ECE449 Intelligent Systems Engineering

Petr Musilek

Fall 2019

Contents

Chapter 1

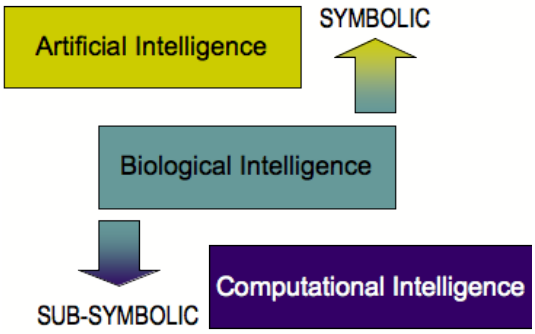
Introduction

1.1 Intelligent Systems

Intelligent systems
generally have a capacity to acquire and apply knowledge in an "intelligent" manner and have the capabilities of *perception*, *reasoning*, *learning*, and making inferences (or decisions) from *incomplete information*. [Karray, 2004]

Iintelligent agent
is any device that *perceives* its environment and takes *actions* that maximize its chance of successfully achieving its *goals*. [AI on wikipedia]

The A-B-C of Intelligence



(Biological) Intelligence

Biological Intelligence
represents "components of intelligence that can be directly attributed to the anatomy and physiology of the central nervous system. Biological intelligence is sometimes distinguished from artificial intelligence, i.e., intelligence demonstrated by computer behavior, and from psychometric intelligence or intelligence as documented by the performance of subjects on IQ tests." [Medical Dictionary, 2009]

- Difficult to define
- Usually described in terms of characteristics or capabilities
  - recognizes patterns
  - resourceful, smart

Artificial Intelligence (AI)

- The science of making machines do things that would require intelligence if done by men.

[Marvin Minsky, 1968]

- Automation of activities that we associate with human thinking: decision making, problem solving, learning

[Bellman, 1978]

- The mimicking of human thought and cognitive processes to solve complex problems

[Stottler, 1999]

Computational Intelligence (CI)

- Techniques able to deal with imprecise, incomplete, partially incorrect information in dynamically changing environments and large variable spaces.
- In contrast to the traditional AI, CI makes use of sub-symbolic (numerical) knowledge representation and processing.
- Includes: Artificial Neural Networks (NN), Fuzzy-Logic (FL), Evolutionary Algorithms (EA), etc.

Capabilities of CI methods

- Adaptivity;
- Fault tolerance;
- Speed of operation (close to the speed of human cognition processes, especially in the case of parallel processing); and
- Error rate optimality (relation of learning effort and error rates).

[Bezdek94]

Hard Computing vs. Soft Computing

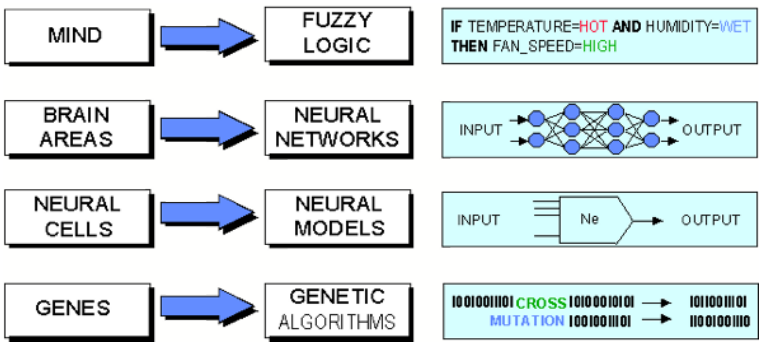
Hard Computing

- based on binary logic, crisp systems, numerical analysis and crisp software

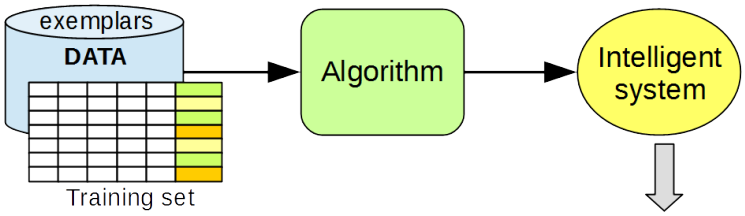
Soft Computing (coined by Zadeh)

- based on fuzzy logic, neural nets and probabilistic reasoning

Various Levels of Inspiration



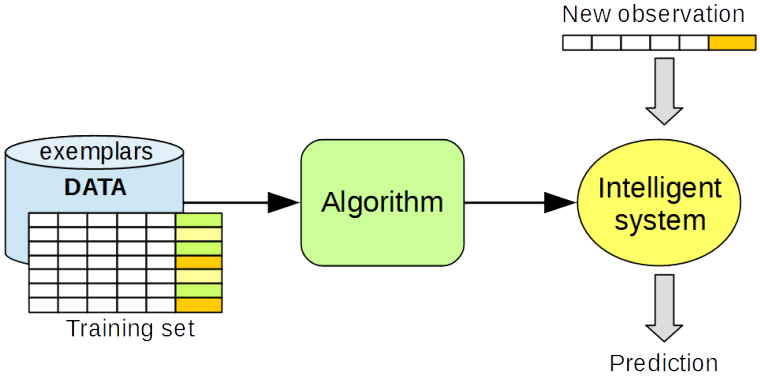
Use of Intelligent Systems



Phase 1/2: Development

Exemplars → Intelligent system (through algorithm)

Use of Intelligent Systems



Phase 2/2: Application

Intelligent system can be, e.g., predictor, classifier, controller

1.1.1 Fuzzy Sets

- Allow for imprecision, uncertainty, ambiguity
- Allow "computers to be less precise"
- Involve the "grey areas"
- Use gradual membership

Fuzzy "mathematics"

Extension of conventional mathematics to handle fuzzy constructs:

- Fuzzy logic
- Fuzzy relations
- Fuzzy arithmetic
- Fuzzy probability
- Fuzzy integrals

When can Fuzzy sets help?

- There is a high level of uncertainty, complexity; and/or non-linearity
- There is human expertise that cannot be easily expressed using analytical methods
- Problem can be described using qualitative terms

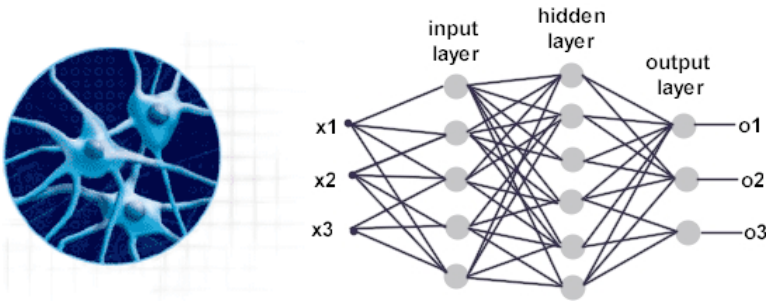
Note
FS are *transparent*, one can understand how system works from its structure and parameters.

Applications of Fuzzy Sets

- Areas that involve a high level of uncertainty, complexity, or nonlinearity - economics, data analysis, others.
- Decision-support and expert systems - medicine, management and financial decision-modeling
- Pattern recognition and classification - allowing belongingness to more than a single class
- Control - electronically stabilized camcorders, autofocus cameras, washing machines, air conditioners, transmissions, subway trains
- Modeling of uncertainty and its propagation - e.g. in maufacturing process, cost estimation

1.1.2 Neural Networks

NNs are based upon the parallel architecture found in animal brains



NN - A New Sort of Computer?
Conventional (von Neumann) computer systems are
Good at

- Fast arithmetic
- Doing precisely what the programmer programs them to do

Not so good at

- Interacting with noisy data or data from the environment
- Fault tolerance
- Adapting to circumstances
- Massive parallelism

What is a Neural Network?

- Different paradigm for computing:
  - Von Neumann machines are based on processing/memory abstraction of human information processing.
  - Neural networks are based on the parallel architecture of animal brains.

Note
Most NNs are opaque (black boxes); they do not provide much insight into *how* a problem is solved!

When can Neural Nets help?

- Cannot formulate an algorithmic solution.
- Can obtain lots of examples of the behavior/function/classification we require
- Need to derive the structure from existing data

Applications of Neural Networks

- Forecasting - weather, energy consumption, investment analysis (predicting stock movements, currencies etc., from historical data)
- Classification - sorting mail by ZIP/PC, occupancy classification
- Signature analysis - comparing signatures made (e.g. in a bank) with those stored (first large-scale applications of neural networks in the USA, also one of the first to use a neural network chip)
- Process control - most processes cannot be determined as computable algorithms, NNs can learn to control the process from data
- Monitoring/diagnostics - the state of aircraft engines or train diesel engines.

1.1.3 Evolutionary Computing

Mimics the processes of biological evolution

- natural selection
- survival of the fittest

to provide effective solutions for (optimization) problems

The first approach to EC - genetic algorithm (GA)

- Solution states encoded as binary-valued strings
- Population of candidate solution states is generated and evaluated
- Fitness function to evaluate the solutions
- Better solutions are recombined using genetic operators to form new solutions (i.e., offspring) which are generally more fit than the previous iteration (i.e., generation).
- The process is repeated until an acceptable solution is found within specific time constraints.

Other EC Paradigms

- Genetic programming,
- Evolution programming,
- Evolution strategies,
- Learning classifier systems.

EC approaches are helpful when

- Other optimization approaches have failed - often because the search space is too large.
- Although form of desired solution is not known, it is possible to evaluate quality of candidate solutions (fitness)

Applications of Evolutionary Computing

- Scheduling, resource allocation,
- Training of artificial neural networks
- Selecting or tuning rules for fuzzy systems
- PCB layout, circuit design
- Telecommunications (routing, switching, etc.)

Other Interesting Paradigms

- Artificial Immune Systems
  - learn to distinguish self from non-self
  - use concept of affinity (a.k.a. fitness)
- Swarm Intelligence
  - modeling of social animals: self-organization, no leader; flexibility, collective problem solving

Chapter 2

Fuzzy Systems

Fuzzy Sets: A Motivation

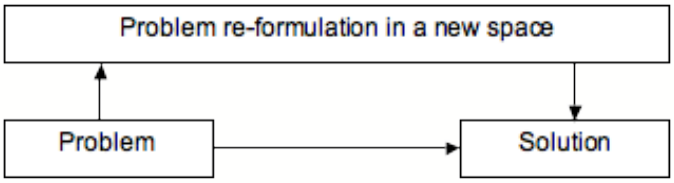
- Data processing: Making sense of data, linguistic summaries
- Decision-making and control problems: buying a car, air-conditioning
  - Decide on car purchase given brand, price, gas consumption, customer rating, etc.
  - Design a controller that maintains a comfortable room temperature
  - Design a highway traffic control system that assures safe driving environment
  - Design a system that can park a car
- Image processing and computer vision: selection of image processing algorithm, interpretation of a scene (image understanding)

- Domain-oriented, common sense knowledge:
  - if a region has color of skin, it is round, and located in the upper part of the filed of view, then confidence of face is high
  - if an object in the filed of view is moving, then use a short exposition time

Rule-based systems

- Expressing domain knowledge in a form of rules IF condition THEN conclusion
- Easy to understand and acquire
- Modular system
- Rules are generalizations of existing patterns of decision-making, classification, control, ...

Problem Solving



Examples of alternative spaces: Laplace, Fourier, fuzzy sets

2.1 Sets

2.1.1 Definition

Used to embrace elements to form some general concepts (granules)

- even numbers
- capital cities of Europe
- sport cars
- ...

but there are also situations like the following

2.1.2 Shortcomings

Sets, really?
... but there are also situations like the following

- large cities in Canada
- low temperature
- high inflation rate

and even terms like these

- small approximation error
- medium size software system
- fast response of a dynamic system
- ill-defined system of linear equations

Dichotomy

Another way of looking at problems associated with sets
One seed does not constitute a pile nor two ... from the other side everybody will agree that 100 million seeds constitute a pile. What therefore is the appropriate limit? Can we say that 325 647 seeds don't constitute a pile but 325 648 do? (Borel, 1950)

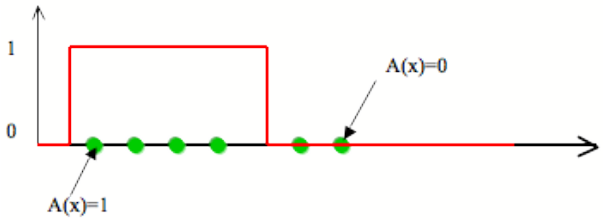
2.1.3 Description

Description of Sets

- Based on the concept of belongingness
  - inclusion $\in$, and
  - exclusion $\notin$
- Described by
  - inclusion - enumeration (characterization) of elements belonging to set *A*
  - characteristic function

Characteristic Functions

$$A: \ X \rightarrow \{0, 1\}$$



Sets subscribe to the concept of dichotomy:

$$x \in A \Leftrightarrow A(x) = 1$$
$$x \notin A \Leftrightarrow A(x) = 0$$

2.2 Fuzzy Stes

History of Fuzzy Sets

1920: J. Lukasiewicz, E. Post (three-valued logic and many valued logic)
1965: L. A. Zadeh (fuzzy sets)
1968: L. A. Zadeh (fuzzy algorithm)
1975: E.H. Mamdani (fuzzy control by linguistic rules)
1987: Fuzzy boom - Industrial applications of fuzzy sets in Japan & Korea

- Home electronics
- Vehicle control, process control
- Pattern recognition, image processing
- Expert systems
- Military systems, space research

1990s: Applications to very complex control problems (e.g. E.G. helicopter autopilot, Japan 1991)

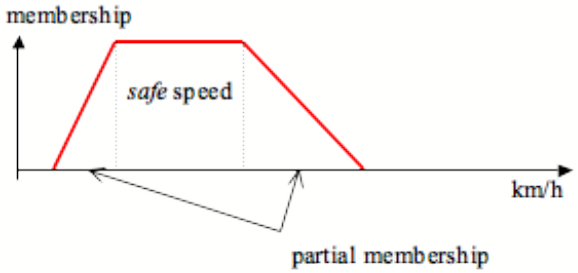
Definition:
Fuzzy set A is characterized by a membership function

Membership Functions:

- admit a notion of partial membership of element to the concept
- the higher the membership value $A(x)$, the more typical x is in A

2.2.1 Representation

Membership Function $A: \ X \to [0,1]$



Membership function describing concept "safe speed" on highway

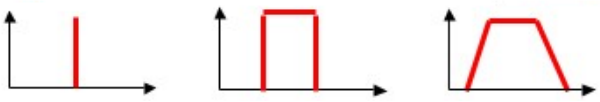
Example: belongingness
The universe of discourse is a group of people X.

Q1: Who has a driver's licence? – a crisp subset $A: \ X \to \{0,1\}$ (described by characteristic function)

Q2: Who can drive very well? – a fuzzy subset $A: \ X \to [0,1]$ (described by membership function)

Example: A control problem
Defining control objective

- single numeric setpoint
- interval (set-based) setpoint
- fuzzy set setpoint



Vector Representation
Vector Representation of Fuzzy Sets and Sets (in finite spaces)

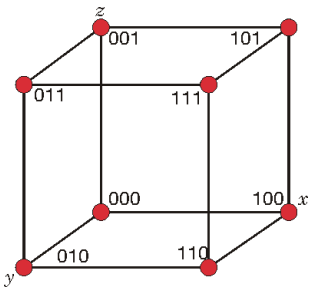
Sets: vectors with entries 0, 1

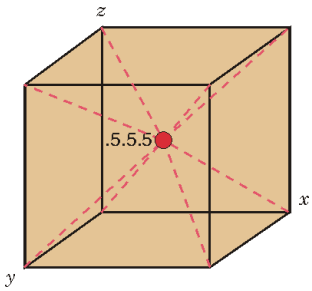
$$A = [0\ 0\ 1\ 1\ 0\ 0\ 0\ 1]$$

Fuzzy sets: vectors with entries in [0, 1]

$$B = [0.2\ \ 0.5\ \ 0.9\ \ 1\ \ 1\ \ 0.8\ \ 0.6\ \ 0.4\ \ 0.3]$$

Geometry of Sets and Fuzzy Sets





2.3 Sets vs. Fuzzy Sets

2.3.1 Operations

Operations on Sets

(Standard) Union	$(A \cup B) = \max\big(A(x), B(x)\big)$
(Standard) Intersection	$(A \cap B) = \min\big(A(x), B(x)\big)$
(Standard) Complement	$(\bar{A}) = 1 - A(x)$

Sets and Two-valued Logic

Sets	**Propositions**
inclusion	truth - assignment
$A(x) = 1$ (inclusion) $A(y) = 0$ (exclusion)	$t(p) = 1$ (true) $t(p) = 0$ (false)
operations	operations
$(A \cap B) = \min\big(A(x), B(x)\big)$ $(A \cup B) = \max\big(A(x), B(x)\big)$ $(\bar{A}) = 1 - A(x)$	$(A\ \&\ B) = A(x) \wedge B(x)$ $(A \mathrm{or} B) = A(x) \vee B(x)$ $t(\neg p) = 1 - t(p)$

Operations on Fuzzy Sets

Union	$(A \cup B) = \max\big(A(x), B(x)\big)$
Intersection	$(A \cap B) = \min\big(A(x), B(x)\big)$
Complement	$(\bar{A}) = 1 - A(x)$

Note:
This formulation is the same as for conventional sets, but:

- $A(x)$, $B(x)$ can attain values from $[0, 1]$ not just $\{0, 1\}$
- for std. operations, sets and fuzzy sets evaluate the same for (boundary) values of 0 and 1
- there are other possibilities besides the standard union, intersection, and complement

2.3.2 Properties

Properties of Sets and Fuzzy Sets (1/2)

Involution	$\overline{\overline{A}} = A$
Commutativity	$A \cap B = B \cap A$ $A \cup B = A \cup B$
Associativity	$(A \cap B) \cap C = A \cap (B \cap C)$ $(A \cup B) \cup C = A \cup (B \cup C)$
Distributivity	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
Idempotence	$A \cap A = A$; $A \cup A = A$
Absorption	$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$
Identity	$A \cup \emptyset = A$; $A \cap X = A$

Properties of Sets and Fuzzy Sets (2/2)

DeMorgan's Law	$\overline{(A \cap B)} = \overline{A} \cup \overline{B}$ $\overline{(A \cup B)} = \overline{A} \cap \overline{B}$
Law of contradiction	$A \cap \overline{A} = \emptyset$
Law of excluded middle	$A \cup \overline{A} = X$

Overlap and underlap of Fuzzy Sets
Law of contradiction does not hold: overlap property

$$A \cap \overline{A} \supseteq \emptyset$$

Law of excluded middle does not hold: underlap property

$$A \cup \overline{A} \subseteq X$$

Dichotomy Problem
"...the law of excluded middle is true when precise symbols are employed, but it is not true when symbols are vague, as, in fact, all symbols are." Russel, 1923

Illustration

2.4 Membership Functions

Membership functions are used to represent (capture) notions expressed in problem description:

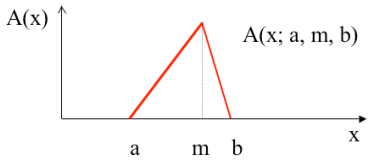
- elements with complete membership
- elements excluded from fuzzy set
- the form of transition between complete membership and complete exclusion

2.4.1 Types of MF

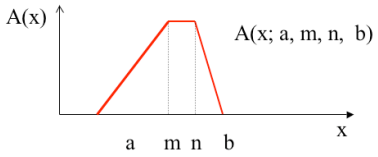
Triangular membership functions

Triangular fuzzy sets:

- modal value m
- lower and upper bounds a and b



Trapezoidal membership functions



Trapezoidal fuzzy sets:

- modal values m and n
- lower and upper bounds a and b

Other types of membership functions

Gaussian m.f.	$A(x) = e^{(-\frac{(x-m)^2}{\sigma}^2)}$
Exponential m.f.	$A(x) = \frac{k(x-m)^2}{1+k(x-m)^2}$
Sigmoidal m.f.	$A(x) = \frac{1}{1+e^{x-m}}$

Examples of fuzzy sets

Fuzzy sets represent concepts in real wolrd:

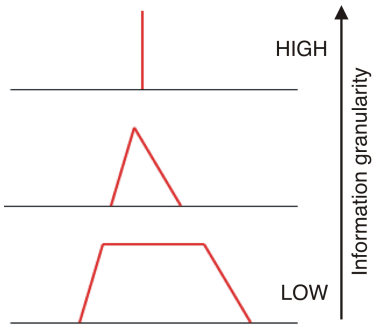
- Size
- Age
- Complexity
- Error
- Reliability
- Power consumption
- Microprocessor speed

Characteristics of Fuzzy Sets

Normality	$A(x)$: $\sup_x A(x) = 1$ (otherwise "subnormal")
Height	$\text{hgt}(A) = \sup_x A(x)$
Support	$\text{Supp}(A) = \{x \in X \mid A(x) > 0\}$
Core	$\text{Core}(A) = \{x \in X \mid A(x) = 1\}$
Cardinality	$\text{Card}(A) = \sum_{x \in X} A(x)$ (or $\int$ for continuous f.s.)

Hierarchy of fuzzy sets

Fuzzy sets representing the same concept but with different resolution lead to different levels of information granularity



Higher granularity: need more constructs (f.s.) to describe the universe of discourse

2.4.2 Operations on FS

Unary operations on fuzzy sets

Operations with single argument – a fuzzy set. They serve for manipulation of fuzzy sets to change their meaning or strength of the concept they represent (i.e. change membership function)

Normalization – converting subnormal fuzzy set into its normal counterpart

$$\text{Norm}(A) = \frac{A(x)}{\text{hgt}(A)}$$

Concentration – concentrating the membership function around points with higher membership values

$$\text{Con}(A) = A^2(x)$$

Dilatation – effect opposite to concentration

$$\text{Dil}(A) = A^{1/2}(x)$$

Of course, concentration and dilatation can be generally expressed in form

$$A^p(x)$$

with concentration corresponding to values $p > 1$ and dilatation to values $p < 1$.

Equality and Inclusion Relations

Two fuzzy sets A and B can be related with the following two relations

Equality:

$$A = B \text{ iff } \forall x : \ A(x) = B(x)$$

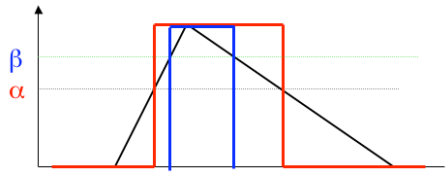
Inclusion:

$$A \subset B \text{ iff } \forall x : \ A(x) \leq B(x)$$

The Representation Theorem

Definition
α - cut of fuzzy set *A* is a set, A_α defined as

$$A_\alpha = \{x | A(x) \geq \alpha\};\ \text{if} \alpha < \beta \text{ then } A_\alpha \supseteq A_\beta$$



Theorem
Any fuzzy set can be represented as a family of sets

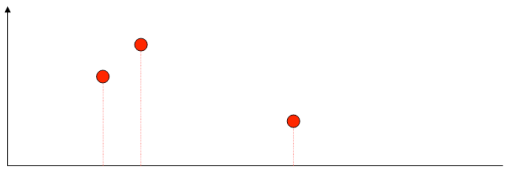
2.4.3 MF determination

Membership Function Determination
How to determine shape/functional description of membership from empirical data?

- Horizontal approach
- Vertical approach
- Pairwise comparison
- Clustering (grouping) method

Horizontal approach
Gather information about membership values at selected elements of the universe of discourse (space) X - polling mechanism (likelihood measure):

Can value x_i be accepted as compatible with given concept (fuzzy set A)?



Vertical approach
Identify α-cuts and "reconstruct" a fuzzy set using these sets:

What interval corresponds fully/partially/not at all to given concept?

Horizontal and Vertical Approaches

- Pro: easy to use
- Con: "local" character of experiments (isolated experiments dealing with single elements of the universe of discourse)

Pairwise Comparison Method
Rationale:

- assume that membership values $A_{x1}, A_{x2}, \ldots, A_{xn}$ are given
- arrange them as a reciprocal matrix $\boldsymbol{A}$
  - reciprocity $a_{ij} = 1/a_{ji}$
  - transitivity $a_{ik} = a_{ij}a_{jk}$
- multiply $\boldsymbol{A}$ by the vector of membership values $\boldsymbol{a}$

$$\begin{aligned} \boldsymbol{A}\boldsymbol{a} &= n\boldsymbol{a} \\ (\boldsymbol{A} - n\,\boldsymbol{I})\,\boldsymbol{a} &= 0 \end{aligned}$$

where I is a unit matrix, and $\boldsymbol{a}$ and n are eigenvector and eigenvalue of the matrix $\boldsymbol{A}$, respectively.

