountably infinite*) domains
  - Real numbers, complex numbers, …
  - Constraints are normally expressed in terms of equations, inequalities, …

Map Coloring (I)

- Variables
  - WA, NT, Q, NSW, V, SA, T
- One domain
  - D = { R, G, B }
- Constraints
  - Neighboring zones must have different colors
  - WA ≠ NT, WA ≠ SA,
    NT ≠ SA, NT ≠ Q,
    SA ≠ Q, …

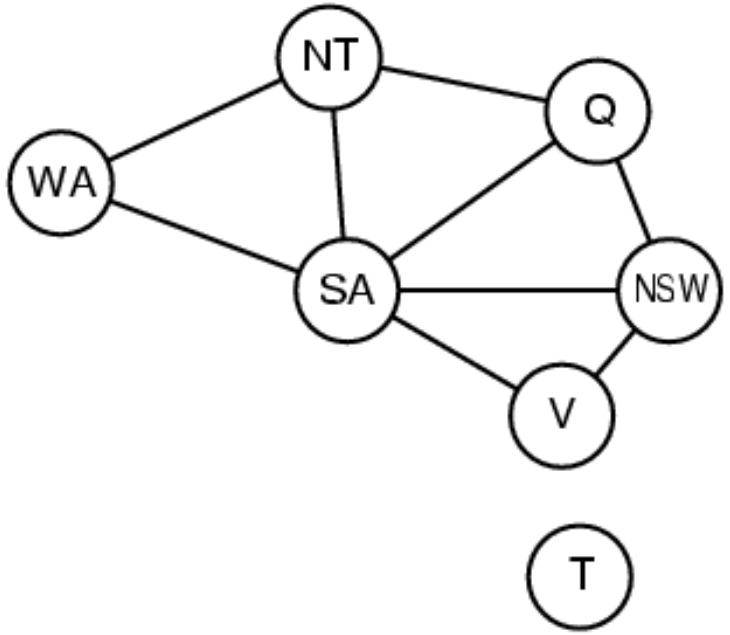
Map Coloring (II)

- Variables
  - WA, NT, Q, NSW, V, SA, T
- One domain
  - D = { R, G, B }
- Constraints
  - Neighboring zones must have different colors
  - WA ≠ NT, WA ≠ SA,
    NT ≠ SA, NT ≠ Q,
    SA ≠ Q, …



Constraint Graph (I)

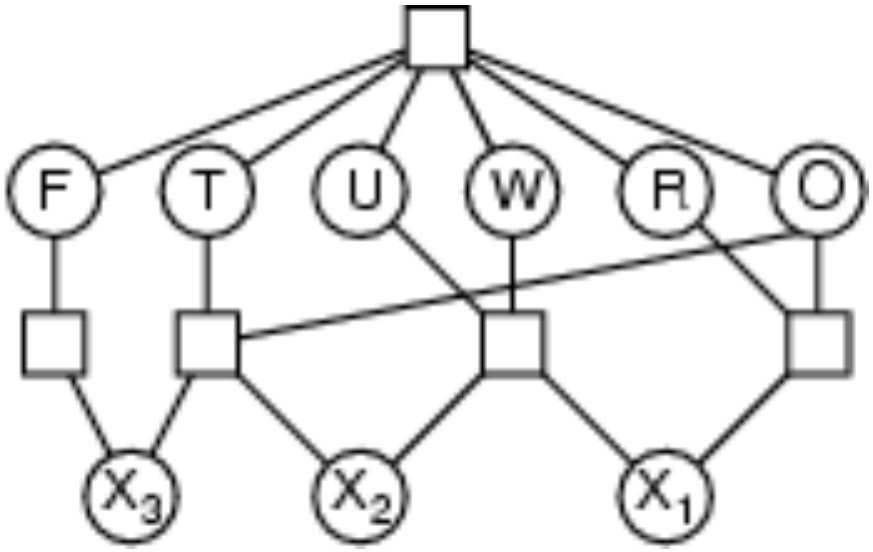
- A CSP is
  - *Unary,* if it has only unary constraints
  - *Binary,* if it has unary and binary constraints
- Given a binary CSP, it is always possible to define a corresponding constraint graph
  - It is an undirected graph
  - The nodes are the variables
  - The arcs are the constraints