Realization:

- compare objects pair–wise in the context of $\boldsymbol{A}$ (e.g. ratio scale 1, 2, ..., 7) and construct the matrix based on these ratios
- assess consistency of the matrix (and, in turn, consistency of the gathered data) by looking at the value of n (should be comparable to the dimension of the matrix)
- normalize the eigenvector $\boldsymbol{a}$ to get an estimate of the membership function

2.5 Operations on Fuzzy Sets

2.5.1 Triangular norms

Standard (Fuzzy) Set Operations

Operation	Notation	Standard model
Union	$A \cup B$	$\max[A(x), B(x)]$
Intersection	$A \cap B$	$\min[A(x), B(x)]$
Complement	$\overline{A}$	$1 - A(x)$

Problems of standard models
Although the entire range [0,1] is available for the values of membership, these operations provide no interaction among the variables, i.e.

- union: no matter how small the other variable, the result of max operation is given by the larger value alone
- intersection: no matter how large the other variable is, the result of min operation is given by the smaller value alone

Alternative definitions of FS operations
Triangular norms and co-norms

- Triangular norms are operations that satisfy reasonable axioms for the definition of intersection and union; they were introduced in probabilistic metric spaces
- models of fuzzy set operations
  - Intersection: **t**-norms (triangular norm)
  - Union: **s**-norms (triangular co-norm)

Axioms (intersection)

Commutativity:	$x \,\mathbf{t}\, y = y \,\mathbf{t}\, x$
Associativity:	$x \,\mathbf{t}\, (y \,\mathbf{t}\, z) = (x \,\mathbf{t}\, y) \,\mathbf{t}\, z$
Monotonicity:	if $x \leq y$ and $w \leq z$ then $x \,\mathbf{t}\, w \leq y \,\mathbf{t}\, z$
Boundary Conditions:	$0 \,\mathbf{t}\, x = 0,\ 1 \,\mathbf{t}\, x = x$

Triangular Norms - models of intersection

Minimum **t**–norm	$x \,\mathbf{t}\, y = \min(x, y)$
Product **t**–norm	$x \,\mathbf{t}\, y = xy$
Lukasiewicz **t**–norm	$x \,\mathbf{t}\, y = \max(x + y - 1, 0)$
Drastic product **t**–norm	$x \,\mathbf{t}\, y = \begin{cases} 0 & \text{if } \max(x, y) < 1, \\ \min(x, y) & \text{otherwise.} \end{cases}$

Axioms (union)

Commutativity:	$x \,\mathbf{s}\, y = y \,\mathbf{s}\, x$
Associativity:	$x \,\mathbf{s}\, (y \,\mathbf{s}\, z) = (x \,\mathbf{s}\, y) \,\mathbf{s}\, z$
Monotonicity:	if $x \leq y$ and $w \leq z$ then $x \,\mathbf{s}\, w \leq y \,\mathbf{s}\, z$
Boundary Conditions:	$0 \,\mathbf{s}\, x = x,\ 1 \,\mathbf{s}\, x = 1$

Triangular Co-norms - models of union

Maximum **s**–norm	$x \,\mathbf{s}\, y = \max(x, y)$
Bounded sum **s**–norm	$x \,\mathbf{s}\, y = x + y - xy$
Lukasiewicz **s**–norm	$x \,\mathbf{s}\, y = \min(x + y, 1)$
Drastic sum **s**–norm	$x \,\mathbf{s}\, y = \begin{cases} 1 & \text{if } \min(x, y) > 0, \\ \max(x, y) & \text{otherwise.} \end{cases}$

Triangular Norms and Co-norms

Cannot be linearly ordered.
However, there are bounds on their values:

$$\text{drasticproduct} \leq \mathbf{t} \leq \min$$

and

$$\max \leq \mathbf{s} \leq \text{drasticsum}$$

Triangular Norms and Co-norms: Duality

For each t–norm, there exists a dual s–norm.
Corresponding to DeMorgan's laws:

$$x \,\mathbf{s}\, y = 1 - (1 - x)\,\mathbf{t}\,(1 - y)$$

and

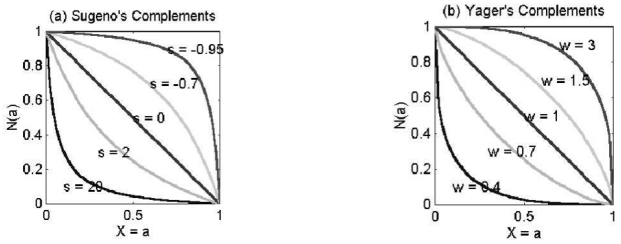
$$x \,\mathbf{t}\, y = 1 - (1 - x)\,\mathbf{s}\,(1 - y)$$

Axioms (Fuzzy Complement)

Monotonicity:	if $a < b$ then $\mathrm{N}(a) > \mathrm{N}(b)$
Involution:	$\mathrm{N}(\mathrm{N}(a)) = a$
Boundary Conditions:	$\mathrm{N}(0) = 1$ and $\mathrm{N}(1) = 0$

Fuzzy Complements

Standard fuzzy complement	$\mathrm{N}(a) = 1 - a$
Sugeno's fuzzy complement	$\mathrm{N}(a) = (1 - a)/(1 + sa)$
Yager's fuzzy complement	$\mathrm{N}(a) = (1 - a^w)^{1/w}$



2.5.2 Comparison operations

So far: set operations on fuzzy sets; i.e. how to combine two fuzzy sets and how to find a fuzzy set which complements another fuzzy set.
Now: comparison operations, to find how similar two fuzzy sets are; there are several possibilities to measure this:

- distance measures
- possibility measure
- necessity measure

Distance Measures

In general, distance between two fuzzy sets can be measured using their membership functions

$$d(A, B) = \sqrt[p]{\int_X |A(x) - B(x)|^p dx};\ p \geq 1$$

For different values of p, we obtain different measures, such as Hamming distance ($p = 1$), Euclidean distance ($p = 2$), Tchebyschev distance ($p = \infty$), etc.

Set based comparison operations
Calculation of distance involves two functions (membership functions A and B) – this measure therefore emphasizes functional aspect of fuzzy sets and not their set–based characteristic.

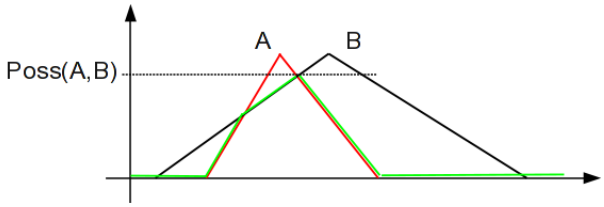
Set based comparison operations

- possibility
- necessity

alleviate this problem by taking into account set–based operations instead of functions.

Possibility Measure
Possibility measure of fuzzy set A with respect to fuzzy set B is defined as

$$\mathrm{Poss}(A, B) = \sup_{x \in X} \left[\min \left(A(x), B(x) \right) \right]$$



This measure quantifies the extent to which fuzzy sets A and B overlap.

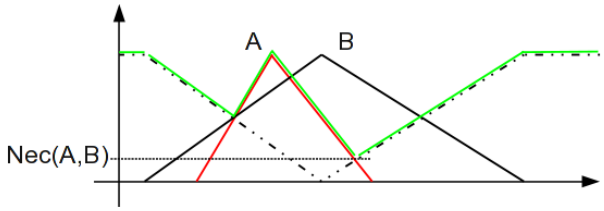
Possibility measure: main properties

$$\mathrm{Poss}(A, B) = \sup_{x \in X} \left[\min \left(A(x), B(x) \right) \right]$$

$$\mathrm{Poss}(A, B) = \mathrm{Poss}(B, A)$$
$$\mathrm{Poss}(A \cup B, C) = \max \left[\mathrm{Poss}(A, C), \mathrm{Poss}(B, C) \right]$$
$$\mathrm{Poss}(\{x\}, B) = B(x)$$

Necessity Measure
Necessity measure of fuzzy set A with respect to fuzzy set B is defined as

$$\mathrm{Nec}(A, B) = \inf_{x \in X} \left[\max \left(A(x), 1 - B(x) \right) \right]$$



This measure quantifies the extent to which fuzzy sets A is included in fuzzy set B.

Necessity measure: main properties

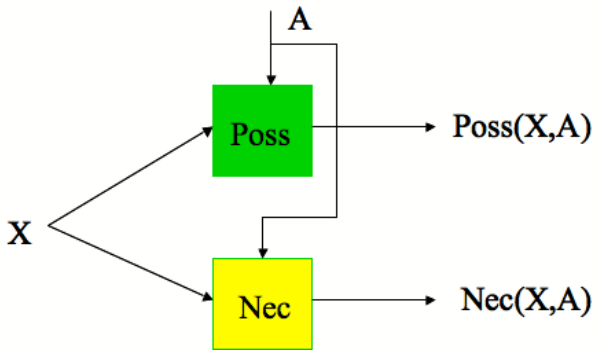
$$\mathrm{Nec}(A,B) = \inf_{x\in X}\left[\max\left(A(x), 1 - B(x)\right)\right]$$

$$\mathrm{Nec}(A,B) \neq \mathrm{Nec}(B,A)$$

$$\mathrm{Nec}(A\cap B, C) = \min\left[\mathrm{Nec}(A,C), \mathrm{Nec}(B,C)\right]$$

$$\mathrm{Nec}(\{x\}, B) = B(x)$$

Possibility and Necessity: a matching interface



Equality index
Consider two fuzzy sets A and B defined in the same finite space $X = \{x_1, x_2, \ldots, x_n\}$. Their equality ($\equiv$) can be assessed as follows:

$$A \equiv B : \ (A \subset B) \wedge (B \subset A)$$

Inclusion ($\subset$) can be modelled with implication ($\rightarrow$)

$$A \equiv B = \frac{1}{n} \sum_{i=1}^{n} \min \left[\big(A(x_i) \rightarrow B(x_i)\big), \big(B(x_i) \rightarrow A(x_i)\big) \right]$$

Operation of Implication
In case of fuzzy sets $a, b, \in [0, 1]$:

$$a \rightarrow b \ = \ \begin{cases} 1 & \text{if } a \leq b, \\ b & \text{if } a > b. \end{cases}$$

$$a \rightarrow b \ = \ \begin{cases} 1 & \text{if } a \leq b, \\ b/a & \text{if } a > b. \end{cases}$$

2.6 Fuzzy Relations

2.6.1 (Crisp) Relations

Relations

Definition
Let X and Y be two universes of discourse. A relation R defined as $X \times Y$ is any subset of the Cartesian product of these two universes:

$$R : X \times Y \rightarrow \{0, 1\}$$

If $R(x, y) = 1$, we say x and y are *related* (in sense of R). Otherwise, if $R(x, y) = 0$, x and y are *unrelated* (in sense of R).

Relations: Examples

Example 1. X is the domain of countries and Y is the domain of currencies. Let's define a relation R describing usage of currencies in countries. Then, for example

$$\begin{aligned} R(\text{Canada,CAD}) &= 1 \text{ while} \\ R(\text{Russia, CAD}) &= 0 \end{aligned}$$

Example 2. Equality relation $\text{Equal}(x, y)$ can be defined as $\{(x, y)|x = y\}$ for $(x, y) \in X$. Then for universe $X = \{1, 2, 3\}$, one can write

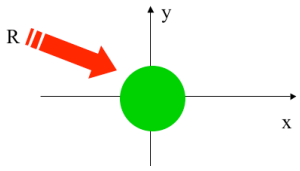
$\text{Equal}(1, 1) = 1, \text{Equal}(2, 2) = 1, \text{Equal}(3, 3) = 1$; and $\text{Equal}(1, 2) = 0, \text{Equal}(1, 3) = 0, \text{Equal}(2, 1) = 0, \text{Equal}(2, 3) = 0, \text{Equal}(3, 1) = 0, \text{Equal}(3, 2) = 0$

Relations: Representation

Relation can be also represented as a matrix

$$\text{Equal} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Relation can be also defined using an expression, e.g. $R(x, y) = \{(x, y)|x^2 + y^2 \leq r^2\}$ which defines a disc of radius r centered at (0,0).

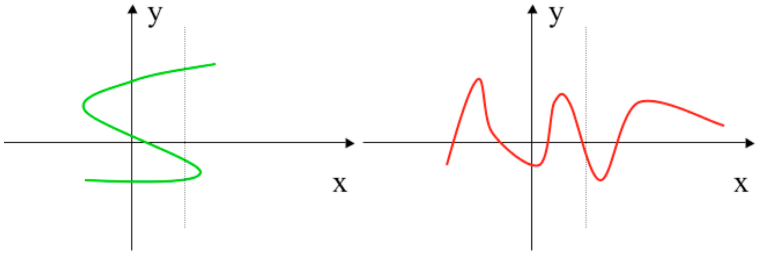


Relations vs. functions

Relations are more general than functions:

- Function – a unique y for each x
- Relation – (possibly) more y for each x

As a consequence, all functions are relations but not all relations are functions.



Relation Function

2.6.2 Fundamentals of Fuzzy Relations

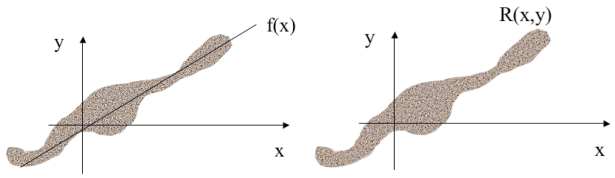
Noisy Function or Relation

From experiments, we usually get "clouds" of data.

- sometimes can be modelled with a function

- other times this is not possible because
  - condition of unique mapping not satisfied
  - causal relationship among variables not clear

In such cases, relation would be a more appropriate model of the data: it can express that data are related but does not attempt to identify independent or dependent variable.



Fuzzy Relations
A fuzzy relation R can be defined in a similar fashion as (crisp) relation, however the values of R are taken from the entire interval $[0, 1]$ instead of the binary values $\{0, 1\}$, i.e.

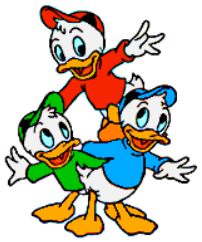
$$R : X \times Y \rightarrow [0, 1]$$

In addition to related/unrelated, fuzzy relation can also express a degree of relationship in sense of R.

Examples: x is much smaller than y x and y are approximately equal x is salary, y is age x is speed on highway, y is intensity of accidents

Similarity among the members of Duck family

	Dewey	Louie
R1=Huey	0.8	0.9



Fuzzy Relations - Basic Notions
Domain of R

$$\text{dom}(R)(x) = \sup_{y \in Y} R(x, y)$$

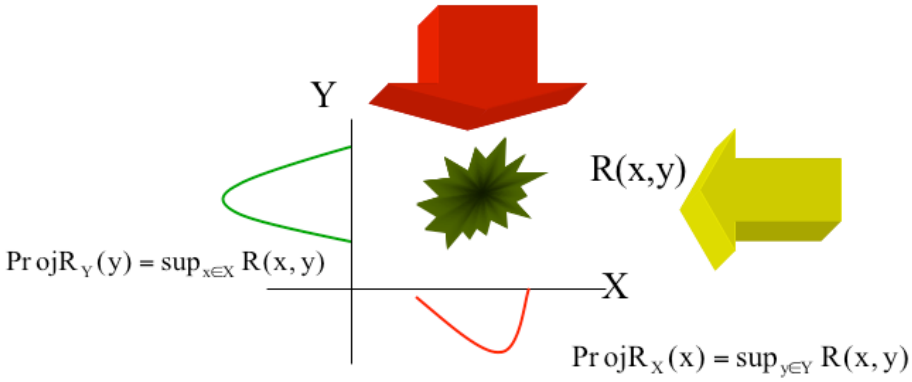
Co-domain of R (term co-domain is used instead of range to follow the concept of *direction-free* relation)

$$\text{co}(R)(y) = \sup_{x \in X} R(x, y)$$

α–cut of R (representation theorem; pyramids)

$$R = \cup_{\alpha \in (0,1]} (\alpha R_\alpha)$$

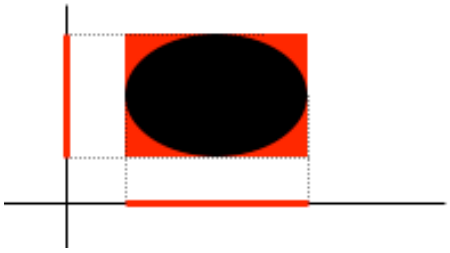
Projections of Fuzzy Relations



Projections of Fuzzy Relations

In general, the original relation R cannot be reconstructed from its projections, i.e.

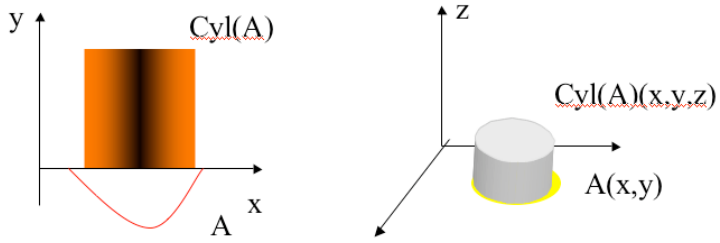
$$\mathrm{Proj}R_Y \times \mathrm{Proj}R_X \neq R \text{ but } \mathrm{Proj}R_Y \times \mathrm{Proj}R_X \supseteq R$$



In fact, only the 'envelope' of the original relation can be reconstructed. Reason: by projection, the data has been compressed (dimensionality has been reduced).

Cylindric Extension
The opposite of projection – increasing dimensionality of a fuzzy set/relation in order to be able to combine it with other construct of the same (higher) dimension.

$$\text{Cyl}(A)(x,y) = A(x) \text{ for all } y \in Y$$



Operations on Fuzzy Relations

Union
$(R \cup W)(x,y) = R(x,y)\, \mathbf{s}\, W(x,y)$

Intersection
$(R \cap W)(x,y) = R(x,y)\, \mathbf{t}\, W(x,y)$

Complement
$\overline{R}(x,y) = 1 - R(x,y)$

Transpose
$R^T(x,y) = R(y,x)$ with properties: $(R^T)^T = R$ $(\overline{R})^T = \overline{R^T}$

2.6.3 Compositions of Fuzzy Relations

Assume

$$\begin{aligned} R &: X \times Y \to [0,1] \\ G &: X \times Z \to [0,1] \\ W &: Z \times Y \to [0,1] \end{aligned}$$

Sup- t composition (e.g. max-min)
$R = G \circ W$
$R(x,y) = \sup_{z \in Z}[G(x,z)\ \mathbf{t}\ W(z,y)]$

Inf- s composition (e.g. min-max)
$R = G \bullet W$
$R(x,y) = \inf_{z \in Z}[G(x,z)\ \mathbf{s}\ W(z,y)]$

Example
Consider two relations G and W defined as follows

$$G = \begin{bmatrix} 1 & 0.6 & 0.5 \\ 0.7 & 0.1 & 1 \end{bmatrix}, W = \begin{bmatrix} 0 & 1 \\ 0.5 & 1 \\ 0.9 & 0 \end{bmatrix}$$

considering min operation in place of triangular norm $\mathbf{t}$, the max-min composition of these two relations is $G \circ W =$

$$= \begin{bmatrix} \max[\min(1,0), \min(.6,.5)\min(.5,.9)] & \max[\min(1,1), \min(.6,1)\min(.5,0)] \\ \max[\min(.7,0), \min(.1,.5)\min(1,.9)] & \max[\min(.7,1), \min(.1,1)\min(1,0)] \end{bmatrix}$$

Compositions of Fuzzy Relations: Interpretation

sup–**t** composition involves *matching* and *inference*

- Data in composed relations is matched using **t** operation
- Values in the non–overlapping regions ($X \times Y$) are then selected using sup operation (providing the inferred result)

inf–**s** composition involves *blending* and *compression*

- Data in composed relations is blended using **s** operation, reinforcing membership in overlapping region (Z) and projecting to the non–overlapping regions ($X \times Y$)
- The membership of the common region (Z) is compressed using the inf operation, ignoring the the non–overlapping regions ($X \times Y$)

Compositions of Fuzzy Relations: Properties

$R = G \circ W$ $\qquad$ $R = G \bullet W$

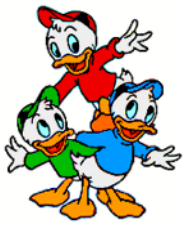
$R(x,y) = \sup_{z \in Z}[G(x,z) t W(z,y)]$ $\qquad$ $R(x,y) = \inf_{z \in Z}[G(x,z) s W(z,y)]$

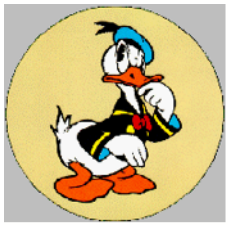
Associativity	
$R \circ (P \circ W) = (R \circ P) \circ W$	$R \bullet (P \bullet W) = (R \bullet P) \bullet W$
Distributivity	
$R \circ (P \cup W) = (R \circ P) \cup (R \circ W)$	$R \bullet (P \cap W) = (R \bullet P) \cap (R \bullet W)$
Weak distributivity	
$R \circ (P \cap W) \subseteq (R \circ P) \cap (R \circ W)$	$R \bullet (P \cup W) \supseteq (R \bullet P) \cup (R \bullet W)$
Monotonicity	
$P \subset W$ then $R \circ P \subseteq R \circ W$	$P \subset W$ then $R \bullet P \supseteq R \bullet W$

Note: Different (and confusing) notation is used in textbook. In these notes (and many other books), symbol $\circ$ is used for sup–**t** composition and symbol $\bullet$ for inf–**s** composition in general, i.e. without regard to particular triangular norm or co–norm used. In the textbook, symbol $\bullet$ is used for sup–product composition, while symbol $\otimes$ is used to denote inf–**s** composition.

Composition of Fuzzy Relations: Application

	Dewey	Louie
R1=Huey	0.8	0.9





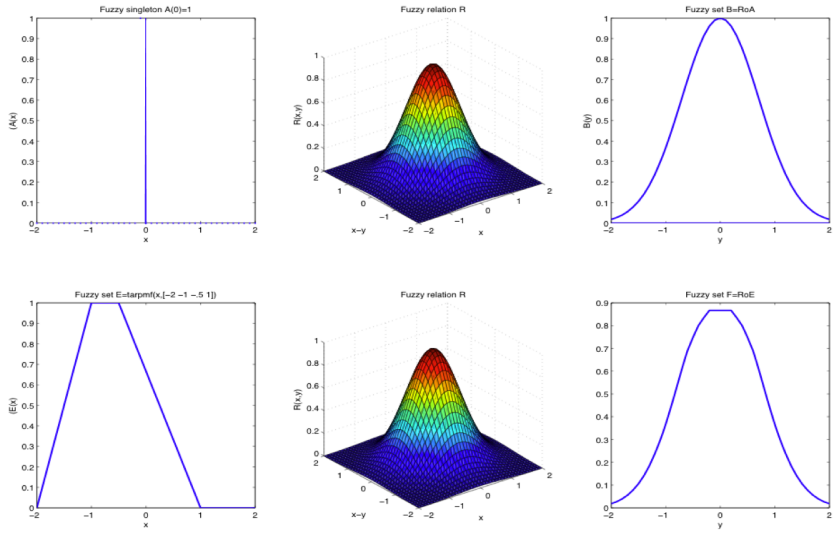
	Donald
R2= Dewey	0.9
Louie	0.7

Variants of Composition
Composition of two fuzzy relations is the most general case. However, there are other possibilities as well.

fuzzy relation	fuzzy relation	fuzzy relation
fuzzy set	fuzzy relation	fuzzy set
value	fuzzy relation	fuzzy set
value	(crsip) function	value
fuzzy set	(crisp) function	fuzzy set

Case of crisp function applied to fuzzy set is described by so called *extension principle* (leading to fuzzy numbers and fuzzy arithmetic).

Examples of Composition



Fuzzy rule–based systems
In knowledge based systems, knowledge can be expressed in form of rules

IF	*condition 1*	AND *condition 2*	THEN	*action*,	or alternatively
IF	*premise 1*	AND *premise 2*	THEN	*conclusion*,	or
IF	*antecedent 1*	AND *antecedent 2*	THEN	*consequent*	

In fuzzy systems, such rules are linguistic statements of expert knowledge in which *conditions* and *actions* are fuzzy sets (e.g. positive small, around zero, fast, etc.). These rules are *fuzzy relations* based on fuzzy implication (IF–THEN).

Inference as composition
A set of fuzzy rules forms a knowledge base of a fuzzy system. Let's denote K this collection of rules (i.e. K is the fuzzy relation describing knowledge about the system).
In fuzzy decision–making (e.g. fuzzy control), the rule base K is first matched with available data describing context in form of a fuzzy set D. Then, *inference* is made on another fuzzy variable represented in K. This matching–inference process is done using the $\sup$–$\min$ composition shown previously.

Compositional rule of inference (CRI)
Application of composition to make inferences:

$$I = D \circ K$$

Making inference
Membership function of the inference (conclusion, consequence, decision, action) can be determined using CRI

$$I(y) = \sup_{x \in X} \min \left[D(x), K(x, y) \right]$$

X denotes the space in which data (inputs) D are defined, and it is a subspace on which the knowledge (rule) base K is defined.

- K consists of rules containing AND connectives and fuzzy implication (IF–THEN), which can be both represented using $\min$ operation.
- Individual rules are connected by ELSE (corresponding to OR) connectives, which can be represented by $\max$ operation applied to membership of individual rules.

Example of making inference
Consider a control system

- Data (context) is described in terms of outputs, Y, of the controlled process
- The control action that drives the process is C (normally y and c are crisp, but let's consider them as fuzzy sets for now)
- Control rule–base is denoted R

Applying CRI we get the fuzzy control action C as

$$C(c) = \max_{Y} \min \left(Y(y), R(y, c) \right)$$

Composition vs. extension principle

- While, in general, *composition* applies to manipulating fuzzy data (fuzzy relations or sets) with fuzzy knowledge (another fuzzy relation),
- *extension principle* describes a special case of composition which applies to manipulation of fuzzy data (fuzzy sets) with crisp knowledge (crisp relation or function).

Consider a crisp relation $y = f(x)$. This may be considered a fuzzy relation R with the following membership function

$$R(x, y) = \begin{cases} 1 & \text{if } y = f(x), \\ 0 & \text{otherwise.} \end{cases}$$

Extension Principle
Now, consider a set of fuzzy data A with membership function $A(x)$. Using compositional rule of inference, one can obtain

$$B(y) = A(x) \circ R(x, y) = \sup_{x \in X} \min \left(A(x), R(x, y) \right)$$

Instead of finding supremum over the entire universe of discourse, we can consider two regions of the universe separately and then get the maximum of the two results

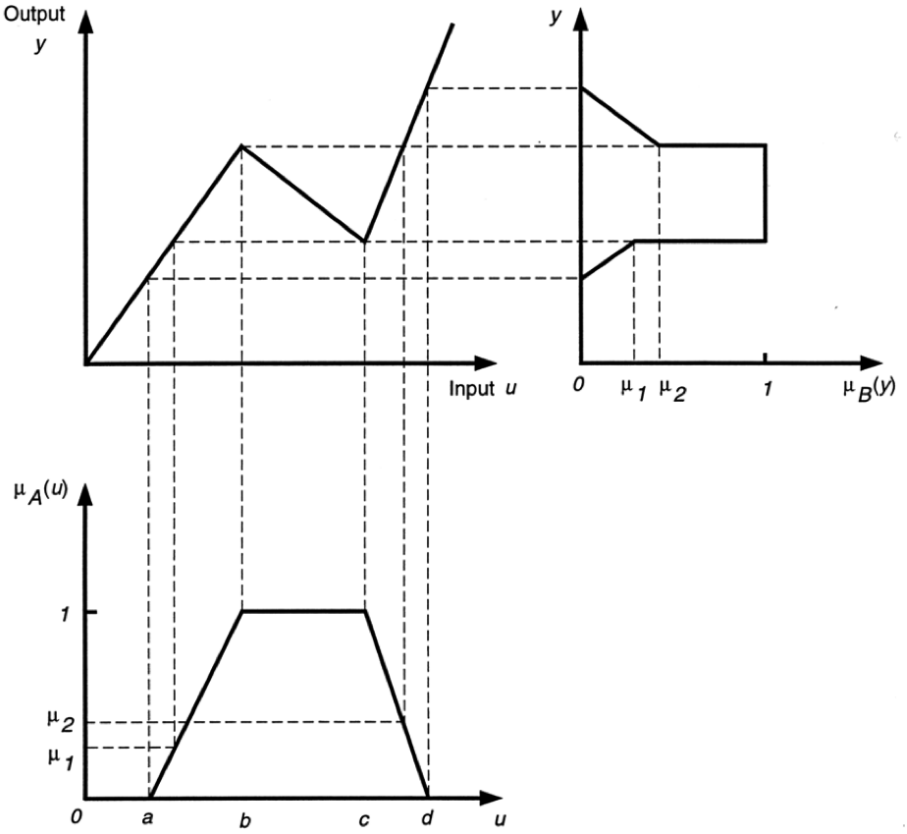
$$B(y) = \max \left[\sup_{x = f^{-1}(y)} \min \left(A(x), R(x, y) \right), \sup_{x \neq f^{-1}(y)} \min \left(A(x), R(x, y) \right) \right]$$

Region	sup (general)	$R(x,y)$	sup evaluates to
$x = f^{-1}(y)$	$\sup_x \min(A(x), R(x,y))$	1	$\sup_x \min(A(x), 1)$
$x \neq f^{-1}(y)$	$\sup_x \min(A(x), R(x,y))$	0	$\sup_x \min(A(x), 0)$

Clearly, the second term (for $x \neq f^{-1}(y)$) gives 0 and can be dropped entirely. The first term takes $\min$ and thus the value 1 can be dropped here. Finally, the following formulation of extension principle is obtained

$$B(y) = A(x) \circ R(x,y) = \sup_{x=f^{-1}(y)} \min(A(x))$$

Extension Principle: graphical method



2.7 Fuzzy Arithmetic

Fuzzy Numbers
Fuzzy sets defined over the domain of real numbers $\mathbb{R}$. They allow us to work with uncertain/fuzzy quantities, such as

- about 5
- smaller than 200
- close to 0.1

Example: Resistors are produced with certain tolerance around their nominal value of resistance, e.g. a 220k resistor has in fact resistance "about 220kΩ"

In turn, we may need to incorporate this uncertainty into our calculations

Fuzzy numbers

Fuzzy number is a fuzzy set

$$A : \mathbb{R} \to [0, 1]$$

that is

- normal,
- unimodal,
- continuous, and has

- bounded support

Interval analysis

Roots of computing with fuzzy numbers originate from interval analysis (a.k.a. calculus of tolerances). The basic objects of interval analysis are intervals I, such as $[1, 2]$, $[a, b]$, etc.

Basic arithmetic operations on intervals are:

$$\begin{array}{lcl} [a,b] & + & [c,d] = [a+c, b+d] \\ [a,b] & - & [c,d] = [a-d, b-c] \\ [a,b] & * & [c,d] = [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \\ [a,b] & \div & [c,d] = [\min(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d}), \max(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d})] \end{array}$$

Fuzzy arithmetic

Computing with fuzzy numbers leads to fuzzy arithmetic, which is based on extension principle. Consider a function F taking as arguments two fuzzy numbers A and B and producing a third fuzzy number C

$$C = F(A, B)$$

The extension principle determines m.f. of C as

$$C(z) = \sup_{x,y \in \mathbb{R}: z = f(x,y)} \big[A(x) \wedge B(y)\big]$$

Fuzzy arithmetic

where $z \in \mathbb{R}$, and $f : \mathbb{R}^2 \to \mathbb{R}$ is a real function that is point-wise consistent with F, i.e.

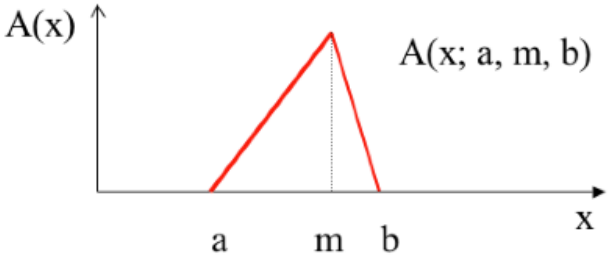
$$F\big(\{x\}, \{y\}\big) = f(x, y)$$

where $\{x\}$ and $\{y\}$ are fuzzy singletons corresponding to values x and y, respectively. The expression for extension principle is also called *sup-min convolution*.

Triangular fuzzy numbers

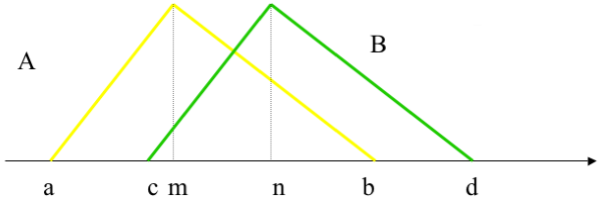
A triangular fuzzy number (TFN) is defined in terms of variable $x \in \mathbb{R}$ with parameters a, m, and b, describing lower boundary, modal value, and upper boundary, respectively

$$A(x; a, m, b) = \begin{cases} \frac{x-a}{m-a} & \text{if } x \in [a, m], \\ \frac{b-x}{b-m} & \text{if } x \in [m, b]. \end{cases}$$



Addition of TFNs

Consider two TFNs $A(x; a, m, b)$ and $B(x; c, n, d)$

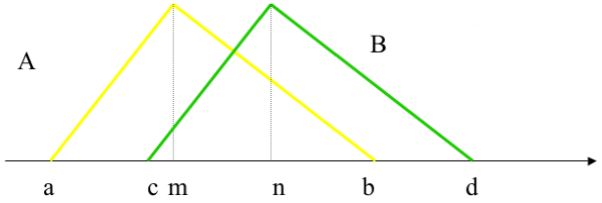


To derive the formula for fuzzy addition, the general form of the extension principle has to be parameterized by $f(x, y) = x + y$, i.e

$$C(z) = \sup_{x,y \in \mathbb{R}: z = x+y} \big[A(x) \wedge B(y)\big]$$

Multiplication of TFNs

Again, consider two TFNs $A(x; a, m, b)$ and $B(x; c, n, d)$



The general form of the extension principle has to be parameterized by $f(x, y) = xy$, i.e

$$C(z) = \sup_{x,y \in \mathbb{R}: z = xy} \big[A(x) \wedge B(y)\big]$$

As in case of addition (done in class), consider two cases:

a) $z < mn$, b) $z > mn$

a) Similarly to addition, we would consider such values of x and y for which memberships $A(x)$ and $B(y)$ are equal and to a constant ω, i.e.

$$A(x) = B(y) = \omega$$

ω can be expressed in terms of parameters/variables of fuzzy numbers A and B as follows

$$\frac{x - a}{m - a} = \omega, \quad \frac{y - c}{n - c} = \omega$$

The above expressions can be rewritten as $x = a + (m - a)\omega$ and $y = c + (n - c)\omega$, respectively. So far, this results is the same as for addition. However, since z is now xy, it can be expressed as follows

$$z = ac + \omega c(m - a) + \omega a(n - c) + \omega^2(m - a)(n - c),$$

i.e. z is a quadratic function of ω, and thus the resulting FN is no longer triangular.

[Note: Derivation of expression for case b) is similar]

Measures of fuzziness

Which set is fuzzier?

Degree of fuzziness is a measure of difficulty of ascertaining membership of elements of a fuzzy set:

- if the membership of an element is less than 0.5, the possibility of the element being outside of the fuzzy set is greater and hence it is easier to exclude it from the set
- if the membership of an element is greater than 0.5, the possibility of the element being inside the fuzzy set is greater and hence it is easier to include it in the set

Fuzziness: Simple energy index

$$F_0 = \int_{x \in \text{Supp}(A)} A(x)dx$$

Any problems here? Compare fuzziness of fuzzy set $A(x) = \text{trimf}(x, [0, 2, 4])$ and that of $B(x) = 1; x \in [1, 3]$?

$B(x)$ is a crisp interval nearest to $A(x)$ and

$$F_0(A) = F_0(B)$$

Fuzziness: Closeness to grade 0.5

The closeness to the most fuzzy grade

$$F_1 = \int_{x \in \text{Supp}(A)} f(x),$$

where the function $f(x)$ is defined in the support of the set A, such that $f(x) : \text{Supp}(A) \leftarrow [0, 0.5]$, i.e.

$$f(x) = \begin{cases} A(x) & \text{for } A(x) \leq 0.5, \\ 1 - A(x) & \text{otherwise.} \end{cases}$$

Fuzziness: Distance from 0.5-cut

$$F_2 = \int_{x \in \text{Supp}(A)} |A(x) - A_{0.5}(x)|dx,$$

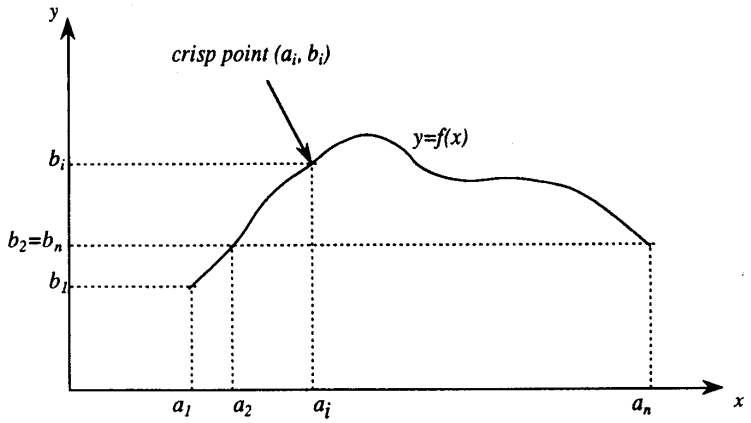
The value of $A_{0.5}$ is 0 for x where $A(x) < 0.5$, and $A_{0.5}$ is 1 for x where $A(x) \geq 0.5$. In effect, $F_1 = F_2$. Other measures have been proposed, e.g. quadratic energy index or inverse of distance from the complement (A_3 in the textbook, p.94)

2.8 Fuzzy Rule-based Computing

Rule-based Systems

Rules provide a formal way of representing knowledge in form of directives and strategies. Rule-based systems (RBS) are appropriate when domain knowledge is available as empirical results or experience. RBS also form basis for fuzzy rule-based systems (FRBS) and fuzzy control (see the previous section for a brief note on fuzzy RBS and their association with fuzzy composition).

To understand how an IF-THEN statement (and a set of IF-THEN-ELSE statements) can be represented as a relation, consider a crisp function.



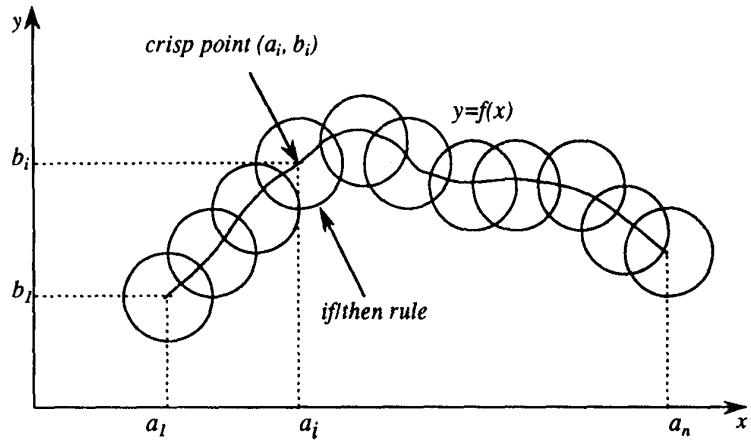
To say "$y = f(x)$" is an analytical form representing the function. Another way of representing this function would be to list all (or a sufficient number of) pairs that describe the function:

$$
\begin{array}{c}
(a_1, b_1) \\
(a_2, b_2) \\
\cdots \\
(a_i, b_i) \\
\cdots \\
(a_n, b_n)
\end{array}
$$

This form can be linguistically represented as

$$
\begin{array}{ll}
\text{IF } x \text{ is } a_1 & \text{THEN } y \text{ is } b_1 \\
\text{IF } x \text{ is } a_2 & \text{THEN } y \text{ is } b_2 \\
\cdots & \\
\text{IF } x \text{ is } a_i & \text{THEN } y \text{ is } b_i \\
\cdots & \\
\text{IF } x \text{ is } a_n & \text{THEN } y \text{ is } b_n
\end{array}
$$

which becomes more "crisply accurate" as n increases. But this is not always necessary (or even desired).
The above concept can be extended to fuzzy input/output (using extension principle)

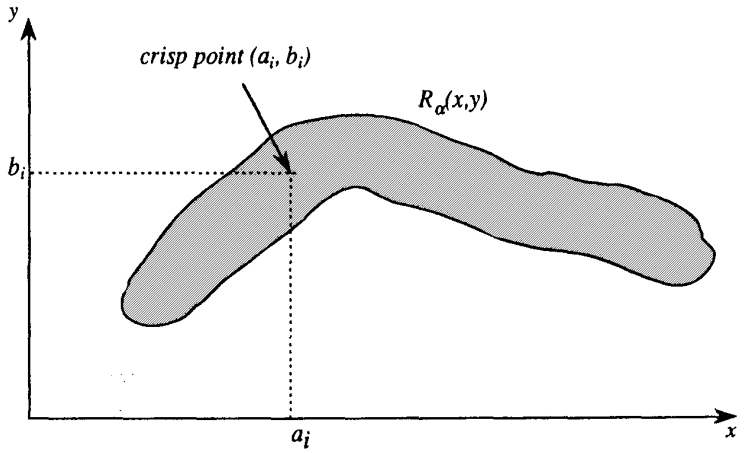


This new representation can be approximated using fuzzy rules as follows

IF x is A_1 THEN y is B_1
IF x is A_2 THEN y is B_2
...
IF x is A_i THEN y is B_i
...
IF x is A_n THEN y is B_n

Where A_i is a fuzzy quantity defined on X, and B_i is a fuzzy quantity defined on Y. The analytical form of these rules is a fuzzy relation $R_i(x, y) \rightarrow$ (implication relation, covered shortly), and each rule used to approximate the function has its own implication relation.

Merging the rules joined together by ELSE depends upon the manner by which the implication relations are created. Regardless the analytical form, this relation is called *fuzzy algorithm* (rather just a single rule).



Rules

Rule-based systems are based on IF-THEN statements in form

IF	*condition*	THEN	*action,* or alternatively
IF	*premise*	THEN	*conclusion*
IF	*antecedent*	THEN	*consequent*

Fuzzy Rules

Consider a rule in the form

IF x is A AND y is B THEN z is C

with x, y and z being fuzzy quantities; e.g.

IF *motor temperature* is *high* AND *motor speed* is *high* THEN *motor current* is *low*

Inference, Modus ponens

Rules are combined with facts (or collections of facts) to make inference. This leads to the sole rule of inference in propositional calculus, called *modus ponens* (from Latin: *mode that affirms*). Modus ponens can be formally expressed as.

$$\frac{\begin{array}{lll} p & & \\ p & \rightarrow & q \end{array}}{q}$$

Informally, this expression states that given an implication (rule, $p \rightarrow q$) and the presence of its premise p, the conclusion q can be taken as true.

Generalized modus ponens

In conventional logic, with modus ponens, the proposition *x* is *p* has to be observed to consider the proposition *y* is *b* . In other words, there has to be an exact match between the premise *p* and the antecedent of the rule *p*→*q*.

In fuzzy logic, a proposition *x* is *A*', close to the premise *x* is *A* can be observed to provide a conclusion *y* is *B*', close to the conclusion *y* is *B (A* and *B* are fuzzy stes). This leads to *generalized modus ponens* that can be formally represented as

$$\begin{array}{lll} A^{'} & & \\ A & \rightarrow & B \\ \hline & & B^{'} \end{array}$$

First, $A^{'}$ is matched with A. To find $B^{'}$, the implication relation $R(x, y)$ is composed with $A^{'}$ (this process is sometimes called *forward chaining* or *data-driven inference*)

$$B^{'} = R(x, y)A^{'}$$

Generalized modus tolens

GMT is essentially the reverse of GMP

$$\begin{array}{lll} & & B^{'} \\ A & \rightarrow & B \\ \hline A^{'} & & \end{array}$$

First, $B^{'}$ is matched to B. To find $A^{'}$, the implication relation $R(y, x)$ is composed with $B^{'}$

$$A^{'} = R(y, x)B^{'}$$

[Note that x and y are now swapped; this is analogous to finding and using $f^{-1}(y)$, the inverse of $f(x)$.]

Linguistic variables and values

Linguistic variables are described by fuzzy sets:

- Primary values of a fuzzy variable typically include terms like "small", "large", "high", etc.
- These can be further modified with *linguistic hedges* as "very", "more or less", "not", e.g.

$$\begin{aligned} \text{not}A(x) &= 1 - A(x) \\ \text{very}A(x) &= A^2(x) \\ \text{more or less}A(x) &= \sqrt{A(x)} \end{aligned}$$

Individual variables can also be connected with AND and OR, using appropriate triangular norms ($\mathbf{t}$ norms for AND, $\mathbf{s}$ norms for OR).

Accumulation and usage of knowledge

In fuzzy RBS, accumulation and usage of knowledge are facilitated by constructing relations and performing composition, respectively.

Accumulation of knowledge: It is evident that rules represent functional dependencies and relations between premises and conclusions. As such, they can be represented using Cartesian product. For example two fuzzy sets

$$A : \mathbf{X} \rightarrow [0, 1] \text{ and } B : \mathbf{Y} \rightarrow [0, 1]$$

can be combined using Cartesian product to form a relation $R : \mathbf{X} \times \mathbf{Y} \rightarrow [0, 1]$. This can be modelled by a **t**-norm operation, such as min.

Because the relation between the sets is generally not uniform, multiple rules/relations are build for multiple segments, k, of the universe of discourse

$$R_k = A_k \times B_k$$

To describe the overall relation, the individual rules R_k are aggregated

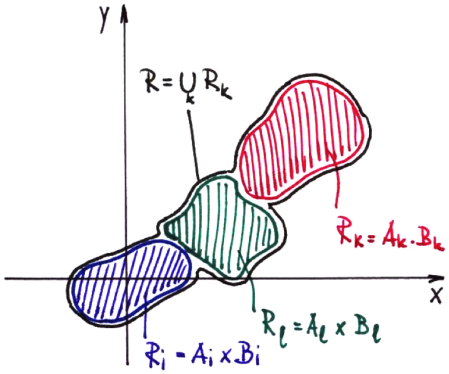
$$R = \cup_{k=1}^{N} R_k$$

This process corresponds to accumulation of knowledge and can be described (assuming min and max operation to model fuzzy intersection and union, respectively) as follows

$$R = \max_{k=1,2,\ldots,N} R_k(x, y) = \max_{k=1,2,\ldots,N} \min[A_k(x), B_k(y)]$$

Fuzzy patches

Graphically, this situation is depicted the following picture.



Accumulation and usage of knowledge

Usage of knowledge: Knowledge accumulated the way indicated above and codified using fuzzy rules can be used to make conclusions. Having described relation *R* between fuzzy sets *A* in $\mathbf{X}$ and *B* in $\mathbf{Y}$, the goal is to infer value of *B* given a particular value of *A*. Because the inference in fuzzy systems is driven by generalized modus ponens, exact match between *A* and one of the antecedentsA_k is not necessary. Using sup–**t** composition, fuzzy set *B* can determined as

$$B^{'} = A^{'} \circ R$$

Therefore, the resulting membership function is described as

$$B(y) = \sup_x [A(x) \, \mathbf{t} \, R(x, y)]$$

and because the relation $R(x, y)$ is a union of individual relations R_k, we can expand the above expression to (assuming min intersection and max union)

$$B(y) = \sup_x \min[A(x), R(x, y)] =$$

$$= \sup_x \left\{ \min \left[A(x), \max_k \left(\min \left(A_k(x), B_k(y) \right) \right) \right] \right\}$$

Rearranging this expression, we obtain

$$B(y) = \max_k \left\{ \min \left[\sup_x \left(\min \left(A(x), A_k(x) \right) \right), B_k(y) \right] \right\}$$

Term $\sup_x \left(\min \left(A(x), A_k(x) \right) \right)$ is expressing the level of overlap between the antecedent A_k of rule R_k, and the actual value of fuzzy set A, or in other words the possibility of A with respect to A_k

$$\text{Poss}(A, A_k) = \sup_x \left(\min \left(A(x), A_k(x) \right) \right)$$

By substituting $\lambda_k = \text{Poss}(A, A_k)$ we finally obtain the following expression to calculate fuzzy set B for given A

$$B(y) = \max_k \left\{ \min \left[\lambda_k, B_k(y) \right] \right\}$$

Based on max-min composition, this expression is called *compositional rule of inference* (CRI, cf. the brief note on CRI in the previous section).