Constraint Graph (II)

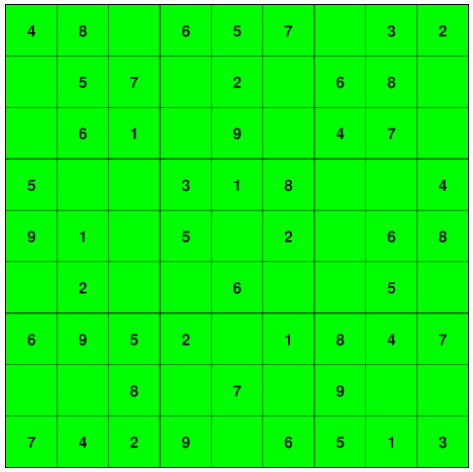
- Variables
  - $T\ W\ O\ F\ U\ R$
  - $X_1\ X_2\ X_3$
- One domain
  - D = [0..9]
- Constraints
  - $O + O = R + 10\ X_1$
  - $X_1 + W + W = U + 10\ X_2$
  - $X_2 + T + T = O + 10\ X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$
  - *alldifferent*(F,T,U,W,R,O)

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

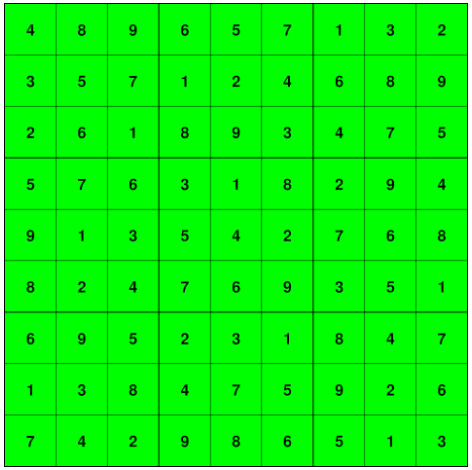


A Real-World Case: Sudoku Puzzles

- Variables
  - The 81 cells in the grid
- One domain
  - D = [1..9]
- Constraints
  - Values in each row do not repeat
  - Values in each column do not repeat
  - Values in each 3x3 block do not repeat
  - Some values are fixed



4	8		6	5	7		3	2
	5	7		2		6	8	
	6	1		9		4	7	
5			3	1	8			4
9	1		5		2		6	8
	2			6			5	
6	9	5	2		1	8	4	7
		8		7		9		
7	4	2	9		6	5	1	3



4	8	9	6	5	7	1	3	2
3	5	7	1	2	4	6	8	9
2	6	1	8	9	3	4	7	5
5	7	6	3	1	8	2	9	4
9	1	3	5	4	2	7	6	8
8	2	4	7	6	9	3	5	1
6	9	5	2	3	1	8	4	7
1	3	8	4	7	5	9	2	6
7	4	2	9	8	6	5	1	3

Real-World CSPs

- Assignment problems
  - For example, who teaches what class?
- Timetabling problems
  - For example, which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling

- Note that many real-world problems involve real-valued variables and are associated with optimization criteria

Generate & Test (I)

- A CSP can be solved searching for at least one complete and consistent assignment

- *Generate & Test (G&T)* algorithm
  - All partial assignments are generated until all complete assignments are found
  - Each complete assignment is tested for consistency
- Normally, the G&T algorithm is the least efficient algorithm because it tests for consistency very late (only for complete assignments)

Generate & Test (II)

```
void solveGT(int index) {
    if(index == n) {
        if(areConstraintNotViolated())
            solutionFound();
    } else {
        if(variable[index] != FREE)
            solveGT(index + 1);
        else {
            for(int i = 0; i < d; i++) {
                variable[index] = dom[i];

                solveGT(index + 1);
            }

            variable[index] = FREE;
        }
    }
}
```

Standard Backtracking (I)

- *Standard BackTracking (SBT)* algorithm
  - All partial assignments are possibly (not necessarily) generated
  - Partial assignments are tested for consistency
    - Inconsistencies can be tested even if only some variables are assigned
  - A solution is found as soon as a complete assignment is found
    - The test for consistency is coupled with the assignment of free variables

Standard Backtracking (II)

```
void solveSBT(int index) {
    if(index == n)
        solutionFound();
    else {
        if(variable[index] != FREE)
            solveSBT(index + 1);
        else {
            for(int i = 0; i < d; i++) {
                variable[index] = dom[i];

                if(areConstraintNotViolated())
                    solveSBT(index + 1);
            }

            variable[index] = FREE;
        }
    }
}
```

Standard Backtracking (III)

- If the CSP has *n* variables, the solutions are at depth *n* in the *search tree*
  - A *depth-first search (children nodes are explored before the parent node)* can be used
  - Depth-first search uses linear memory usage
  - Depth-first search terminates if the tree is finite
- Each node of the search tree corresponds to the assignment of one variable
  - *Branching factor b=d*, where *d* is the cardinality of domains
  - Therefore, the tree has d^n leafs

Standard Backtracking (IV)

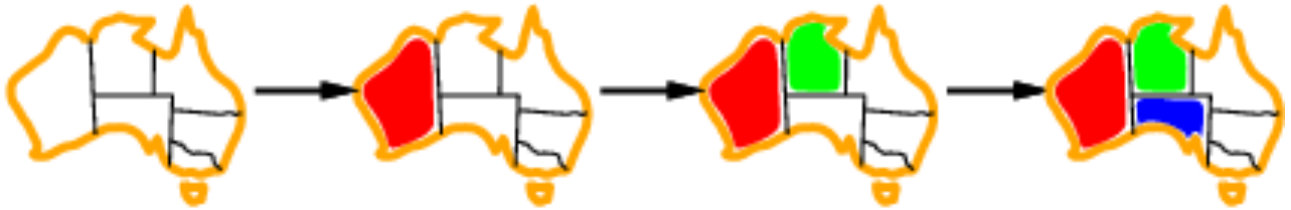
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

Search Heuristics

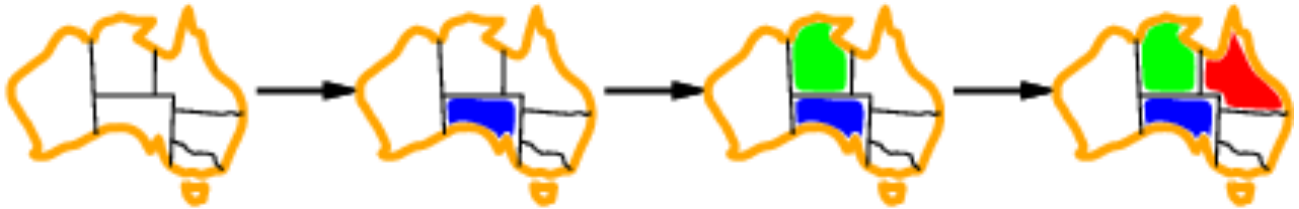
- The effectiveness of SBT can be improved using general-purpose search heuristics to
  - Choose which variable to use in next assignment
    - Function `SELECT-UNASSIGNED-VARIABLE`
  - Choose which value to use in next assignment
    - Function `ORDER-DOMAIN-VALUES`
  - Identify inconsistencies as soon as possible
    - The sooner inconsistencies are identified, the better it is
  - Choose which variable to set free in case of backtracking
    - These heuristics are not treated in this course, and only *chronological backtracking* is studied