Implication operators

The results presented in the previous section represent only one specific case of fuzzy implication relation, based on max and min operations. There are, however, numerous other possibilities how implication can be modelled.

There are two general ways to define fuzzy implications:

(i) $A \wedge B$ which corresponds to matching operation; it provides a stronger (local) implication meaning A is coupled with B; examples are Larsen and Mamdani implications (see bellow);

(ii) $\overline{A} \vee B$ which corresponds (NOT A) OR B implication found in binary logic; it provides a weaker (global) implication, meaning "A entails B"; examples include Lukasziewicz and Boolean implications;

(iii) $(A \wedge B) \vee \overline{A}$ which is equivalent to (ii) and has similar properties; an example is Zadeh implication

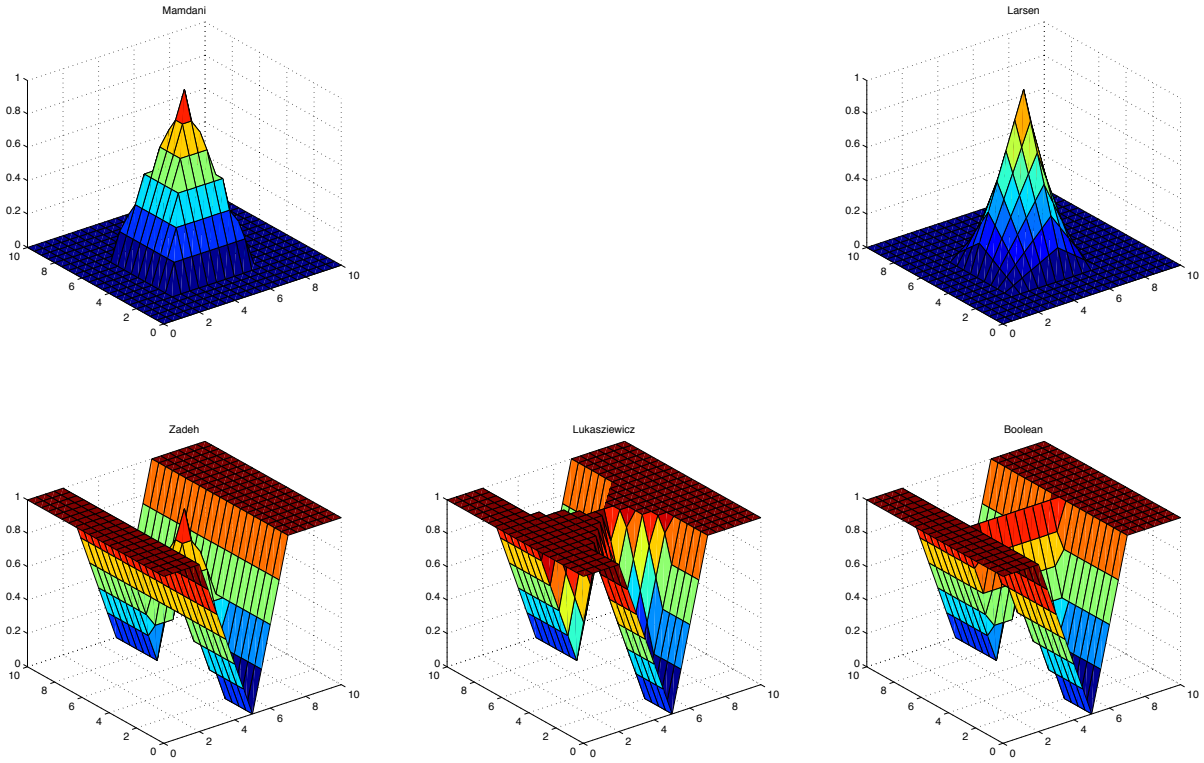
Rule IF x is A THEN y is B can be described in general analytical form of relation

$$R_{A \to B}(x, y) = \Phi[A(x), B(y)],$$

where Φ is the implication operator. A list of commonly used implication operators follows

Name	Implication operator $\Phi[A(x), B(y)]$
Mamdani (min)	$\min[A(x), B(y)]$
Larsen (product)	$A(x) \cdot B(y)$
Zadeh (max-min)	$\max\{\min[A(x), B(y)], 1 - A(x)\}$
Lukasziewicz (arithmetic)	$\min[1 - A(x) + B(y), 1]$
Boolean (Dienes-Rescher)	$\max[1 - A(x), B(y)]$

Fuzzy Implication Operators



Multivariate Implication

These elementary fuzzy implications can be extended to multivariate cases, in order to allow decision making or control with more than one independent variable. In such case, we consider a rule in the form

$$\text{IF } x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_2 \text{ AND} \ldots \text{AND } x_m \text{ is } A_m \text{ THEN } y \text{ is } B$$

The connective AND can be modeled either as min or product operations leading to implication operators in one of the following forms

$$\Phi\{\min[A_1(x_1), A_2(x_2), \ldots, A_m(x_m)], B(y)\}, \text{ or}$$
$$\Phi\{A_1(x_1)A_2(x_2)\ldots A_m(x_m), B(y)\}$$

.

2.8.1 Fuzzy Algorithm

A fuzzy algorithm is a procedure of collecting fuzzy IF-THEN rules. The rules are defined over the same universes (entering the Cartesian product of corresponding fuzzy sets), and are connected by connectives ELSE:

$$\begin{array}{ll}
\text{IF } x \text{ is } A_1 & \text{THEN } y \text{ is } B_1 \text{ ELSE} \\
\text{IF } x \text{ is } A_2 & \text{THEN } y \text{ is } B_2 \text{ ELSE} \\
\ldots & \\
\text{IF } x \text{ is } A_i & \text{THEN } y \text{ is } B_i \text{ ELSE} \\
\ldots & \\
\text{IF } x \text{ is } A_n & \text{THEN } y \text{ is } B_n
\end{array}$$

Each rule is represented by a relation whose form depends on particular implication operator, Φ, used. The form of ELSE connective, in turn, depends on the form of the implication, Φ.

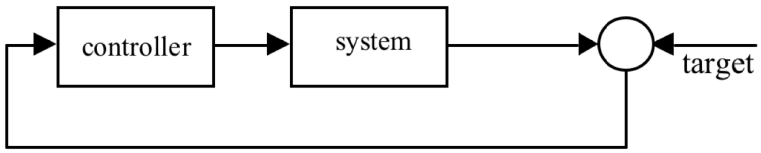
Implication operator and ELSE connectives

Name	$\Phi[A(x), B(y)]$	ELSE connectives
Mamdani	$\min[A(x), B(y)]$	OR (**s** -norm)
Larsen	$A(x) \cdot B(y)$	OR (**s** -norm)
Zadeh	$\max\{\min[A(x), B(y)], 1 - A(x)\}$	AND (**t** -norm)
Lukasziewicz	$\min[1 - A(x) + B(y), 1]$	AND (**t** -norm)
Boolean	$\max[1 - A(x), B(y)]$	AND (**t** -norm)

2.8.2 Fuzzy Control

Control Problem

Most typical control scheme is that of *direct control*, where the controller is in the forward path of a feedback control system.



Due to the diversity of systems and control objectives, design of controller is usually nontrivial and includes

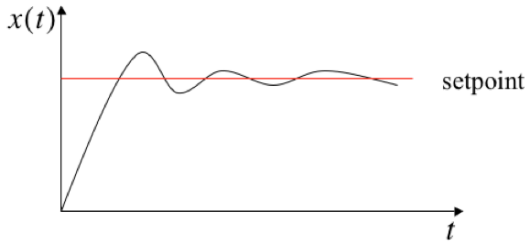
(i) identification of the system, i.e. building of mathematical model of system behaviour and its parameters, and
(ii) analytical description of control objectives.

In real situations, mathematical model of the system can be too difficult or too expensive to obtain. This is especially evident in systems that are normally controlled by human, for example: parking a car, driving a car, operating a crane, maintaining comfortable temperature, etc. These situations are characteristic by

- unavailability of mathematical model of the system, and
- no explicitly formulated performance index, but
- availability of experiential knowledge

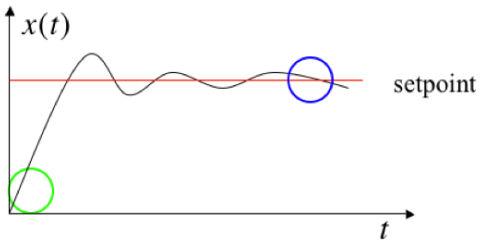
This last fact (availability of experiential knowledge) is very important because it allows linguistic description of the control and controlled variables (in form of fuzzy sets) and linguistic description of the control strategies (in form of fuzzy rules).

Consider the following simple example of control problem



The objective is to stabilize system very quickly. These requirements are conflicting: we can either approach the setpoint in a short time but with many large oscillations, or avoid oscillations in which case the approach will be very slow.

This problem can be eliminated by considering different strategies in different stages of control, as in the following picture:



By considering variables error = setpoint – x(t), chage_of_error = error(t)- *error*(t-1), and control,

and defining fuzzy sets *positive large* (*PL*), *positive small* (*PS*), *zero* (*Z*), *negative small* (*NS*), *negative large* (*NL*) on these variables,

we can formulate the following set of rules:

For phase 1 (green circle)

IF *error* is *PL* and *change_of _error* is *NL* THEN *control is PL*

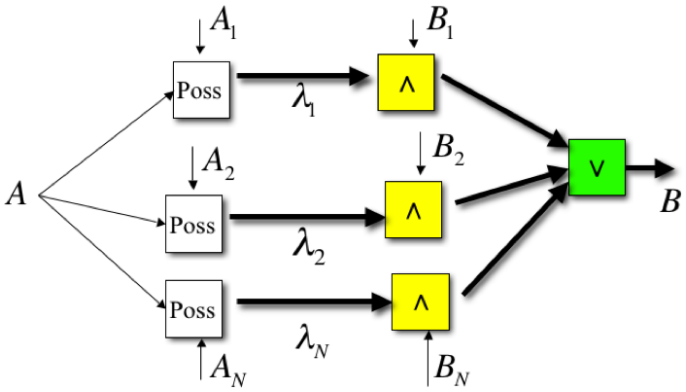
For phase 2 (blue circle)

IF *error* is *Z* and *change_of _error* is *Z* THEN *control is Z*

By constructing more rules/relations for additional phases/segments of the control process, a smooth control of the system can be achieved.

Fuzzy Controller

A general structure of fuzzy controller corresponds to the compositional rule of inference developed earlier.



The first column of blocks (Poss) performs matching of fuzzy set A with antecedents A_k of individual rules R_k. The second column ($\wedge$) corresponds to min operation of the inner term,

$$\min\left[\lambda_k, B_k(y)\right].$$

The last block ($\vee$) finally aggregates the results of individual rules

$$B(y) = \max_k \min[\lambda_k, B_k(y)]$$

Transformations between crisp and fuzzy values

Operation of the fuzzy controller can be summarized as follows: given set of rules IF x is A_k THEN y *is* B_k, and fuzzy set A describing the state of the system, fuzzy controller applies compositional rule of inference to determine control variable in form of fuzzy set B, or formally

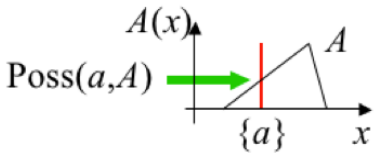
$$\begin{array}{llll} & x \text{ is } A & & \\ \textsf{IF} & x \text{ is } A_k & \textsf{THEN} & y \text{ is } B_k \\ \hline & & & y \text{ is } B \end{array}$$

However, real systems do not operate with fuzzy sets. This implies the need of transforming real (crisp) values to fuzzy sets to be processed by the fuzzy controller, and transforming the resulting fuzzy set into a real value that can be used to actually control the system. These transformations (encoding and decoding) are called fuzzification and dufuzzification, respectively.

Fuzzification

Encoding of crisp input value is already embedded in the existing structure of the fuzzy controller.

Recall that it is possible to apply possibility measure on a fuzzy singleton $\{a\}$ corresponding to the crisp value of input variable:



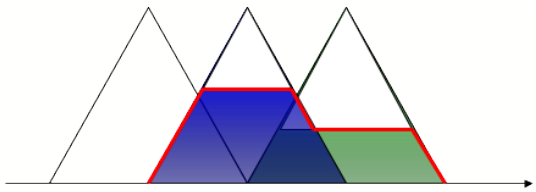
In such case, the result of antecedent matching is simply

$$\lambda_k = Poss(a, A_k) = A_k(a)$$

Defuzzification

Due to the nature of fuzzy rules, its possible that more than one rule applies to a given input value a. As a result, the output fuzzy set can be of complicated shape composed of several consequent fuzzy sets B_k.

This situation is illustrated in the following figure for three consequent fuzzy sets B_1, B_2, and B_3, forming the output set B (indicated by red outline).



The goal of defuzzification is to provide a single value that best represents the output fuzzy set. It is important to consider not only shape/location of the fuzzy sets forming B, but also their height carrying information about relative significance of its corresponding fuzzy rule (i.e. how much is given rule activated – how well is its antecedent matching current input).

One possibility is to determine the center of the resulting area B, and project the location of this point to y axis, i.e.

$$b = \text{COG}\left(B(y)\right) = \frac{\int B(y) y dy}{\int B(y) dy}$$

This approach is called Center of Gravity (COG), center of area, or *centroind.*

Other defuzzification methods include:

- SCOG (Simplified Center of Gravity) $b = \text{SCOG}\left(B(y)\right) = \frac{\sum B(y)y}{\sum B(y)}$
- BOA (Bisector of Area) $b = \text{BOA}\left(B(y)\right) : \int_{\alpha}^{\text{BOA}} B(y)dy = \int_{\text{BOA}}^{\beta} B(y)dy$, where α and β are, respectively, the left and right boundaries of the output fuzzy set
- MOM (Mean of Maximum) $b = \text{MOM}\left(B(y)\right) = \frac{\alpha^* + \beta^*}{2}$ where α^* and β^* are, respectively, the left and right boundaries of the maximum level of the output fuzzy set.
- SOM (Smallest of Maximum) $b = \text{SOM}\left(B(y)\right) = \alpha^*$
- LOM (Largest of Maximum) $b = \text{LOM}\left(B(y)\right) = \beta^*$

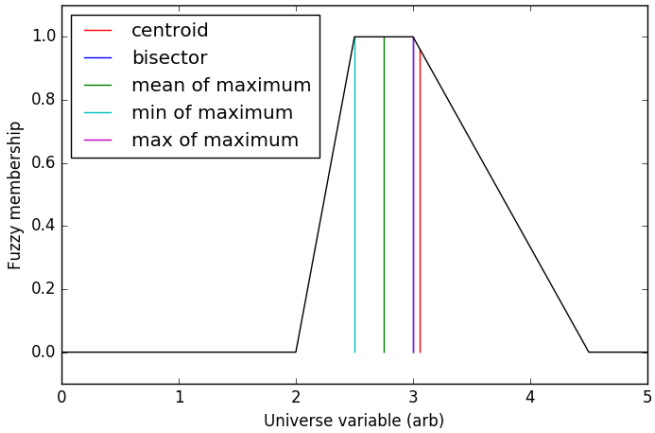
Defuzzification in python/scikit-fuzzy

In python/scikit-fuzzy, there is a function `defuzz`, which implements all basic defuzzification methods:
`b = defuzz(x,B,type)`, where type can be one of the following: `centroid`, `bisector`, `mom`, `som`, `lom`

Sample code

```
y = -10:0.1:10;
B = trapmf(y,[-10 -8 -4 7]);
b = defuzz(y, B, 'centroid');
```

```
defuzz = fuzz.defuzz(x, mfx, 'centroid') {bisector, mom, som, lom}
```



Numerical characteristic of FLC

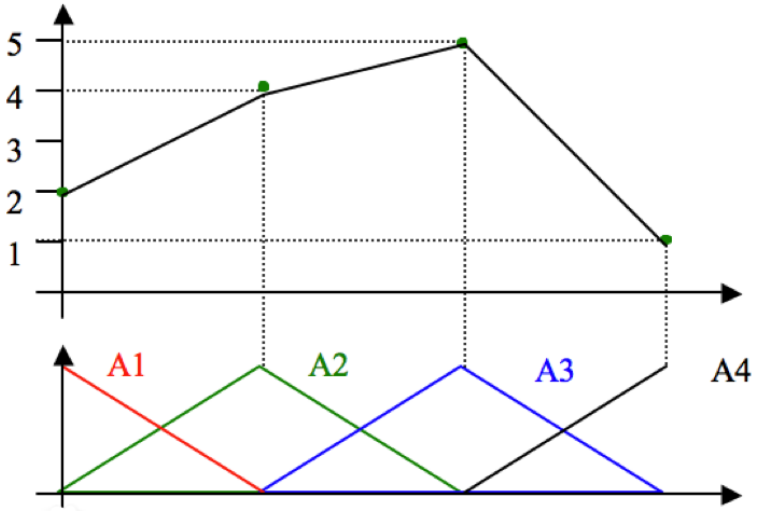
A graph depicting the behavior of a FLC from the point of view of controlled system. It shows functional dependence between input and output variable(s) in their crisp (non-fuzzy) form.

Consider a FLC with input fuzzy sets $A1\{x; 0, 0, 1\}, A2\{x; 0, 1, 2\}, A3\{x; 1, 2, 3\}, A4\{x; 2, 3, 3\}$, output fuzzy singletons $b1 = \{1\}, b2 = \{2\}, b3 = \{4\}, b4 = \{5\}$, and the following set of rules:

IF A_1 THEN b_2
IF A_2 THEN b_3
IF A_3 THEN b_4
IF A_4 THEN b_1

The numerical characteristic of this FLC is constructed by sweep of the values of input variable x over the universe $[0, 3]$, and determining numerical value of the output. In general, this involves determining the activity level of individual rules λ_k for each value of x and defuzzification of the output set B obtained as the union of the output subsets B_k combined with their activities λ_k.

In our case, the situation is simplified due to the fuzzy singletons used in place of output fuzzy subsets. In addition, because the input space is partitioned evenly (i.e. $\sum kA_k(x) = 1$ for all x), we can first concentrate on the modal values of the input fuzzy sets: for x laying at the modal value of input fuzzy set A_k, the activity value is $\lambda_k = 1$ leading directly to output value $y = b_l$, where correspondence between k and l is given by the set of rules. These values are indicated by solid circles in the picture below.



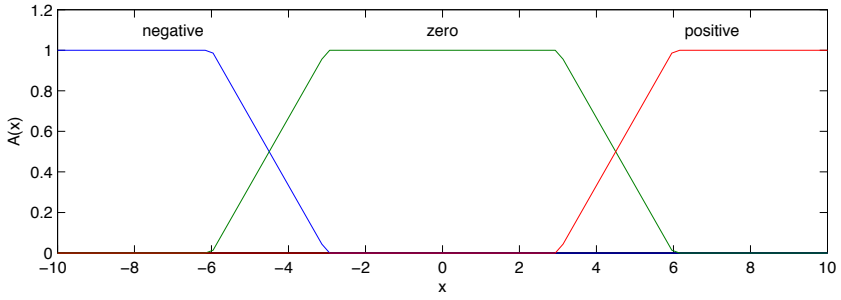
- As two subsequent input fuzzy sets compensate activities of their corresponding rules (i.e. as λ_k decreases λ_{k+1} increases totalling to $\sum kA_k(x) = 1$ for any value of x), these points corresponding to the modal values can be connected by straight lines.
- This example illustrates that using linear concepts (triangular fuzzy sets, even partition) can yield a non-linear behaviour of fuzzy controller (due to nonlinearity of rules).

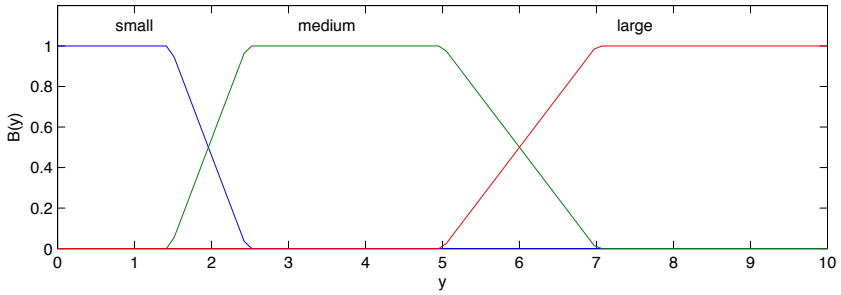
Compiled fuzzy controller

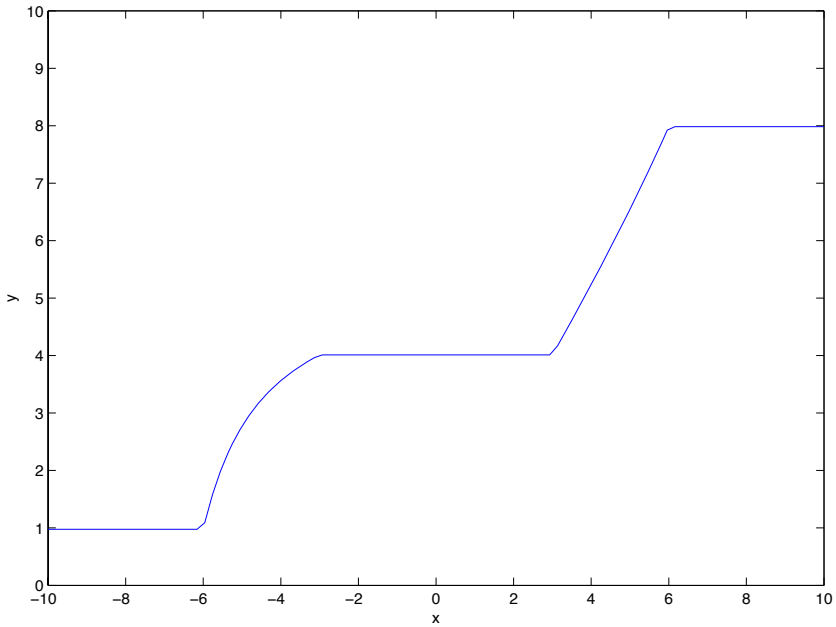
The concept of numerical characteristic can be used to build a compiled version of fuzzy controller stored as a look-up table on particular hardware (ROM, FPGA, etc.).

Example: Single-input Mamdani fuzzy model

IF x is "negative" THEN y is "small"
IF x is "zero" THEN y is "medium"
IF x is "positive" THEN y is "large"

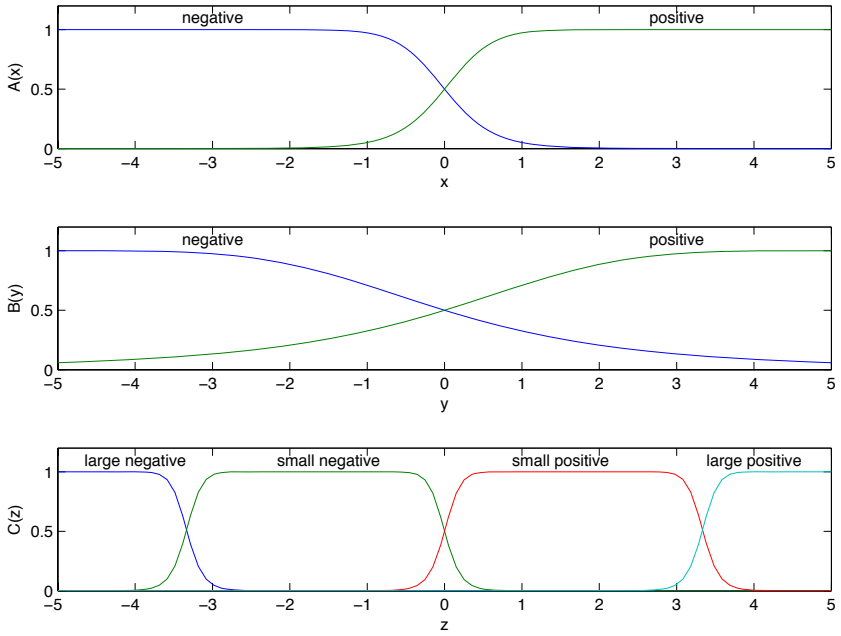


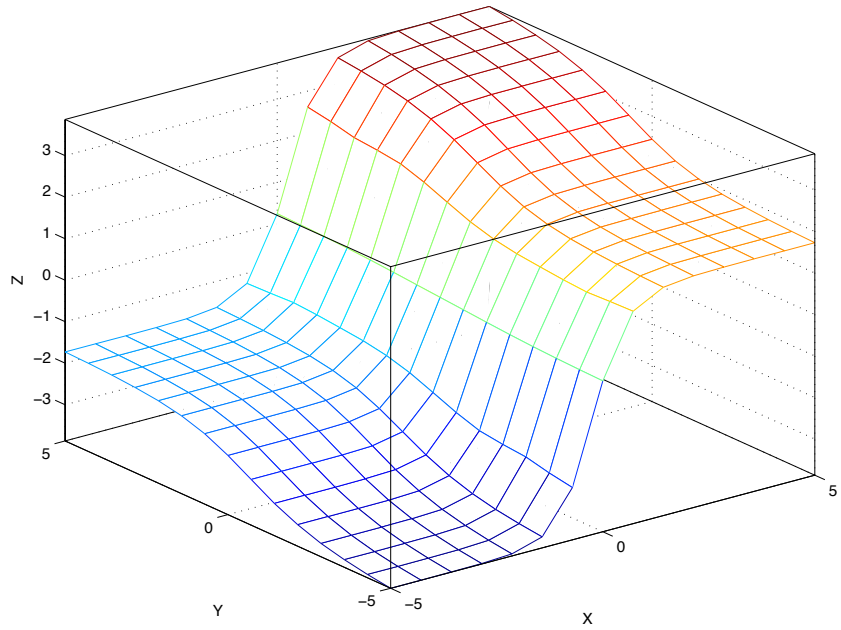




Example: Two-input Mamdani fuzzy model

IF x is "negative"	AND y is "negative"	THEN z is "negative large"
IF x is "negative"	AND y is "positive"	THEN z is "negative small"
IF x is "positive"	AND y is "negative"	THEN z is "positive smal"
IF x is "positive"	AND y is "positive"	THEN z is "positive large"



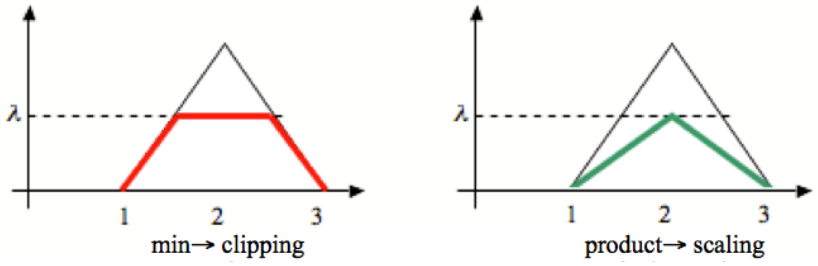


Other types of fuzzy inference

The basic type of fuzzy inference is based on Mamdani fuzzy implication operator (min) and the standard form of rules

$$\mathsf{IF}\ x \text{ is } A\ \mathsf{AND}\ y \text{ is } B\ \mathsf{THEN}\ z \text{ is } C$$

Its variant is based on rules of the same form, but uses Larsen fuzzy implication operator (product). These two types of fuzzy inference provide clipping and scaling of corresponding output fuzzy set



Selection of defuzzification scheme

The choice of defuzzification scheme depends largely on the requirements for speed of processing and implementation constraints: the centroid (COG, COA) method is most precise, but also most demanding. However, using the sum-product composition, calculation of the centroid can be greatly simplified. In this case, it can be shown, that the crisp output obtained through centroid method is equal to the weighted average of the centroids of consequent membership function, i.e.

$$b = \text{COA}\left(B(y)\right) - \frac{\int B(y)ydy}{\int B(y)dy} = \frac{\sum \lambda_i a_i y_i}{\sum \lambda_i a_i}$$

where $a = \int B(y)dy$ and $y_i = \int B(y)ydy / \int B(y)dy$ are the area and centroid of the consequent membership function $B_i(y)$, respectively. These values can be easily determined analytically for most standard membership functions.

Sugeno fuzzy inference

This type of inference modifies the standard form of fuzzy rules to

$$\mathsf{IF}\ x \text{ is } A\ \mathsf{AND}\ y \text{ is } B\ \mathsf{THEN}\ z = f(x, y),$$

i.e. the output of the controller is determined using a crisp function of the input variables. Usually, $f(x, y)$ is polynomial in the input variables x and y, but it can be an arbitrary function that appropriately describes the output of the model within the fuzzy region specified by the corresponding antecedent of the rule.

Sugeno fuzzy inference – order

When $f(x, y)$ is a first-order polynomial, the resulting fuzzy RBS is called a first-order Sugeno fuzzy model. A zero-order Sugeno fuzzy model ($z = k$, a constant) is also often used, and can be viewed as a special case of Mamdani FRBS with consequents specified in form of fuzzy singletons

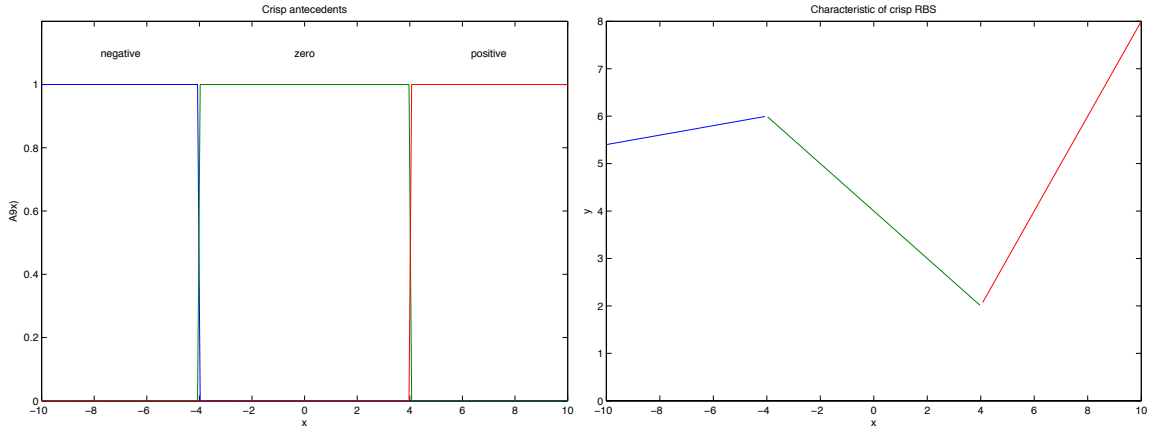
$$\mathsf{IF}\ x \text{ is } A\ \mathsf{AND}\ y \text{ is } B\ \mathsf{THEN}\ z = k,$$

The output of a Sugeno model is smooth as long as the antecedent fuzzy sets have enough overlap (i.e. discontinuity of the consequents does not break the smoothness).

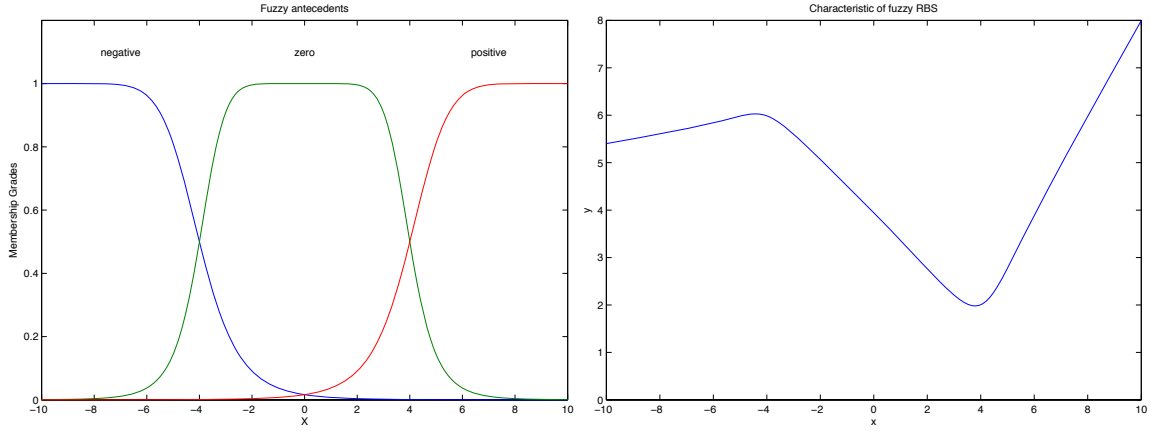
Example: Single-input first-order Sugeno fuzzy model

IF x is "negative" THEN $y = 0.1x + 6.4$
IF x is "zero" THEN $y = -0.5x + 4$
IF x is "positive" THEN $y = x - 2$

Sugeno - crisp antecedents

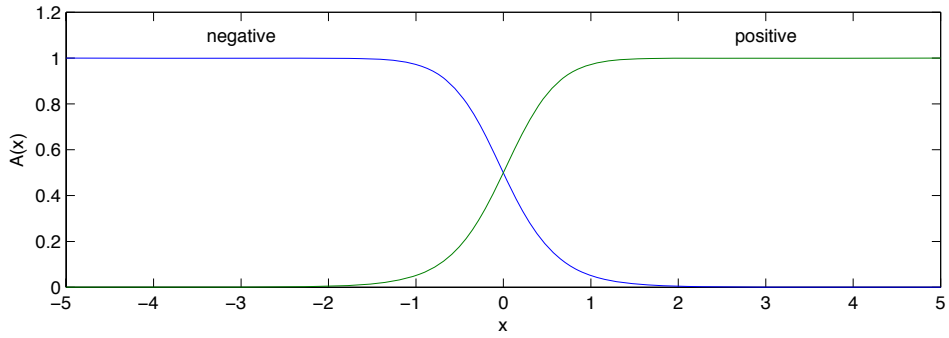


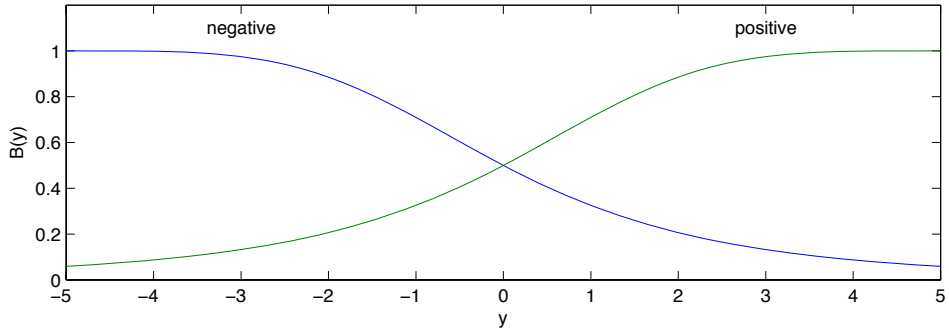
Sugeno - fuzzy antecedents

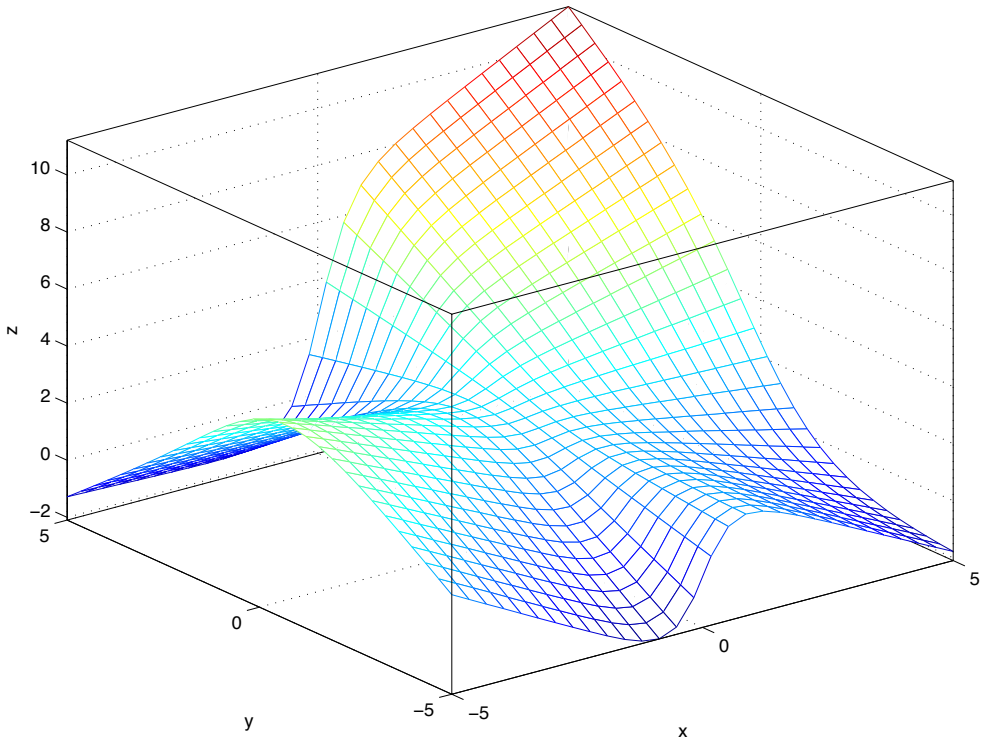


Example: Two-input first-order Sugeno fuzzy model

IF x is "negative" AND y is "negative" THEN $z = -x + y + 1$
IF x is "negative" AND y is "positive" THEN $z = -y + 3$
IF x is "positive" AND y is "negative" THEN $z = -x + 3$
IF x is "positive" AND y is "positive" THEN $z = x + y + 2$


negative
positive
A(x)
x


negative
positive
B(y)
y


z
y
x

Properties of Fuzzy Controllers

Some considerations are important in design and development of fuzzy controllers

Static aspects of FLCs

(i) Completeness
(ii) Continuity
(iii) Consistency

Dynamic aspects of FLCs

(iv) No-interactivity
(v) Robustness and stability

Completeness of fuzzy rule base

The rule base is complete if the fuzzy sets standing in the condition parts of the rules cover the corresponding universes of discourse. Given a set of rules

$$\mathsf{IF}\ x \text{ is } A_i\ \mathsf{THEN}\ y \text{ is } B_i,\ i = 1, 2, ..., n,$$

the condition of completeness can be formulated as

$$\forall x \in X\ \exists i = 1, 2, \ldots, n :\ A_i(x) > 0$$

i.e. at least on rule is active for each x. This condition ensures that a satisfactory control inference is made for any valid set of data (observations). Completeness is, however, a non-strict requirement: less important or unknown conditions can be omitted without seriously affecting the system performance.

Continuity of fuzzy rule base

Requirement of "no gaps" between any rules with respect to their MF. More specifically, when the condition fuzzy set overlap, the action fuzzy sets should also overlap. Formally, a rule base is continuous if for some rule, $A_i \to B_i$, there exist at least one rule

$$A_j \to B_j;\ j \neq i$$

such that both antecedents and consequents of these rules overlap, i.e.

$$\text{Poss}(A_i, A_j) > 0 \text{ and } \text{Poss}(B_i, B_j) > 0$$

Consistency of fuzzy rule base

Consistency as a very important requirement, ensuring that there are no inconsistent (contradictory) rules in the rule base. Consider the following two pairs of rules:

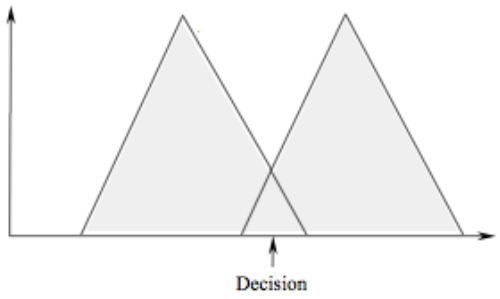
IF road is crowded	THEN drive slowly
IF road is crowded	THEN drive fast
IF close to obstacle	THEN turn left
IF close to obstacle	THEN turn right

They both represent conflicting rules. This inconsistency can be caused by collecting rules from different experts (the first pair of rules), or by basing rules on different observations (the second pair). Taking the rules blindly, without checking their consistency, the system would not operate effectively and could cause serious problems. Conceptually, one can envision the following cases of inconsistency.

Inconsistency - case 1

$$A_i \rightarrow B_i \text{ OR } A_i \rightarrow B_i^{'}$$

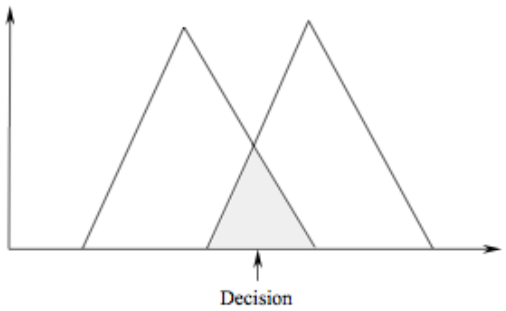
In this case, there are two rules with the same antecedent but different consequents, connected with OR. If the consequents are mutually exclusive, i.e. $B_i \bigcap B_i^{'} = \emptyset$, the rule base is said to be inconsistent. In the case of partial overlap, the level of inconsistency decreases, but the inference provided by the system is still inconsistent as it lies in the middle of a bimodal (or multimodal) membership function.



Inconsistency - case 2

$$A_i \rightarrow B_i \text{ AND } A_i \rightarrow B_i^{'}$$

In this case, there are two rules with the same antecedent but different consequents, this time connected with AND. If the consequents are mutually exclusive, i.e. $B_i \bigcap B_i^{'} = \emptyset$, the rule base is said to be inconsistent. In the case of partial overlap, the level of inconsistency decreases, but the inference provided by the system is still inconsistent – although this time the resulting MF is unimodal, its peak lies in the middle of two consequent membership functions.



Inconsistency evaluation

Quantitatively, inconsistency can be evaluated using possibility measure, which can be used as an expression of similarity between two fuzzy sets. We need to capture situation of similar conditions and different conclusions, i.e. in extreme case

$$\text{Poss}(A_i, A_j) = 1, \ \text{Poss}(B_i, B_j) = 0$$

Consistency between two rules, i and j, can be evaluated with an implication operator:

$$\text{Cons}(i, j) = \text{Poss}(A_i, A_j) \rightarrow \text{Poss}(B_i, B_j)$$

In the case above:

$$1 \rightarrow 0 = 0$$

Implication operator

An example of implication operator suitable for this purpose has the following form

$$a \to b = \begin{cases} 1 & \text{if} a \leq b, \\ 0 & \text{if} a > b. \end{cases}$$

Mutual consistency of all pairs of rules in a rule base can be arranged in the following consistency matrix

$$\begin{bmatrix} \text{Cons}(1,1) & \text{Cons}(1,2) & \text{Cons}(1,3) & \text{Cons}(1,n) \\ \text{Cons}(2,1) & \text{Cons}(2,2) & \cdots & \\ & & & \\ \text{Cons}(n,1) & \text{Cons}(n,2) & & \text{Cons}(n,n) \end{bmatrix}$$

Implication operator

To evaluate the consistency of one particular rule, say i, with respect to all other rules in the base, average of mutual consistencies is taken across the i-th row of the matrix

$$\text{Cons}(i, \text{RB}) = \frac{1}{n} \sum_{j=1}^{n} \text{Cons}(i,j)$$

After determining consistency of all rules this way, the rules can be sorted according to their consistency value $\text{Cons}(i, RB)$, and most inconsistent rules (from the bottom of the sorted group) manipulated to improve the overall consistency of the rule base.

Enhancing consistency

1. Eliminating inconsistent rules – simple but can cause problems with coverage/continuity of the rule base
2. Modifying inconsistent rules – there are several approaches:
   - Making conditions more specific and/or conclusions more general
   - Adding a (missing) condition

Making conditions more specific and/or conclusions more general

$$\text{Cons}(i,j) = \text{Poss}(A_i, A_j) \to \text{Poss}(B_i, B_j)$$

This can be achieved, e.g. using linguistic modifiers (hedges), such as "very", "more or less". There are limits to this, as the modal values do not change their location, MFs are only compressed or expanded.

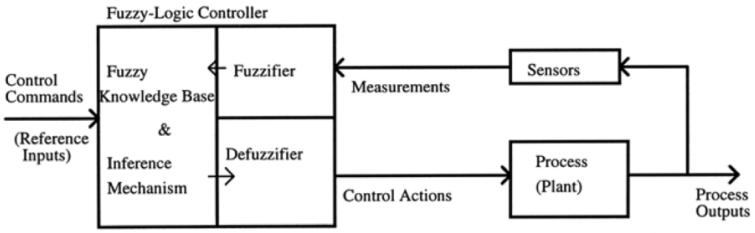
Adding a (missing) condition C

IF road is crowded	AND road is slippery	THEN drive slowly
IF road is crowded	AND road is dry	THEN drive fast

This changes the situation by removing the conflict. This way, we are now evaluating $\text{Poss}(A_i \wedge C_i, A_j \wedge C_j)$ instead of $\text{Poss}(A_i, A_j)$ which leads to lowering the amount of condition overlap, effectively removing the inconsistency.

Fuzzy Control Architectures

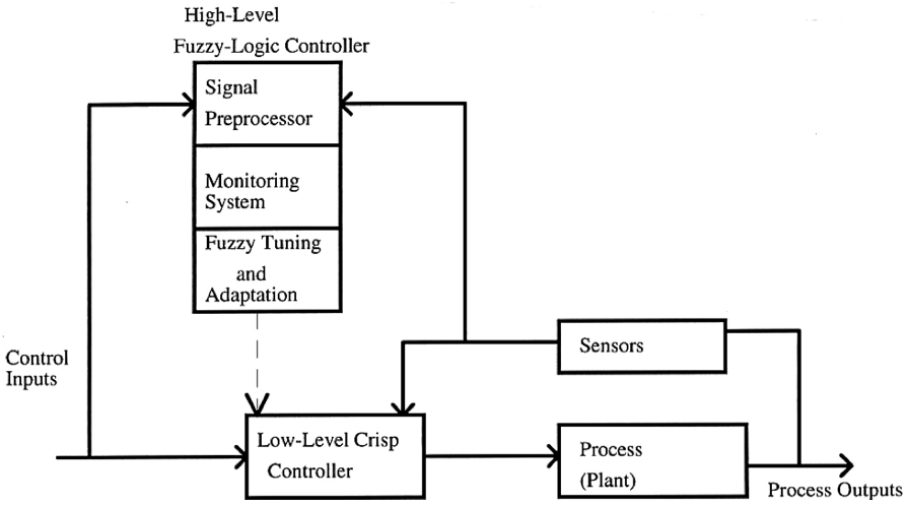
1) The most common, conventional, direct-control architecture. FLC generates low-level direct control signals. This architecture is suitable for low demand control problems (high time constants of controlled process, low control bandwidth) which would be difficult to model analytically and for which there is a sufficient experience with low-level human-in-the-loop control.



Fuzzy Control Architectures (cont.)

2) Hierarchical supervisory control architecture using crisp low-level direct controller and high-level fuzzy controller performing supervisory tasks, i.e.

- process monitoring
- performance assessment
- tuning, adaptation, and restructuring of the low-level controller



Chapter 3

Neural Networks

Neural Networks

- Information processing systems inspired both by
  - biological nervous systems, and
  - mathematical theories of learning.
- Massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations which are intended to interact with objects of the real world in the same way as biological nervous systems do

[Kohonen, 1988]

3.1 Introduction

3.1.1 Biological Inspiration

The Brain

- The brain is a highly complex, non-linear, and parallel computer, composed of some 10^{11} neurons that are densely connected ($\sim 10^4$ connections per neuron)
- A neuron is much slower (10^{-3}sec) compared to a silicon logic gate (10^{-9}sec), however the massive interconnection between neurons make up for the comparably slow rate.

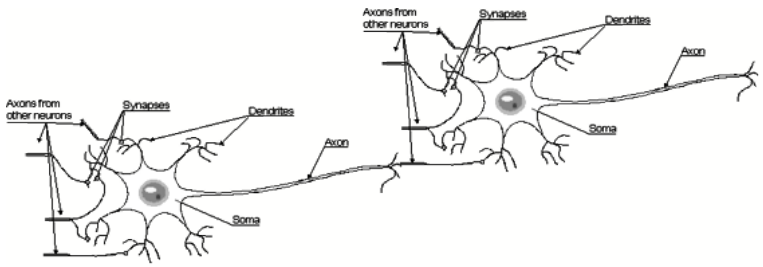
Information Processing in Brain

Hundred Steps rule: Complex perceptual decisions are arrived at quickly (within a few hundred ms)
Individual neurons operate slowly (10^{-3}sec), these calculations do not involve more than about 100 serial steps and the information sent from one neuron to another is very small (a few bits)

Neural Plasticity: Some of the neural structure of the brain is present at birth, while other parts are developed through learning, especially in early stages of life, to adapt to the environment (new inputs).