Which Variable to Assign Next

- *Choose the variable with less values in the domain (min-domain variable)*

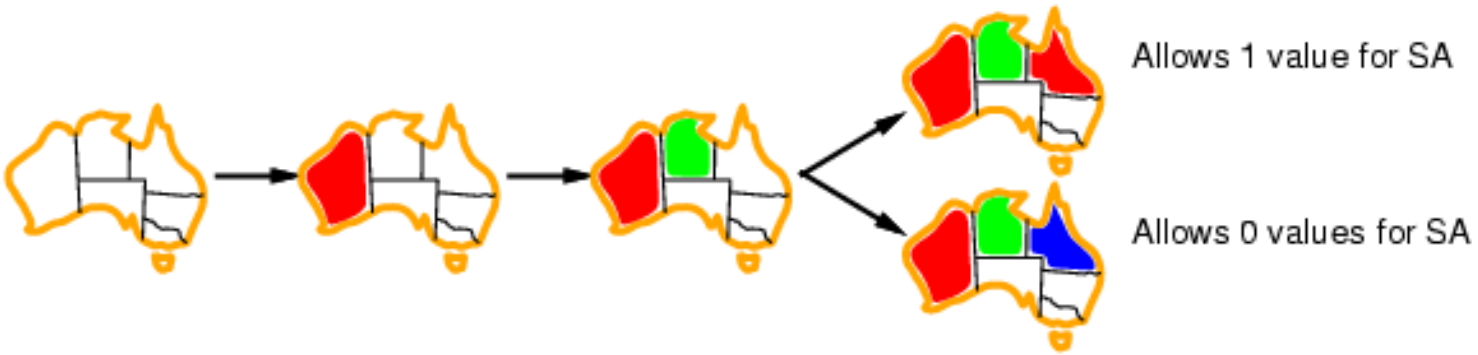


- *Choose the variable involved in more constraints (max-degree variable)*



Which Value to Assign Next

- Normally, min-domain variables are considered and one of the max-degree variables among them is finally chosen
- *Once a variable is chosen for the next assignment, the value that violates the fewest constraints is taken (min-conflict value)*

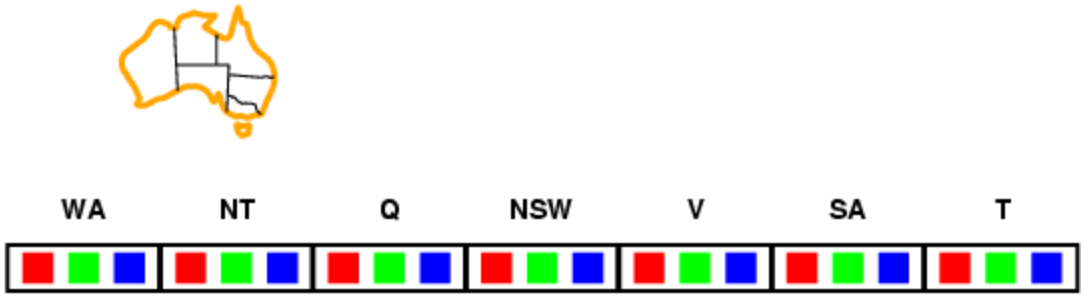


Constraint Propagation

- Given a CSP, the domains of variables can be reduced by reasoning on constraints
  - The process of reasoning on constraints to reduce (*filter*) the domains of variables is called *constraint propagation*
- Constraints can be propagated
  - At the beginning of the search to reduce the search space and better apply heuristics
  - After the assignment of a value to a variable to reduce the search space and better apply heuristics
  - Actually, the assignment of a value to a variable is nothing but the addition of a new constraints that can be propagated

Forward Checking (I)

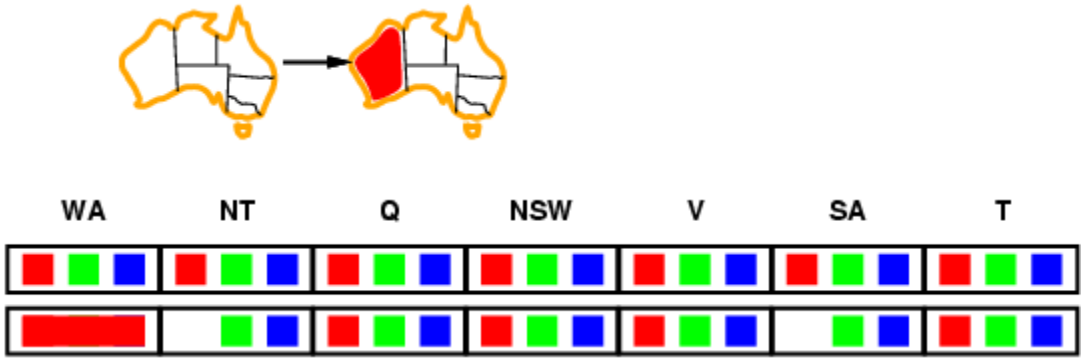
- The assignment of a value to a variable *X* can be used to remove inconsistent values from the domains of all variables that share constraints with *X*



- Backtracking occurs when the domain of a variable reduces to the empty set

Forward Checking (II)

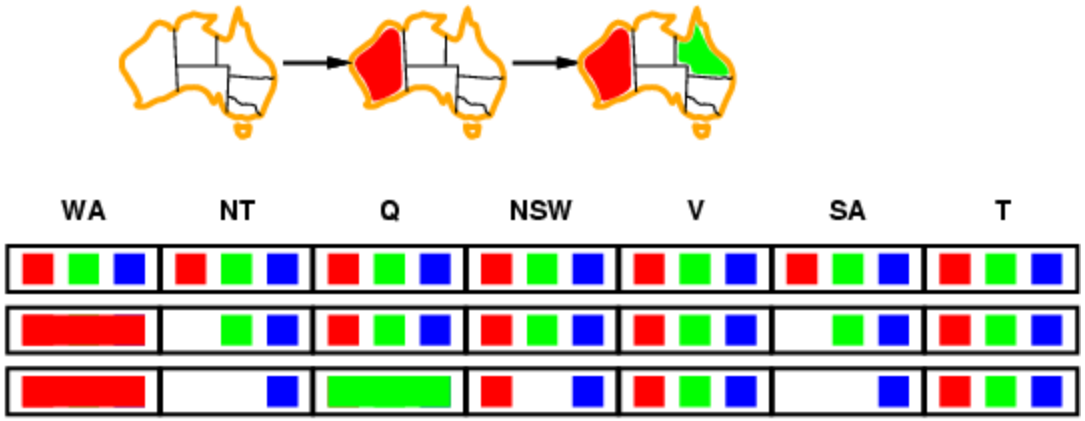
- The assignment of a value to a variable *X* can be used to remove inconsistent values from the domains of all variables that share constraints with *X*



- Backtracking occurs when the domain of a variable reduces to the empty set

Forward Checking (III)

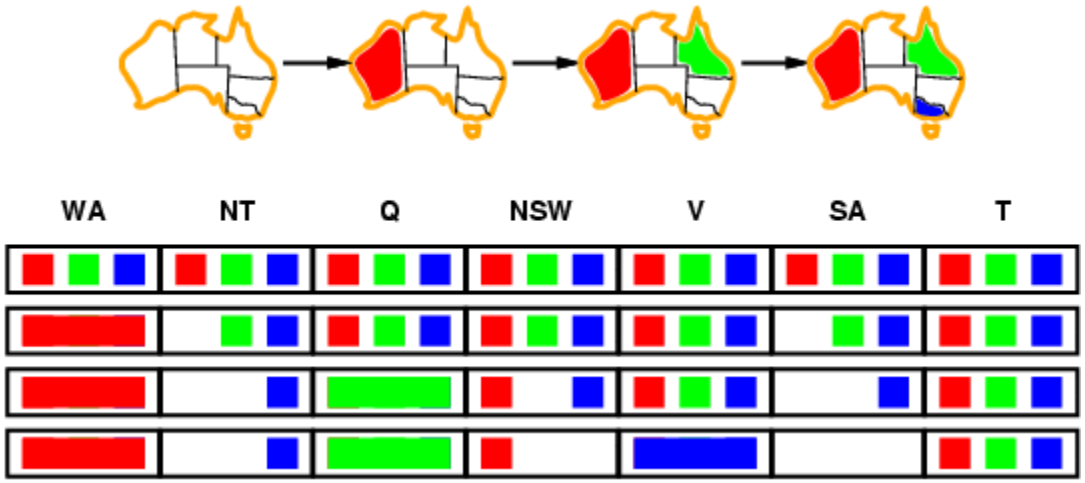
- The assignment of a value to a variable *X* can be used to remove inconsistent values from the domains of all variables that share constraints with *X*