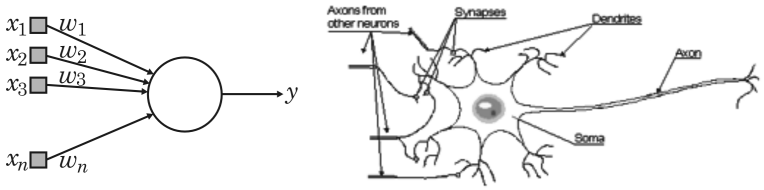
3.1.2 Neurons

The Biological Neuron

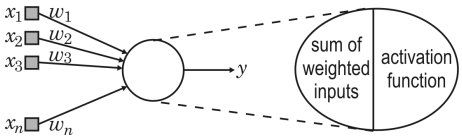


The Artificial Neuron

Model with components analogous to those of biological neurons



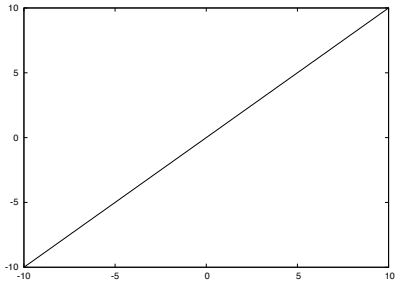
The Artificial Neuron



Sum of weighted inputs $tot = \sum_{i=1}^{n} w_i x_i$ Activation Function $o = f(tot)$

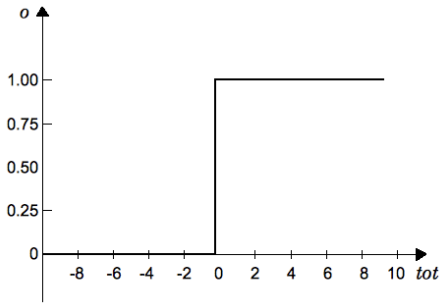
Note: there are other alternatives too.

Activation Function Linear

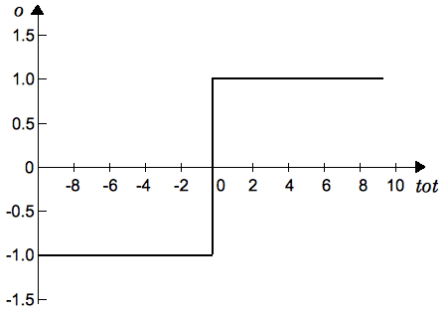


$$o = f_{lin}(tot) = tot$$

Activation Function Hard Limiting

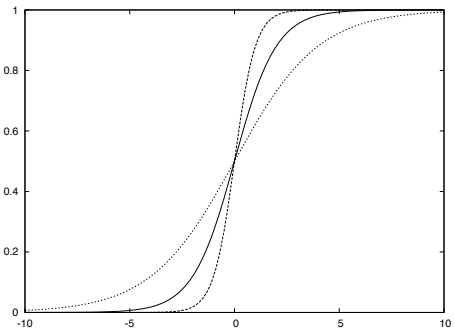


$$o = f_{hlu}(tot) = \begin{cases} 0 & \text{if } tot \leq 0, \\ 1 & \text{if } tot > 0. \end{cases}$$

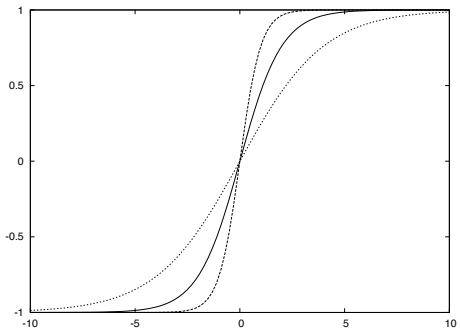


$$o = f_{hlb}(tot) = \begin{cases} -1 & \text{if } tot \leq 0, \\ 1 & \text{if } tot > 0. \end{cases}$$

Activation Function Sigmoid



$o = f_{sigu}(tot) = \frac{1}{1+e^{-\alpha tot}}$



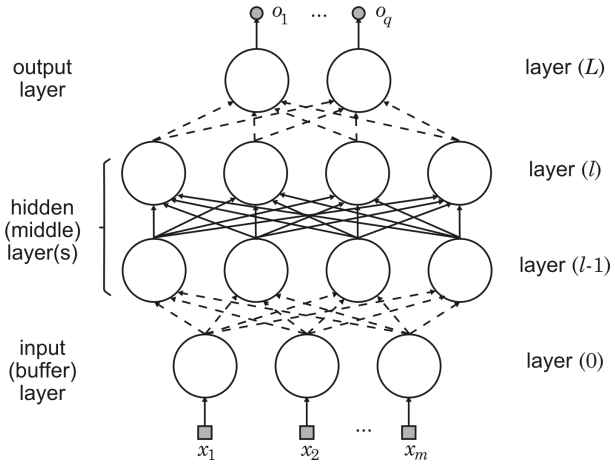
$o = f_{sigb}(tot) = \frac{1-e^{-\alpha tot}}{1+e^{-alphatot}}$

3.1.3 Network Topology

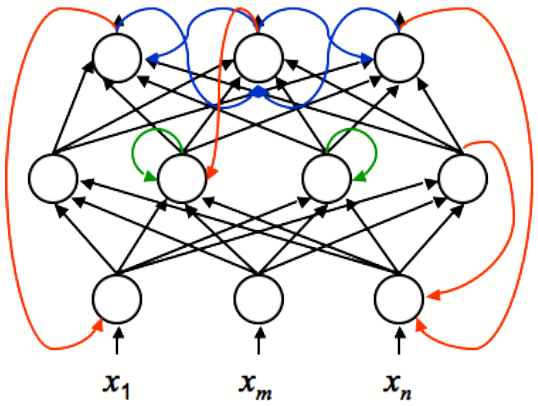
Artificial Neural Networks (ANN)

- An information processing system consisting of a large number of simple, highly interconnected processing elements (neurons) in an architecture inspired by the structure of the cerebral cortex of the brain.
- Usually organized into a sequence of layers with full connections between the layers.

Example architecture of ANN



Interconnection Variations



feedforward local feedback global feedback lateral

Impact of recurrent connections

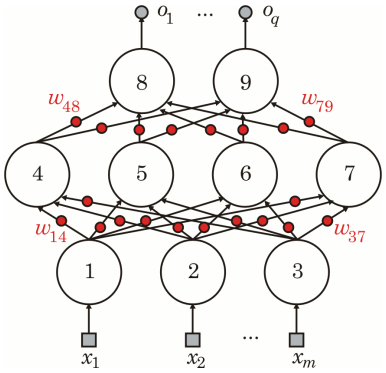
Fedforward	Recurrent
direct calculation	state-machine like
fixed calculation time	require "setting time"
direct I/O mapping	have memory
open loop	closed loop

Note
Feedforward NN with linear activation function can be described using *linear algebra*.

3.1.4 Weights

Weights

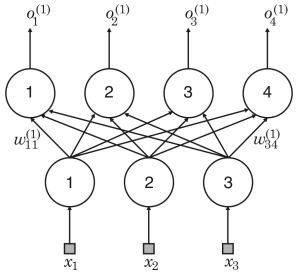
Each connection between neurons has an adjustable weight.



Notation

This notation is not very practical, let's develop a better one.

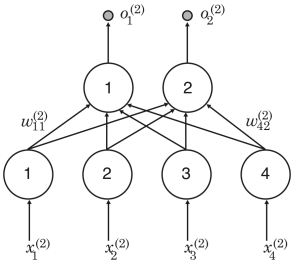
$$\mathbf{o}^{(1)} = [o_1^{(1)}, o_2^{(1)}, o_3^{(1)}, o_4^{(1)}]$$



$$\mathbf{x}^{(0)} = \mathbf{x} = [x_1, x_2, x_3]$$

$$\begin{aligned}\mathbf{o}^{(1)} &= \mathbf{W}^{(1)} \cdot \mathbf{x}^{\mathrm{T}} = \\ &= \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{23}^{(1)} \\ w_{14}^{(1)} & w_{24}^{(1)} & w_{34}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\end{aligned}$$

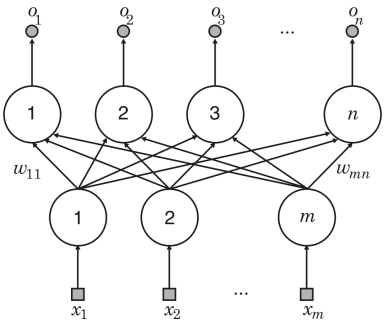
$$\mathbf{o} = \mathbf{o}^{(2)} = [o_1^{(2)}, o_2^{(2)}]$$



$$\begin{aligned}\mathbf{x}^{(2)} &= [x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, x_4^{(2)}] = \\ &= [o_1^{(1)}, o_2^{(1)}, o_3^{(1)}, o_4^{(1)}]\end{aligned}$$

$$\begin{aligned}\mathbf{o}^{(2)} &= \mathbf{W}^{(2)} \cdot \mathbf{x}^{(2)\mathrm{T}} = \\ &= \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} & w_{41}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} & w_{42}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ x_4^{(2)} \end{bmatrix}\end{aligned}$$

Example: Linear Associator



- Very simple ANN
- Capable to store more than one (linear) relationships simultaneously
- Not very accurate

3.2 Historical Overview

History: 1940s and 1950s

- 1940 Similarity of the physiology of the brain and information processing
- 1943 First formal model of an elementary computing neuron, based on threshold logic (McCulloch and Pitts)
- 1949 Learning hypothesis: information is stored in connections; learning scheme for updating connections (Hebb)
- 1954 First computational machines used to simulate a Hebbian network (Farley and Clark)
- 1958 Perceptron - a trainable machine for pattern recognition (Rosenblatt)
- 1959 Biological model of simple and complex cells in the primary visual cortex (Nobel laureates Hubel and Wiesel)
- 1960 ADALINE - device for adaptive control and pattern recognition, trained by the least-mean-square (LMS) learning rule (Widrow)

History: 1960s and 1980s

- 1965 First functional networks with many layers (Ivakhnenko and Lapa, Group Method of Data Handling)
- 1965+ Weak learning theory and modest computational resources led to stagnation
- 1969 Minsky and Papert pointed out limitations of neural networks and questioned their usefulness. Only a handful of researchers (including those in Europe and Japan) continued NN research afterwards.
- 1975 Cognitron and neocognitron for pattern recognition (Fukushima)
- 1977 Mathematical theory of NNs (Amari)
- 1982 Unsupervised learning neural networks for feature mapping (Kohonen)
- 1982 Neural theories inspired by developmental physiology (Grossberg)
- 1985 Recurrent neural networks for information storage and retrieval and optimization (Hopfield)

History: 1980s and 1990s - turning point

- 1975 Backpropagation algorithm for training multi-layer feedforward percetrons was first developed, but received little attention (Werbos)
- 1986 The powerful gradient descend based backpropagation training algorithm received wide attention - parallel distributed processing, connectionism (Rumelhart and McClelland)
- 1991 The vanishing gradient (VG) problem affects many-layered feedforward networks that used backpropagation and also recurrent neural networks (RNNs)
- 1992 Multi-level hierarchy of pre-trained and then fine-tuned networks to overcome the VG proble (Schmidhuber); other aleternatives relied only on the gradient sign (Rprop, 2003)

History: 2000s

- Learning high-level representations using successive layers of binary or real-valued latent variables (Hinton)
- Earlier challenges in training deep neural networks were successfully addressed with methods such as unsupervised pre-training, while available computing power increased through the use of GPUs and distributed computing.
- Neural networks were deployed on a large scale, particularly in image and visual recognition problems. This became known as "deep learning".

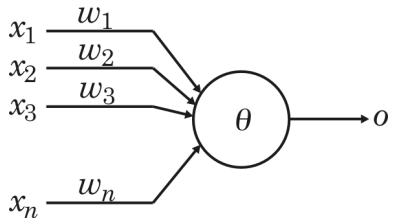
History: 2010s

- Networks learn to recognize higher-level concepts (such as cats) only from watching unlabeled images taken from YouTube (Ng and Dean).
- The state of the art in deep learning feedforward networks alternated between convolutional layers and max-pooling layers, topped by several fully or sparsely connected layers followed by a final classification layer.
- Supervised deep learning methods achieve human-competitive performance on certain tasks.

3.3 Early Models

3.3.1 McCulloch-Pitts

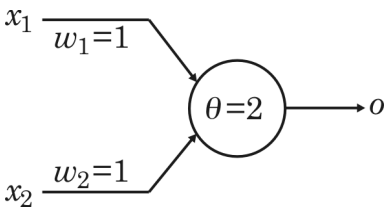
McCulloch-Pitts Neuron Model



$$w_i = \pm 1;\ x_i = \{0, 1\}$$

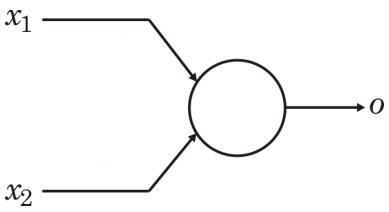
$$o = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i < \theta. \end{cases}$$

Boolean Logic Implementation: AND Logic Function



x_1	x_2	$\rightarrow$	o
1	1		1
1	0		0
0	1		0
0	0		0

Boolean Logic Implementation: OR Logic Function

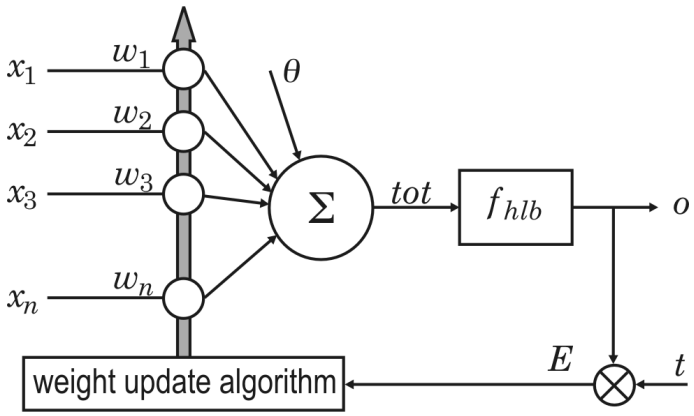


x_1	x_2	$\rightarrow$	o
1	1		1
1	0		1
0	1		1
0	0		0

3.3.2 Perceptron

- 1958 (Rosenblatt); training algorithm that provided the first procedure for training a simple ANN
- The simplest form of a neural network - single neuron with adjustable synaptic weights and a hard limiter activation function (based on the McCulloch and Pitts neuron model).

- Operation: The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to $+1$ if its input is positive and -1 if it is negative (sometimes values $[0, 1]$ are used).
- Learning: making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron.
- To allow realization of mappings that require position of hard limit different from 0, a threshold term θ is added.
- Initial weights and threshold are randomly assigned, usually in the range $[-0.5, 0.5]$, and then updated to obtain the output consistent with the training examples.



Perceptron: training algorithm

1. Initialization. Set initial weights w_i and threshold θ to small random numbers (e.g. in the range $[-0.5, 0.5]$)
2. Activation. From a training set T, select an input pattern $\mathbf{x}(k) \in T$. Apply it to activate the perceptron and calculate the actual output $o(k)$
$$o(k) = f_{hlb}\left(\sum_{i=1}^{m} w_i x_i(k) - \theta\right),$$
where m is the number of the perceptron inputs (equal to the dimension of input patterns), and f_{hlb} is the bipolar step activation function.
3. Determine error using the desired output (target) $t(k) \in T$:
$$E(k) = t(k) - o(k)$$
4. Weight update. Update the weights according to the following rule $w_i^{\text{new}} = w_i^{\text{old}} + \Delta w_i$
$$\begin{array}{ll} \text{If } E(k) > 0: & \text{increase perceptron output } o(k), \\ \text{If } E(k) < 0: & \text{decrease } o(k), \end{array}$$
i.e. $\Delta w_i(k) = \eta \cdot x_i(k) \cdot E(k)$
5. Iteration.Steps 2-4 compose an *iteration*. Perform one iteration for each training pattern $k = 1, \ldots, n$.
6. Repetition. One set of updates of all the weights for all the training patterns is called one *epoch* of training. Repeat the process until convergence.

Perceptron for Classification

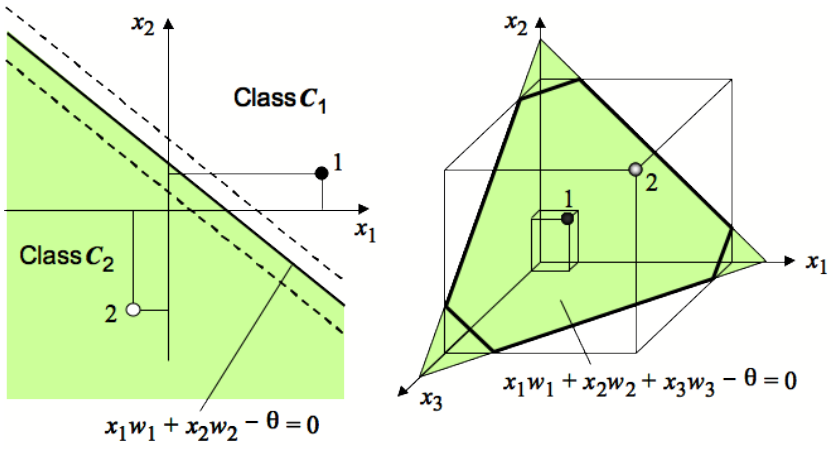
- Can be used for binary classification.
- Given training examples of classes C_1, C_2 we train the perceptron in such a way that it classifies correctly the training examples:
  - If the output is +1 then the input is assigned to C_1
  - If the output is -1 then the input is assigned to C_2

Perceptron for Classification – Learning

Numerically: finding suitable values for the weights in such a way that the training examples are correctly classified.
Geometrically: finding a hyper-plane that separates the examples of the two classes.

$$\sum_{i=1}^{m} w_i x_i - \theta = 0$$

Perceptron - Geometric View

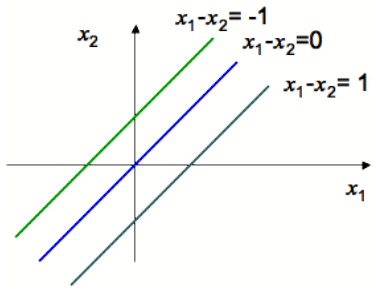


Bias of a Neuron

The bias θ has the effect of applying an affine transformation to the weighted sum

$$tot = \sum_{i=1}^{m} w_i x_i - \theta,$$

tot is sometimes called *net input* to the neuron

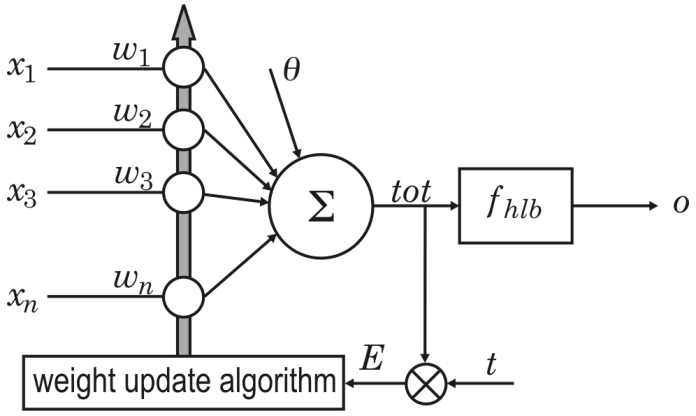


Note

It is practical to augment m–dimensional input vector by $x_0 = 1$ and designate $w_0 = \theta$ to represent the threshold term.

3.3.3 ADALINE

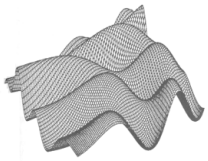
Widrow and Hoff, 1960s



Note the difference compared to perceptron learning.

ADALINE Learning: LMS and Gradient Descent

- LMS (Least Mean Squares): form of gradient descent through weight space
- Waves in the ocean: How to find the bottom of a wave using only local info?



Gradient Descent
Determine error: $E = \sum_{k=1}^{n} (t_k - tot_k)^2$

Move the weights in the negative direction of the gradient

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

Move the weights

How to do this ?

For an individual pattern (particular k)

$$E = (t - tot)^2 = \left(t - \sum_{i=0}^{m} x_i w_i \right)^2$$

differentiating

$$\frac{\partial E}{\partial w_j} = 2\,(t - tot)\,\frac{\partial}{\partial w_j}(-tot) = -2\,(t - tot)\,x_j$$

Now change the weights in the opposite direction of $\frac{\partial E}{\partial w_j}$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = -\eta\,(t - tot)\,x_j$$

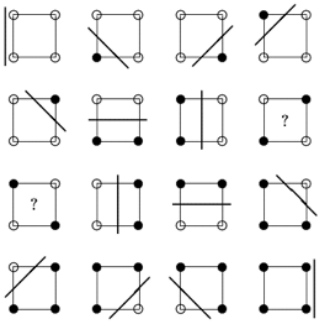
... and finally

$$w_j^{\text{new}} = w_j^{\text{old}} - \Delta w_j = w_j^{\text{old}} + \eta\,(t - tot)\,x_j$$

3.3.4 Pattern separability and ADALINE

Linear separability of patterns
Separation of Input Space

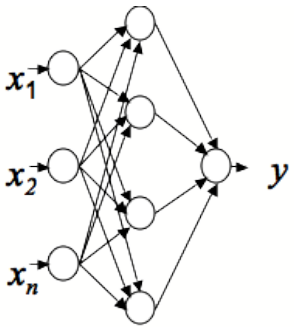
- Possible input values $x_i = \pm 1 : d = 2$
- Dimension of inputs $m = 2$
- Number of possible m-element vectors $k = dm = 4$
- Number of possible partitions into two classes ($y = \pm 1$): $2k = 16$



MADALINE
is a 3-layer network with Many ADALINEs

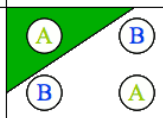
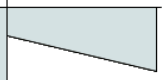
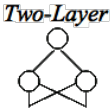
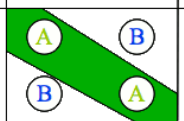
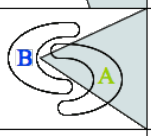
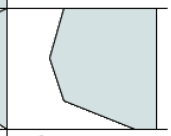
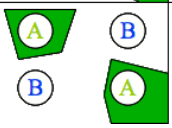
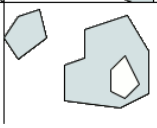
- ADALINE - only linear decision boundary
- MADALINE - with multiple ADALINEs, a decision boundary can be “carved” into more complex shapes

Unfortunately
MADALINEs are hard to teach . . .



ADALINE – MADALINE

Non linearly separable problems

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Abitrary (Complexity Limited by No. of Nodes)			

3.4 Multilayer Perceptron

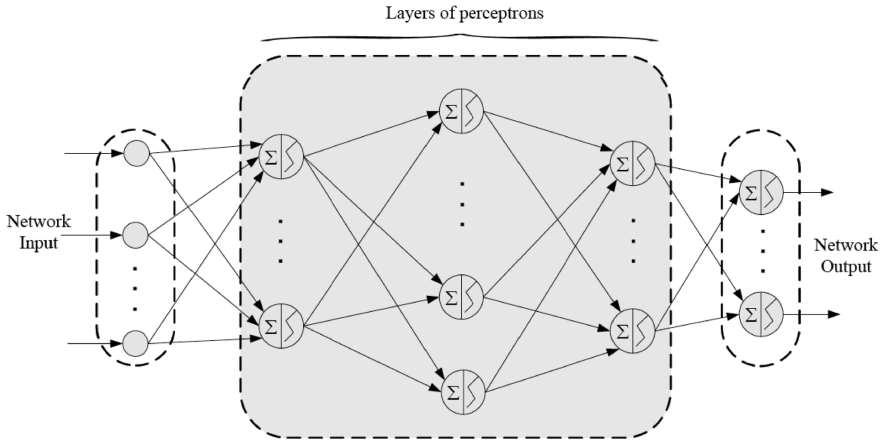
The Multilayer Perceptron (MLP)

- Training of synaptic weights is difficult when dealing with more than 1 neuron, let alone multiple layers of neurons
- MLP uses a training method called backpropagation (BP) learning which uses (again) the idea of gradient descent to determine weight modifications to minimize error.
- This is the same concept as used in ADALINE, but
  a) the actual system output is used (rather than the product of linear combination with a neuron); and
  b) the concept is extended backward into the system to allow for adjustment of cascaded layer weights

3.4.1 Topology

MLP Topology

- The networks of this type are feedforward and contain one or more hidden layers
- The input layer is often referred to as a *buffer layer* and the weights of its inputs are held at unity



3.4.2 Backpropagation algorithm

Cumulative error

$$E_c = \sum_{k=1}^{n} E(k) = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

Cummulative error

Cycling through the iterations (training data)

Cycling through the output neurons

Note:
The outer sum (over k, training data) is used only for *batch training*. When performing *on–line training*, it is not included (next slide).

3.4.3 On–line vs. batch learning

Training: batch vs. on–line

Training can be stated as an optimization problem: finding set of weights that minimizes error E.

- In *on–line* training, weight modifications are made pattern by pattern.

$$\min_w E(k) = \frac{1}{2}\sum_{i=1}^{q}\left[t_i(k) - o_i(k)\right]^2$$

- In *batch* (or off–line) training, all training patterns are presented to the system before weights are updated.

$$\min_w E_c = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q}\left[t_i(k) - o_i(k)\right]^2$$

On–line training

Similarly as before (in the case of ADALINE):

$$\Delta \mathbf{w}^{(l)} = -\eta \nabla_{\mathbf{w}^{(l)}} E = -\eta \frac{\partial E(k)}{\partial \mathbf{w}^{(l)}}$$

Using chain rule:

$$\Delta \mathbf{w}^{(l)} = \Delta w_{ij}^{(l)} = -\eta \left[\frac{\partial E(k)}{\partial o_i^{(l)}}\right]\left[\frac{\partial o_i}{\partial tot_i^{(l)}}\right]\left[\frac{\partial tot_i}{\partial w_{ij}^{(l)}}\right]$$

$\frac{\partial E(k)}{\partial o_i^{(l)}}$ – how the error varies with a change in the output

$\frac{\partial o_i}{\partial tot_i^{(l)}}$ – how the output varies with a change to the neuron inputs

$\frac{\partial tot_i}{\partial w_{ij}^{(l)}}$ – how the total input vary with changes to the specific weight

For the output layer, $l = L$, we can write:

$$\begin{aligned}
\left[\frac{\partial E(k)}{\partial o_i^{(L)}}\right] &= \frac{\partial}{\partial o_i^{(L)}}\frac{1}{2}(t_i - o_i^{(L)})^2 = -1 \cdot (t_i - o_i^{(L)}) \\
\left[\frac{\partial o_i^{(L)}}{\partial tot_i^{(L)}}\right] &= \frac{\partial}{\partial tot_i^{(L)}}\big(\, f(tot_i^{(L)})\big) \\
\left[\frac{\partial tot_i^{(L)}}{\partial w_{ij}^{(L)}}\right] &= x_{ij}^{(L)} = o_j^{(L-1)}
\end{aligned}$$

For sigmoidal function (nicely differentiable - necessary for BP)

$$\begin{aligned}
f^{'}(tot_i^{(L)}) = \frac{\partial}{\partial tot_i^{(L)}}\left(\frac{1}{1+e^{-tot_i}}\right) &= -\frac{1}{(1+e^{-tot_i^{(L)}})^2}\cdot(-e^{-tot_i^{(L)}}) \\
&= \left(\frac{1}{1+e^{-tot_i}}\right)\cdot\left(\frac{1+e^{-tot_i^{(L)}}-1}{1+e^{-tot_i^{(L)}}}\right) \\
&= f(tot_i^{(L)})\cdot(1 - f(tot_i^{(L)}))
\end{aligned}$$

Put together: $\Delta w_{ij}^{(L)} = \eta(t_i - o_i^{(L)})[\, f^{'}(tot_i^{(L)})]o_j^{(L-1)}$

So far so good, but:
This is only useful for the output layer neurons!

Error propagation

The approach now is to isolate the terms that allow us to propagate the error back into the layers before the output.

$$\begin{aligned} \text{For output layer}: \quad & \Delta w_{ij}^{(L)} = \eta \delta_i^{(L)} o_j^{(L-1)} \\ \text{where}: \quad & \delta_i^{(L)} = (t_i - o_i^{(L)}) f^{'}(tot_i^{(L)}) \end{aligned}$$

$$\begin{aligned} \text{For hidden layer(s)}: \quad & \Delta w_{ij}^{(l)} = \eta \delta_i^{(l)} o_j^{(l-1)} \\ \text{where}: \quad & \delta_i^{(l)} = f^{'}(tot_i^{(l)}) \sum_{p=1}^{n_i} \delta_p^{(l+1)} w_{p_i}^{(l+1)} \end{aligned}$$

δ is called *error signal*. Because error signals of layer $(l+1)$ are used to determine the error signal of neurons in layer (l), this algorithm is called (error) *backpropagation*.

BP algorithm summary

1. Initialize weights to small random values;
2. Select a piece of training data;
3. Propagate the input through the network;
4. Calculate the total cumulative error to this iteration

   $$E_c = E_c + E(k)$$

   and the error signal for the output layer neurons [sigmoid]

   $$\delta_i = (t_i - o_i) o_i (1 - o_i)$$

5. Update the output layer weights $\Delta w_{ij} = \eta \delta_i o_j$ and proceed backward using [sigmoid]:

   $$\delta_i = o_i^l (1 - o_i^l) \sum_{p=1}^{n_i} \delta_p^{(l+1)} w_{p_i}^{(l+1)}$$

6. Repeat with next training data
7. If the cumulative error, E_c is within tolerances, terminate. Otherwise, continue with another epoch (Steps 2–6)

Batch learning

- For batch learning, weights are only updated at the end of an epoch.
- However, individual $\Delta w_{ij}^{(l)}(k)$ are calculated for each pattern k using the same procedure as in online learning.
- At the end of the epoch (all patterns presented) , weights are updated by

  $$\Delta w_{ij}^{(l)} = \sum_{k=1}^{n} \Delta w_{ij}^{(l)}(k)$$

  (the sum of incremental weight updates for all patterns)

3.4.4 Practical Considerations

1. Training Data
2. Initial Weights
3. Learning Rate
4. On-line vs. Batch Training
5. Activation Function
6. Flat Spots and Local Minima
7. When to Stop the Training
8. Number of Hidden Units
9. Learning Rates Variation
10. Learning with Momentum

Preparation of Training Data

- In principle, *raw data* can be used to train networks
- In practice, data *preparation* methods are often used to help appropriate learning
  - Data selection - training data should be representative
  - Rescaling and normalizing data
  - Shuffling data if on–line training is used

More on data preparation later (next section)

Weight Initialization

- Gradient descent treats all weights the same way – if they were all set to 0, no learning would occur
- Generally, weights are initialized at the beginning of learning process to small random values (usually using uniform distribution from interval $[-r, +r]$)
- Value of r should be as large as possible, but such that sigmoid functions are not saturated (i.e. 0.5)
- To make sure that network performance is independent of the initialization, several sets of initial weights should be generated and the best one selected based on observing network performance during learning

Learning Rate η

- If η is too small, learning takes too long (too many incremental weight changes are needed to get close to the minimum of error function)
- If η is too large, the wight updates can over–shoot the minimum and oscillate (or even diverge)

The optimal value of η is problem and network dependent; a range of different values can be used (e.g. $\eta \in \{.001, 0.01, 0.1\}$) and the most suitable value chosen by observing the process.

The value can be changed during the learning process so that it is large at the beginning and keeps getting smaller as learning proceeds (i.e. as the weights are getting close to the minimum)

$$\eta(t) = \frac{\eta(0)}{t} \quad \text{or} \quad \eta(t) = \frac{\eta(0)}{1 + t/\tau}$$

Batch vs. on-line Training

- *Batch training* approximates true gradient descent (weight updates are performed only after all patterns have been presented)
- *On–line training* (or sequential training) performs weight updates immediately after processing each training pattern.

Because there are $n = \texttt{number of patterns}$ updates per epoch in on–line training (as opposed to 1 update per epoch in batch training), it is usually much quicker.
At the same time, the weight changes are rather erratic - this may be advantageous to escape from the vicinity of a local minimum.

Activation (transfer) Function

- Differentiability
- Computational efficiency
  - Sigmoid (logistic) function
    $$f^{'}(x) = f(x)\,(1 - f(x))$$
  - Hyperbolic tangent
    $$f^{'}(x) = 1 - f(x)^2$$
  - Linear function $f^{'}(x) = 1$
- Hardware implementation (avoid complicated functions)
  - Eliott function
    $$f(x) = \frac{x}{1 + |x|}$$

Flat spots in the error function

- Flat spots are caused by saturated portions of sigmoids
- It takes long time to pass through them, and get anywhere close to a minimum
- Sigmoidal activation function has derivative $\rightarrow 0$ as it saturates i.e. if the outputs are totally wrong (0 instead of 1, or vice versa), weight updates are very small
- Solutions:
  - Target off–sets: 0.1 and 0.9 instead of 0 and 1 (no saturation; learning does not get stuck)
  - Sigmoid prime off–set – add a small value (0.1) to the sigmoid derivative so it is no longer 0 even if saturated

Note:
By offsetting the targets the weights will not grow too large in attempt to achieve high precision (exactly 0 or 1)

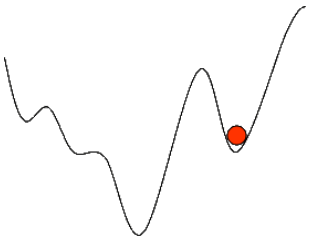
Local minima in the error function

- Error function can easily have more than one minimum
- If learning starts in vicinity of a local minimum, it will likely end in this minimum

Solutions:

- Starting with a range of different initial weight sets increases chance to find the global minimum
- Variation from the true gradient descent (e.g. online learning) also increases chances of getting to the g. m.

- Other methods (simulated annealing, etc.)



When to stop training

- Sigmoid takes on its extreme values 0 and 1 for $tot = \pm\infty$
- To achieve the binary targets, at least some weights would have to reach ∞, which is impossible in finite time due finite step size
- This is true even for off–set targets 0.1 and 0.9 as the gradient is increasingly getting close to a (flat) minimum
- Solutions:
  - Stop when the approximation is good enough (e.g. ± 0.1) from targets; this depends on the problem to be solved
  - Stop when the training error (SSE or MSE) becomes less than a particular small value (again, problem dependent)
  - Use validation data to ensure generalization

Number of hidden units

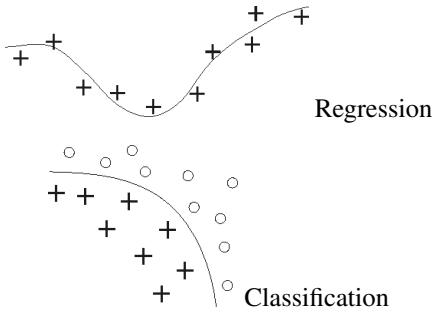
- This depends on many factors
  - Number of training patterns
  - Number of input and output units
  - Amount of noise in training data
  - Complexity of function or classification to be learned
  - Type of hidden unit activation function
  - Training algorithm
- FF NN with a *single layer* can approximate any continuous function - the number of neurons in that layer, however, depends on the degree of nonlinearity
- The smallest network that can perform the desired mapping provides the best generalization

How many hidden units?

- Too few hidden units – high training and generalization errors (under–fitting)
- Too many hidden units – low training errors but slow learning and poor generalization (over–fitting)
- Solutions (rules of thumb usually do not work)
  - Trying a range of numbers of hidden units and observing the training process to find which works best
  - Network construction methods
    - Pruning: large $\rightarrow$ small
    - Growing: small $\rightarrow$ large
  - Network regularization including weight term $\sum_j w_{ij}$ in the error function causes the network to have smaller weights and this forces network response to be smoother and less likely to overfit

Generalization

Ability to predict function output or class for inputs that the network has not encounter before (during training)



Overtraining and specificity

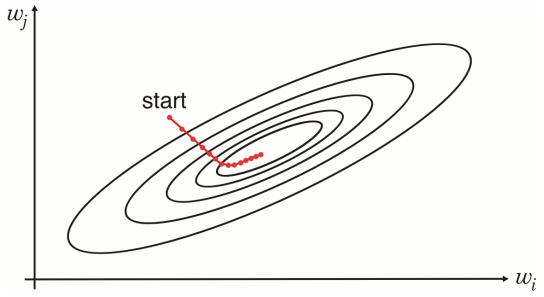
- Breaks generalization: network first learns rough structure of data; as it continues to learn, it will memorize the details (pick up noise)
- Indication that *memorization* occurs $E_v > \overline{E}_v + \sigma_{E_v}$
- Validation error E_v is often calculated on a set of *validation data* - similar to normal training set, but not used in training.
- To reduce chances of memorization - have many more training data points than parameters in the network

Learning rates variation

- Network as a whole learns most efficiently if all neurons learn at roughly the same speed – perhaps different parts of the network should have different learning rates:
  - Later layers (close to outputs) tend to have larger local gradients than layers closer to inputs
  - Activation of units with many connections (both incoming and outgoing) tend to change faster
  - Activations for linear units are different than those for sigmoids
- Approaches:
  - In practice it is usually easier/faster to have the same rates for all units;
  - Evolutionary approaches are powerful for tuning η (as opposed to manual design based on the factors)

Learning visualization

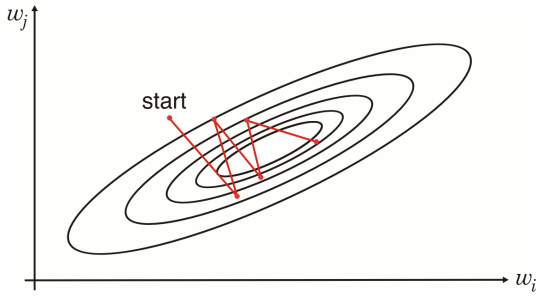
Visualizing learning in 2D to get the idea what is happening



Gradient descent produces a smooth curve perpendicular to the contours; small step η provides good approximation to true gradient descent

Step size too large

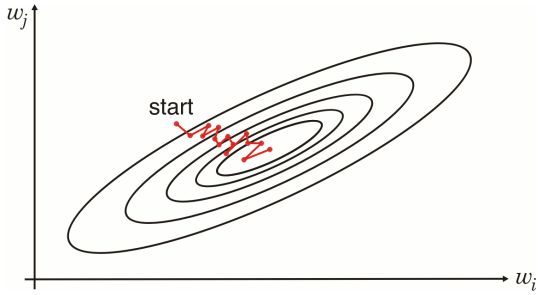
Approximation to true gradient descent is poor and the process ends up with overshoots or even divergence



This is also visible from error graph

Online learning

- Weights are updated after each pattern – no longer to true gradient descent; weight changes are not perpendicular

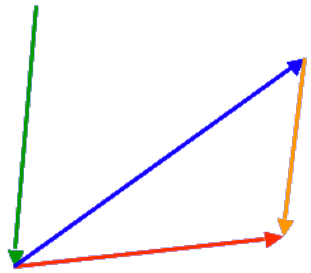


- If η is small enough, the erratic behavior is on a small scale and not much of a problem – increased number of weight changes will lead to the minimum faster than true gradient descent (batch learning)

Learning with momentum

- A compromise to smooth the erratic behaviour of the on-line updates, without significant slow–down
- Weights are updated with the moving average of the individual weight for single training patterns

$$\Delta w_{ij}(t) = -\eta \Delta(w) E(w) + \gamma \Delta w_{ij}(t-1)$$



3.4.5 Data preparation

Data preparation

- This discussion applies to all supervised learning methods
- Preprocessing data assists in convergence and developing an appropriately generalized networks
- Obviously if at all possible irrelevant inputs should be eliminated (training data should be *representative* and *consistent*)
- Preparing a set of training data may include
  1. Handling missing values
  2. Coding of input values
  3. Handling outliers
  4. Scaling and normalization
  5. Noise injection
  6. Manipulation of training set

1. Handling missing values

- eliminate the pattern;
- fill in using statistical methods; or
- add extra network input to indicate "data missing"

2. Coding of input values

- all variables must be numeric;
- enumerated inputs can be converted to unweighted binary inputs

$$\text{red} \in [0,1], \text{green} \in [0,1], \text{blue} \in [0,1]$$

- conversion to a continuous numerical value is possible but the discrete nature of the data is lost

$$\text{red} = 0.0, \text{green} = 0.5, \text{blue} = 1.0$$

3. Handling outliers

- data that deviates from the norm affects error surfaces, and therefore weight updates, drastically
- To handle:
  - remove the outliers before-hand using statistical methods;
  - use an error calculation that minimizes the effects of outliers (robust objective function); and/or
  - remove the outliers during training using, e.g. the frequency distribution of the error

4. Scaling and normalization

Data must be within limits imposed by the activation functions; if close to the limits, weight updates are small and convergence slow (e.g. for sigmoid, use $(0.1, 0.9)$ rather than $(0, 1)$)

- Amplitude (min-max) works but the smaller range (smaller error) leads to longer training times

$$v_s = \frac{v_u - v_u^{\min}}{v_u^{\max} - v_u^{\min}}(v_s^{\max} - v_s^{\min}) + v_s^{\min}$$

- Mean centering shifts the input and output pairs in the training data by their mean value across all patterns. Each individual input and output is scaled $v^{\text{new}} = v^{\text{old}} - \overline{v}$
- Variance scaling is also performed on each input and output individually $v^{\text{new}} = \frac{v^{\text{old}}}{\sigma_v}$ (σ_v is the standard deviation of the input or output across all training patterns)

4. Scaling and normalization (continued)

- Normalizing to unit length: some NNs require that the input vectors be of unit length
  - $x^{\text{new}} = \frac{x^{\text{old}}}{\sqrt{\sum x_i^2}}$
  - Problem: vectors in the same "direction" will normalize to the same input vector (magnitude is lost)
- Z-axis normalization preserves magnitude
  - Input values are scaled to $[-1, 1]$ and then normalized by $x_i^{\text{new}} = \frac{x_i^{\text{old}}}{\sqrt{m}}$
  - finally, a synthetic parameter is fed to the network through an extra input unit $x_0 = \sqrt{1 - \frac{D^2}{m}}$, where m is number of inputs and D is the Euclidean length of input vector

5. Injecting noise

It is possible to generate new tarining patterns by injecting noise into the inputs. The error must be random on a normal distribution with relatively small variance.

6. Manipulating training set

There are several approaches outlining how the training data can be presented to the NN in a bid to speed up learning and increase accuracy.

- Selective presentation (fig L)
- Increased complexity training (fig R)
- Delta subset training (ordered by intra-pattern distance)
- Reduction of the training set (for enormous data sets)
- Randomization

3.5 Radial basis function networks

Function approximation [in this case function $f(x) = x\sin(x)$] in MLP networks is realized by composing segments of sigmoidal functions. The quality of the approximation is influenced by the number of hidden nodes (i.e. the number of sigmoids available for composition) and by the amoun (and dsitribution) of thraining patterns (i.e. the number of points presenting the characteristics of the function to the network, a.k.a. sampling rate).

Radial Basis Function (RBF) Networks
RBF networks are based directly the theory of function approximation
They posses the following characteristics:

- They are two-layer feed-forward networks.
  - hidden nodes implement a set of radial basis functions
  - output nodes perform linear combination of hidden signals
- The network training is organized into two stages
  - first the parameters of hidden units are determined;
  - then the weights from the hidden to output layer
- The training/learning is very fast
- The networks are very good at interpolation

3.5.1 Exact interpolation

Exact interpolation
Exact interpolation of a set of n points: each input vector $\boldsymbol{x}(k) = [x_1(k), x_2(k), \ldots, x_m(k)]$ is mapped onto its corresponding target output $t(k)$. The goal is to find a function $f(\boldsymbol{x})$ such that

$$f\big(\boldsymbol{x}(k)\big) = t(k),\ \forall k = 1, 2, \ldots, n$$

RBF approach uses a set of n basis functions, one for each data point, in form $\Phi(\|\boldsymbol{x} - \mu_k\|)$, where $\Phi(\cdot)$ is specific non–linear function. The output mapping is then obtained as a linear combination of the basis functions

$$f(\boldsymbol{x}) = \sum_{k=1}^{n} w_k \Phi(\|\boldsymbol{x} - \mu_k\|)$$

The goal of exact approximation is to find weights w_k such that the function passes through all data points

Weight determination
We can write

$$f(\boldsymbol{x}(l)) = \sum_{k=1}^{n} w_k \Phi\big(\|\boldsymbol{x}(l) - \mu_k\|\big) = t(l)$$

and thus

$$\boldsymbol{w}\boldsymbol{\Phi} = \boldsymbol{t}$$

Standard matrix inversion techniques can be used to obtain the weights

$$\boldsymbol{w} = \boldsymbol{\Phi}^{-1}\boldsymbol{t}$$

With such set of weights, the function $f(\boldsymbol{x})$ represents a continuous surface that passes *exactly* through each data point.

Common RBFs

1)	Gaussian function	$\Phi(v) = \exp(-\frac{v^2}{2\sigma^2})$
2)	Inverse mq function	$\Phi(v) = (v^2 + \sigma^2)^{-\frac{1}{2}}$
3)	Generalized inverse mq function	$\Phi(v) = (v^2 + \sigma^2)^{-\beta}$
4)	Multi–quadratic function	$\Phi(v) = \sqrt{v^2 + \sigma^2}$
5)	Generalized mq function	$\Phi(v) = (v^2 + \sigma^2)^{\beta}$
6)	Thin plate spline function	$\Phi(v) = v^2 \ln(v)$
7)	Cubic function	$\Phi(v) = v^3$
8)	Linear function	$\Phi(v) = v$
	mq $\sim$ multi-quadratic	$\sigma, \alpha > 0;\ 0 < \beta < 1$

Nice list, but only FYI

It has been shown that properties of the interpolating function are relatively insensitive to the precise form of the basis functions $\Phi(v)$.

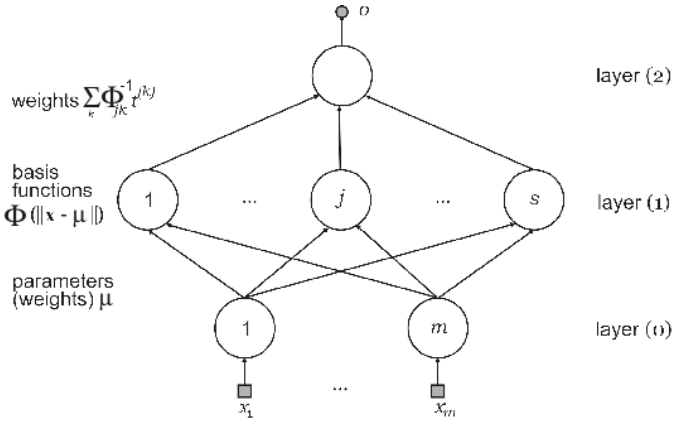
Properties of RBFs

Some (1–3) are localized: $\Phi(v) \to 0$ as $v \to \infty$,
while others (4–8) are not: $\Phi(v) \to \infty$ as $v \to \infty$

All are non-linear in terms of inputs (including the linear function $\Phi(v) = v = \|\boldsymbol{x} - \boldsymbol{\mu}\|$).

- For neural network mappings, localized functions are more suitable (due to the linear combinations of hidden signals in the output layer)
- Gaussian basis function is the most commonly used.

Exact interpolation as NN



Weights are determined directly by n training patterns $\{\boldsymbol{x}, t\}$

Shortcomings of Exact Interpolation

- Poor performance with noisy data (the mapping passes through all data points presented)
- Poor computational efficiency (one hidden unit for each training pattern)
- Possible improvements (to get "real" RBF networks):
  - There can be more than one output neuron, $q > 1$
  - Number of hidden units s could be less then the number of training patterns n (and likely $s \ll n$)
  - Centres of BF could be determined by learning (not set by input data vectors)

- BF can have variable width (a.k.a. spread σ)
- Bias parameter could be introduced in the output layer to allow correct mappings without extreme weights

RBF Network
A more general (and powerful) RBF can be described as follows:

$$o_l(\boldsymbol{x}) = w_{l0} + \sum_{j=1}^{s} w_{jl}\Phi_j(\boldsymbol{x})$$

or, by introducing an extra BF $\Phi_0 = 1$

$$o_l(\boldsymbol{x}) = \sum_{j=0}^{s} w_{jl}\Phi_j(\boldsymbol{x})$$

The remaining BFs are usually in the form of Gaussian

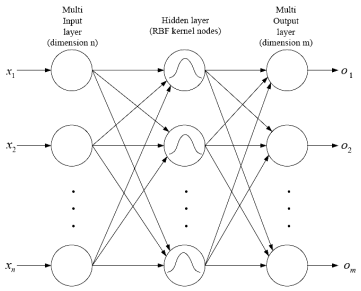
$$\Phi_j(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right)$$

Network parameters
How to determine the values of parameters w_{kj}, $\boldsymbol{\mu}_j$, and σ_j?