- Backtracking occurs when the domain of a variable reduces to the empty set

Forward Checking (IV)

- The assignment of a value to a variable *X* can be used to remove inconsistent values from the domains of all variables that share constraints with *X*



- Backtracking occurs when the domain of a variable reduces to the empty set

Forward Checking (V)

- Forward checking is a simple constraint propagation algorithm that is not able to capture all inconsistencies
  - For example, NT and SA cannot be both colored blue

Arc Consistency (I)

- Forward checking propagates constraints only from the last assigned variable X to the free variables that share constraints with X
  - It does not further propagate the reduction of the domains to other variables
- Note that the reduction of the domain of a variable can be also considered as an added constraint
  - For example, if v is removed from the domain of X, then the added constraint is $X \neq v$
  - The *arc consistency algorithm* can be used to further propagate constraints

Arc Consistency (II)

- Note that
  - Given a generic CSP, an equivalent binary CSP can be found
  - Given a binary CSP, unary constraints can be enforced and removed
  - Therefore, a generic CSP can be always associated with a constraint graph that has no self-loops
- A binary CSP is called *arc consistent* if the domains of variables ensure that all arcs in the constraint graph are consistent
  - The directed arc $X \rightarrow Y$ is consistent if and only if for all values $x \in \text{dom}(X)$ there exists a value $y \in \text{dom}(Y)$ such that the constraint is satisfied
  - The undirected arc $X \text{—} Y$ is consistent if and only if $X \rightarrow Y$ and $Y \rightarrow X$ are both consistent

Arc Consistency (III)

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if RM-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
        add (X_k, X_i) to *queue*

---

function RM-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy constraint(X_i, X_j)
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

Arc Consistency (IV)

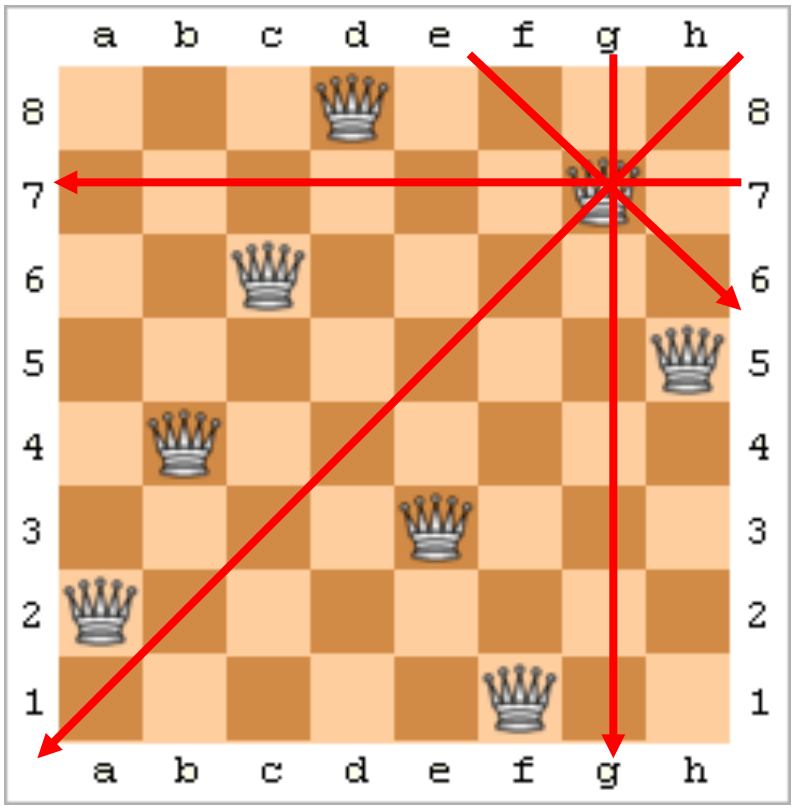
- The *MAC* (*Maintaining Arc Consistency*) algorithm is a variant of the SBT algorithm that enforces arc consistency before every assignment
  - It can use one of the several arc consistency algorithms available in the literature
  - It can benefit from appropriate search heuristics
- However, note that arc consistency does not capture all inconsistencies
  - It considers only binary constraints (in turns)
  - It can be generalized to *path consistency* to better capture inconsistencies

Parametric CSPs

- Parametric CSPs can be used to study the performance of solving algorithms
  - The number of variables of the CSP should be allowed to increase
  - The number of values in the domains of variables should be allowed to increase
  - The size of the CSP (number of variables and number of values in domains) should be controlled by a single parameter

- Parametric CSPs allows studying the asymptotic performance of studied algorithms both in terms of time and memory

n-Queens (I)



- A problem popularized by
  - *Carl Friedrich Gauss* (1777-1855) for *n*=8
  - *Edsger Dijkstra* (1930-2002), which used it to describe the SBT algorithm
- Given an *n*x*n* chessboard, the problem is to position *n* queens such that they do not lay
  - In the same row
  - In the same column
  - In the same diagonal

n-Queens (II)

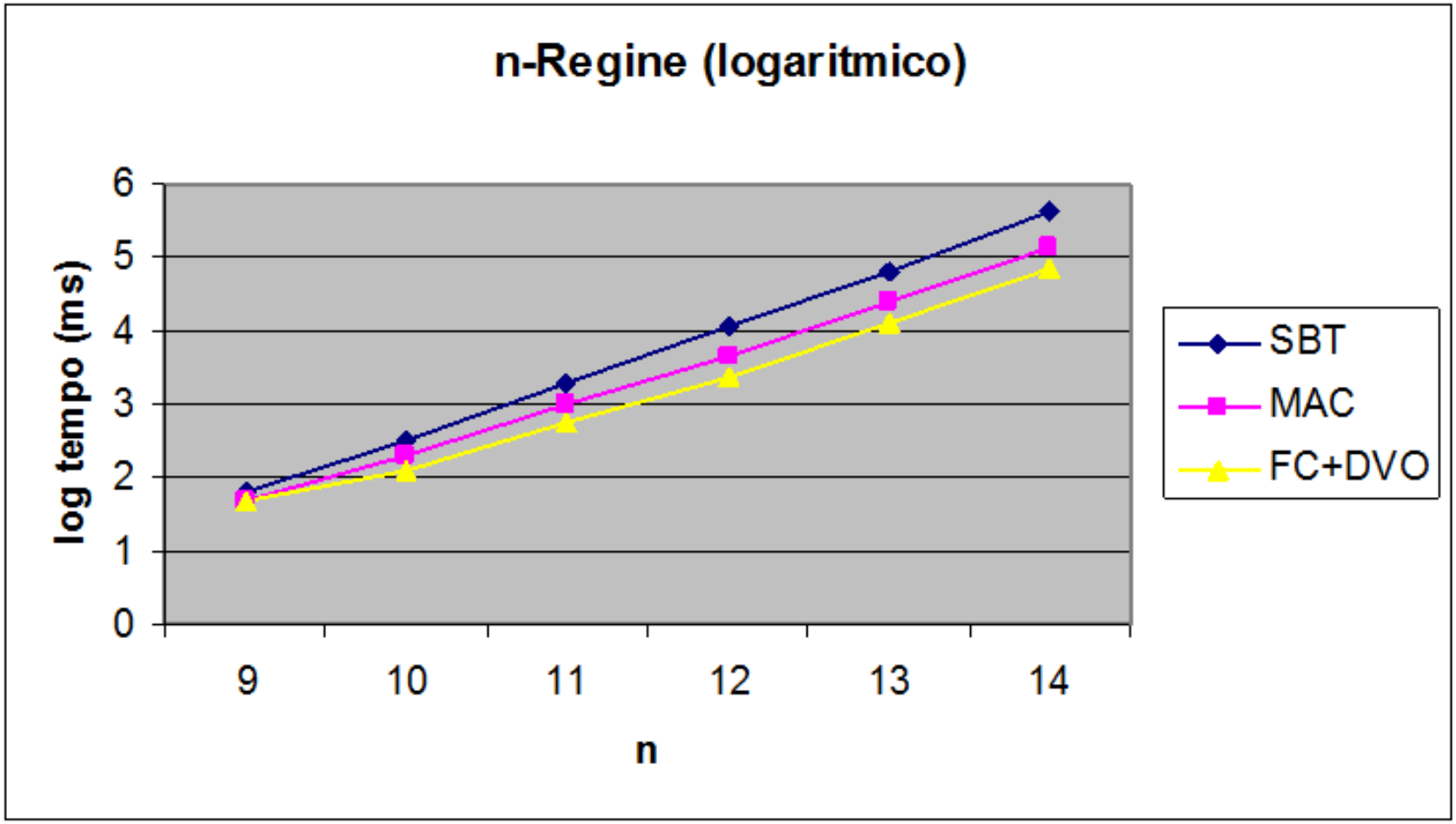
n	# solutions
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2,680
12	14,200
13	73,712
14	365,596
15	2,279,184

- The problem is not easy
  - The number of valid arrangements of the queens increases rapidly with *n*
  - But, for example, even if for *n*=8 there are 16,777,216 possible arrangements, only 92 of them are solutions
- Note that the *n*-queens problem is classic, but it is no longer considered representative of complex real-world problems

n-Queens (III)

n	G&T (ms)	SBT (ms)	MAC (ms)	FC+DVO (ms)
4	0	0	0	0
5	0	0	0	16
6	32	0	0	31
7	625	0	0	32
8	15.188	0	15	31
9	321.828	63	47	47
10	> 1 ora	328	204	125
11		1.953	984	547
12		12.016	4.391	2.328
13		62.172	24.219	12.547
14		427.906	143.203	72.047

n-Queens (IV)


n-Regine (logaritmico)
log tempo (ms)
0
1
2
3
4
5
6
9
10
11
12
13
14
n
SBT
MAC
FC+DVO

MiniZinc (I)

- MiniZinc is a language for the description of CSPs
  - It is independent of the adopted solving algorithm
  - It supports CSPs over the integers, the reals, the Booleans, and even over sets of them
- MiniZinc is a language of a family of languages that include
  - *Zinc*, which is a higher level language that can be effectively translated to MiniZinc
  - *FlatZinc*, which is minimalistic and used as a lower level alternative to MiniZinc
    - Note that MiniZinc is normally translated to FlatZinc

MiniZinc (II)

- The parts of a simple MiniZinc program are
  - Parameters of the CSP (can be overridden before execution)
    - `int: nc = 3;`
  - Variables of the CSP and their domains
    - `var 1..nc: wa;`
  - Constraints of the CSP
    - `constraint wa != nt;`
  - Type of problem (and optimization criteria, if needed)
    - `solve satisfy;`
    - `solve maximize 400*b + 450*c;`
  - Output procedure
    - `output ["wa=", show(wa)];`

Map Coloring in MiniZinc

```
% Map coloring
int: nc = 3;
var 1..nc: wa;
var 1..nc: nt;
…
constraint wa != nt;
…
solve satisfy;
output ["wa=", show(wa), …];
```

AILAB

Artificial Intelligence Laboratory

Artificial Intelligence

Prof. Federico Bergenti
Artificial Intelligence Laboratory
www.ailab.unipr.it/bergenti

www.ailab.unipr.it

Artificial Intelligence Laboratory
Università degli Studi di Parma
Parco Area delle Scienze 53/A
43124 Parma (Italy)

ailab@unipr.it