3.5.2 Topology

RBF Topology

- Feedforward networks with
  - unity input weights,
  - single hidden layer,
  - specialized activation functions (RBF), and
  - output layer performing linear combination of hidden values



Performance of RBF Networks
It has been shown that linear superposition of localized basis functions provide *universal approximation* capability

3.5.3 Training

Training RBF networks
The proof about universal approximation capabilities of RBF networks are existential (as in the case of MLP) and they do not propose any mechanism how to find structure and parameters of such networks. Unlike in MLP:

- the weights $\boldsymbol{\mu}_j$ represent points in m–dimensional input space to which network inputs are compared using a distance measure
- the weights w_{jl} provide coefficients weighting individual BF in their linear combination

RBF is a two–stage learning process that uses algorithms different from those used in MLPs

BF optimization

- The first stage of the learning process
- Designed to set the parameters of BF

- There are several possible approaches
  1. Random selection of fixed centres
  2. Orthogonal least squares
  3. K–means clustering

- They are all unsupervised techniques
- Remaining problem: determining appropriate value of s (the number of BFs)

BF optimization: random selection
Simplest (and quickest) approach

- BF centres fixed at $s < n$ points selected at random from n training data points;

$$\Phi_j \boldsymbol{x} = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right); \text{where } \{\boldsymbol{\mu}_j\} \subset T$$

- BF widths fixed at an appropriate size $\sim$ distribution of training data

$$\sigma_j = \frac{d_{\max}}{\sqrt{2s}}, \text{ or } \sigma_j = 2d_{\text{avg}}$$

where $d_{\max}$ and d_{avg} are maximum and average distance between chosen centers $\boldsymbol{\mu}_j$, respectively.

- Individual BFs are not too wide or narow
- Works well for large training data sets

BF optimization: orthogonal least squares
More advanced approach

- BF are added sequentially, each centered on a data point
- At each step, each potential l–th BF is tested using remaining $n - l$ data points
- BF providing smallest residual output error is selected (although targets for hidden neurons are unknown, the potential change in error can be determined using *projection matrix* which describes relation between desired and observed outputs for optimal, yet unknown, weights)
- To obtain good generalization, cross–validation is used to determine when to stop the addition process

From a set of orthogonal vectors in the space of hidden unit activations, data points for selecting the BF can be calculated directly

BF optimization: K-means clustering

Considers distribution of training data more accurately

- Select K centres at random
- Iteratively re-estimate to partition the training data into K disjoint subsets S_j containing N_{S_j} data points to minimize

$$\text{clustering function} \quad J = \sum_{j=1}^{K} \sum_{k \in S_j} \|\boldsymbol{x}(k) - \boldsymbol{\mu}_j\|^2$$

  where $\boldsymbol{\mu}_j$ is the centroid (mean) of the data points in set S_j

$$\boldsymbol{\mu}_j = \frac{1}{N_{S_j}} \sum_{k \in S_j} \boldsymbol{x}(k)$$

- Finally, σ_j are set according to variances of points in individual clusters.

Determining the output weights

$\Phi_j(\boldsymbol{x}, \boldsymbol{\mu}_j, \sigma_j)$ are fixed – we have a single–layer linear network SSE same as for MLP $E = \frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{q} [o_l(k) - t_l(k)]^2$ but outputs are simple linear combination of hidden activations

$$o_l(\boldsymbol{x}(k)) = \sum_{j=0}^{s} w_{jl} \Phi_j (\boldsymbol{x}(k))$$

At the minimum of E, the gradients with respect to weights

$$\frac{\partial E}{\partial w_{li}} = \sum_{k=1}^{n} \left(\sum_{j=0}^{s} w_{jl} \Phi_j (\boldsymbol{x}(k)) - t_l(k) \right) \Phi_i(\boldsymbol{x}(k)) = 0$$

which can be solved analytically !

Calculating the output weights

The equation minimizing the gradient can be written in matrix form

$$\boldsymbol{\Phi}^{\mathrm{T}} \left(\boldsymbol{\Phi} \boldsymbol{W}^{\mathrm{T}} - \boldsymbol{T} \right) = 0$$

with the formal solution $\boldsymbol{W}^{\mathrm{T}} = \boldsymbol{\Phi}^{+} \boldsymbol{T}$, where $\boldsymbol{\Phi}^{+}$ is pseudo inverse of matrix $\boldsymbol{\Phi}$

$$\boldsymbol{\Phi}^{+} \equiv (\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^{\mathrm{T}}$$

which can be solved by fast linear matrix inversion algorithms.

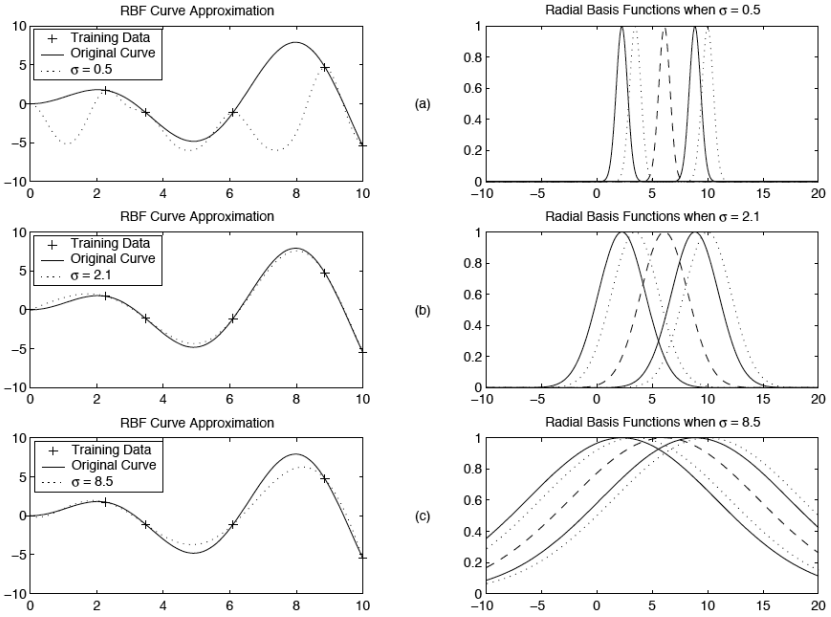
Properties of RBF

- Outputs provide a degree of membership to a class (cluster of data) when used to categorize
- The algorithm does not get stuck in local minima (as is the case with gradient descent learning)
- The first (unsupervised) learning stage is not sensitive to the order of training data

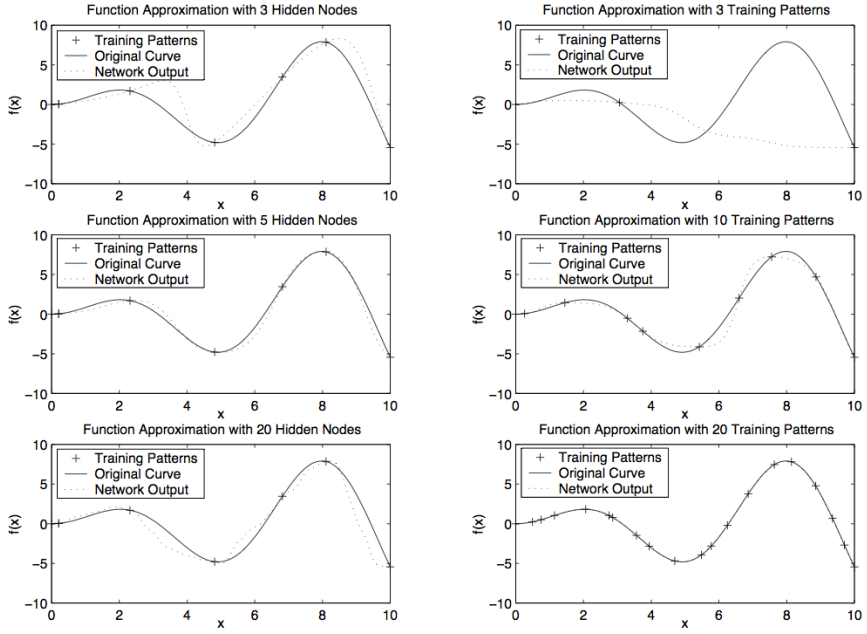
MLP vs. RBF

	MLP	**RBF**
Classification	hyperplanes	hyperspheres
Learning	distributed	localized and faster
Structure	1 or more hidden layers	1 hidden layer but more hidden neurons

Function approximation with RBF Networks



Function approximation with MLP Networks



3.6 Associative Networks

3.6.1 Hebbian Learning

Donald Hebb (1949)
As neuron i becomes more efficient in stimulating neuron j during training

- i sensitizes j to its stimulus
- Weight w_{ij} increases

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \eta x_i o_j$$

Problems?

Modified Hebian learning (Grossberg)
Allows weights to decrease when not longer stimulated

$$\begin{aligned} w_{ij}^{\text{new}} = w_{ij}^{\text{old}}(1-\alpha) &+ \eta x_i o_j \\ \text{forgetting term} &+ \text{Hebbian term} \end{aligned}$$

$$\frac{w_{ij}^{\text{new}} - w_{ij}^{\text{old}}}{\Delta t} = \frac{\Delta w_{ij}}{\Delta t} \rightarrow \frac{dw_{ij}}{dt} = -\alpha w_{ij}^{\text{old}} + \eta x_i o_j$$

i.e. slow exponential weight decay with time constant α

Differential Hebian learning
Allows weights to decrease when the output decreases

$$\begin{aligned} w_{ij}^{\text{new}} = w_{ij}^{\text{old}}(1-\alpha) &+ \eta \frac{dx_i}{dt}\frac{do_j}{dt} \\ \text{forgetting term} &+ \text{differential term} \end{aligned}$$

(rate of change used instead of value)

Note:
No target values used for training - this is an unsupervised learning.

3.6.2 Grossberg Learning

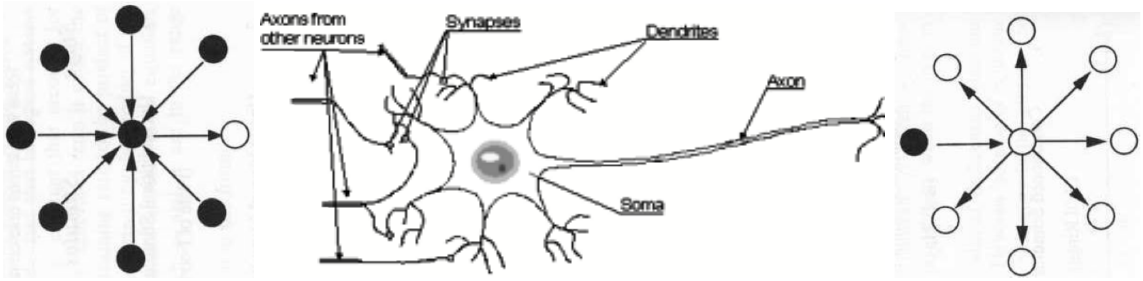
Attempt to mathematically explain psychological conditioning experiments

- Observational conditioning (monkeys)
- Operational conditioning (action/response)
- Classical conditioning (Pavlov)

Pavlov's experiment

Stage	Stimulus	Response
1	Unconditional (food)	Unconditioned (dog salivates)
2	Unconditional (food) PLUS Conditioned (bell)	Conditioned (dog salivates)
3	Conditioned (bell)	Conditioned (dog salivates)

Concepts of instar and outstar:



Instar activity:

1. The activity must grow when there is an external stimulus
2. It must rapidly decrease if it is no longer externally stimulated
3. It must respond to stimuli from other neurons in the network

Instar Learning rule
If inputs and weights are normalized, tot is largest when weights are identical to input values

- weights would be changed only if they are different from the inputs

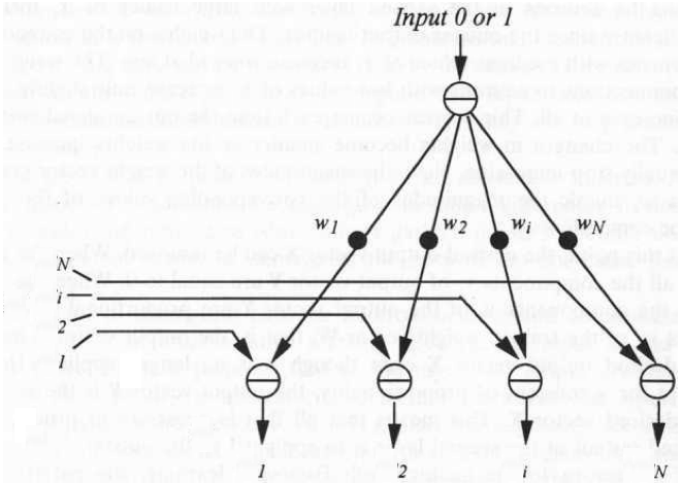
$$\Delta w_i = \eta(x_i - w_i)$$

- incoming weights of neuron converge to input pattern (previous layer)
- unsupervised learning (no targets available)

Outstar Learning rule
Weights connected to certain node should be equal to the desired outputs for the neurons they connect

$$\Delta w_{ij} = \eta(t_j - w_{ij})$$

- outgoing weights of neuron converge to output pattern (next layer)
- neuron learns to recall pattern when stimulated
- supervised learning



3.6.3 Associative Memories (AM)

Systems that store information by associating each data item with one or more other stored data items, (usually) in distributed form.

- Content addressable (CAM)
- Robust
  - Noisy data
  - Incomplete data
  - Failed elements

- Concept: object or pattern $\boldsymbol{x}$ (input) reminds the network of object or pattern $\boldsymbol{o}$ (output)
- Many biological neural nets are associative memories
- Heteroassociative Vs. autoassociative memories
  - If $\boldsymbol{x}$ and $\boldsymbol{o}$ are different, the system is called *heteroassociative* network
  - If $\boldsymbol{x}$ and $\boldsymbol{o}$ are the same, the system is called *autoassociative* nettwork

- Unidirectional vs. bidirectional memories
  - Unidirectional: $\boldsymbol{x}$ reminds you of $\boldsymbol{o}$
  - Bidirectional: $\boldsymbol{x}$ reminds you of $\boldsymbol{o}$ and $\boldsymbol{o}$ reminds you of $\boldsymbol{x}$
- Recognizing new or incomplete patterns
  - Recognizing patterns that are similar to one of the patterns stored in memory (generalization)
  - Recognizing incomplete or noisy patterns whose complete (correct) forms were previously stored in memory

3.6.4 Bidirectional Associative Memory

BAM is a matrix representing a crossbar network with symmetric weights

- the network has two layers
- each neuron in a layer has one output from the outside, and
- inputs from all neurons of the other layer

BAM can store pattern pairs $[\boldsymbol{x}(k), \boldsymbol{o}(k)]$; , $k = 1, \ldots, n$ and recall

- $\boldsymbol{o}$ corresponding to particular $\boldsymbol{x}$, or
- $\boldsymbol{x}$ corresponding to particular $\boldsymbol{o}$
- i.e. it works in both directions $\rightarrow$ *bidirectional*

Setting of BAM weights
Weights are not obtained by training process but directly constructed from input–output pairs, e.g. binary to grey code

$$\begin{array}{lccccccccccccc}
\boldsymbol{x}(1): & (& 0 & 0 & 1 & 0 &) & \Longleftrightarrow & (& 0 & 0 & 1 & 1 &) & :\boldsymbol{o}(1) \\
\boldsymbol{x}(2): & (& 0 & 0 & 1 & 1 &) & \Longleftrightarrow & (& 0 & 0 & 0 & 0 &) & :\boldsymbol{o}(2) \\
\boldsymbol{x}(3): & (& 0 & 1 & 0 & 0 &) & \Longleftrightarrow & (& 0 & 1 & 1 & 0 &) & :\boldsymbol{o}(3)
\end{array}$$

- Before proceeding, the values must be converted to bipolar (0s become -1s while 1s remain)
- From each pair, a mapping matrix $\boldsymbol{M}_k = \boldsymbol{x}(k) \times \boldsymbol{o}(k)$ is constructed
- The matrices are then combined into a master matrix $\boldsymbol{M} = \boldsymbol{M}_1 + \boldsymbol{M}_2 + \cdots + \boldsymbol{M}_n$

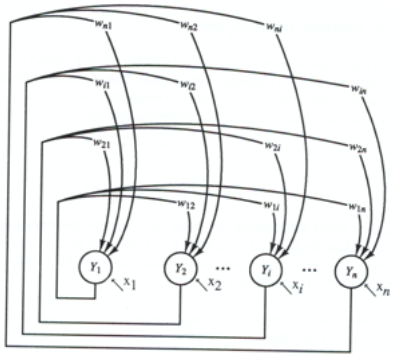
3.6.5 Hopfield Network

- A more advanced type of autoassociative memory
- Almost fully connected

- Architecture
  - symmetric weights

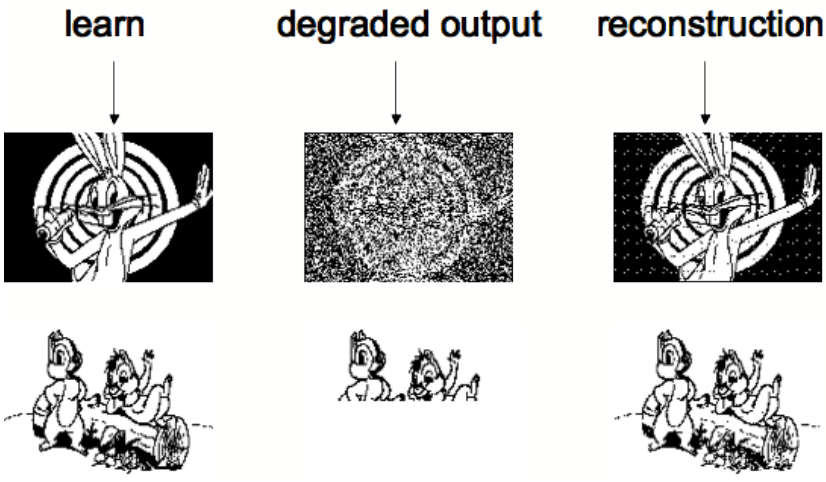
$$w_{ij} = w_{ji}$$

  - notice the "feedback" in the network structure
  - no feedback from a cell to itself $w_{ii} = 0$



Principle
Based on the physical principle that every object (system) seeks a low energy static position (state).
During learning, the network uses input to define the low energy points.
When a similar to a learned position is encountered, some neurons will drag the other neurons to the static position.

Illustration



Operation

- Task: Store images with resolution 20×20 pixels (need a network with 400 nodes)
- Learning:
  1. Present image
  2. Apply Hebb rule: cells that fire together, wire together (increase weight between two nodes if both have same activity, otherwise decrease)
  3. Go back to 1
- Recall:
  1. Present incomplete pattern
  2. Pick random node, update
  3. Repeat 2 until settled

Hebbian rule for Hopfield network
Changes are proportional to the correlation between the firing (activity) of the pre- and post-synaptic neurons.

Technically:

- Consider $T = \{x(k)|x(k) = (x_1(k), \ldots, x_m(k)) \in \{-1, 1\}^m, k = 1, \ldots, n\}$
- Start with $w_{ji} = 0\,(j = 1, \ldots, m;\ i = 1, \ldots, m)$
- For given training set T do

$$w_{ij} = \sum_{k=1}^{n} x_j(k)x_i(k), 1 \leq j \neq i \leq m$$

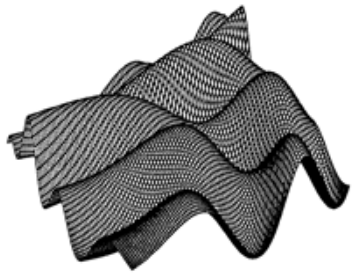
Active Mode of Hopfield Network

1. Set $o_i = x_i (i = 1, \ldots, m)$
2. Go through all neurons and at each time step select one neuron j to be updated according the following rule:
   - compute its internal potential $tot_j = \sum_{i=1}^{m} w_{ij} o_i$
   - set its new state $o_j = \begin{cases} 1 & \text{if } tot_j > 0, \\ o_i & \text{if } tot_j = 0, \\ -1 & \text{if } tot_j < 0. \end{cases}$
3. IF not stable configuration THEN go to step 2 ELSE end - output of the net is determined by the current state of neurons.

Energy Function and Landscape
Energy function $E(o) = -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} w_{ij} o_i o_j$

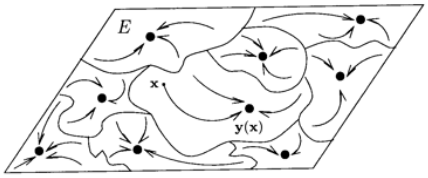
Energy Landscape

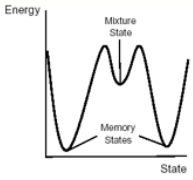
- high energy - unstable states
- low energy - more stable states
- energy always decreases (or remain constant) as the system evolves according to its dynamical rule



Attractors and Phantoms

- Local minima of the energy function represent stored examples - attractors
- Basins of attraction - catchment areas around each minimum
- False local optima - phantoms





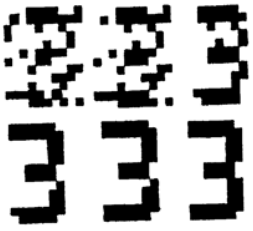
Storage Capacity of Hopfield Network

- Capacity of the network - maximum number of patterns that can be stored without unacceptable errors.
- Empirical results
  - $n \leq 0.138m$ - training examples as local minima of $E(o)$
  - $n < 0.05m$ - training examples as global minima of $E(o)$, deeper minima than those corresponding to phantoms
- Example: 10 tr. examples, 200 neurons $\rightarrow$ 40000 weights

Example

Pattern recognition

- 8 examples, matrix 12×10 pixels $\rightarrow$ 120 neurons
- input pattern with 25% bits incorrect



3.7 Competitive Neural Networks

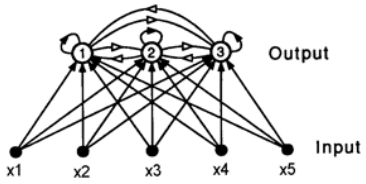
3.7.1 Selforganisation

- Unsupervised learning: system discovers for itself patterns, features, regularities, or categories in the input data and code them in the output
- Units and connections display some degree of selforganisation
- Competitive learning
  - output units compete for being excited
  - only one output unit is active at a time (winner-takes-all mechanism)
- Feature mapping: development of significant spatial organisation in the output layer
- Applications: function approximation, image processing, statistical analysis, combinatorial optimization

Selforganising Network

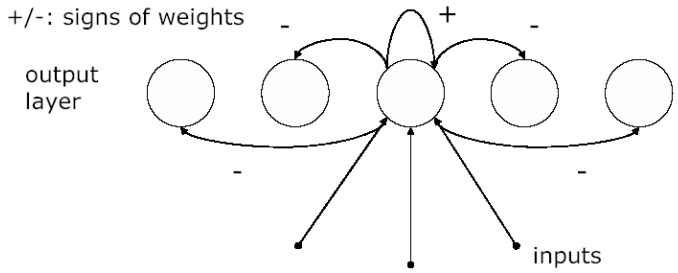
- Goal: to approximate the probability distribution of real-valued input vectors with a finite set of units
- Given a training set T of example patterns (inputs) $\boldsymbol{x} \in \mathcal{R}^n$, and q representatives (outputs) $\boldsymbol{o}_l$, $l = 1, \ldots, q$

- Network topology:
  - weights belonging to one output unit determine its position in the input space
  - lateral inhibition (to facilitate competition)



Competitive dynamics

Single element from a competitive layer: coupling within layer, such that the 'best' wins, or 'the winner takes all'. Achieved by positive feedback to itself, and inhibition of others.

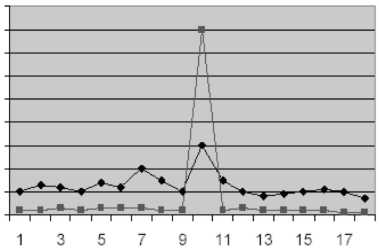


- Suppose a vector $\boldsymbol{x}$ is presented at the input.

- Each unit computes its net input as follows:

$$tot_j = \sum_i w_{ij} x_i + w_{jj}^{+} o_j + \sum_{k \neq j} w_{jk}^{-} o_k$$

- Winner is found after an iterative process
- Implementation can be much simpler!



Competitive learning

Consider:

- training set where $|\boldsymbol{x}| = 1, \forall \boldsymbol{x}$ (all vectors are on the unit hypersphere)
- competitive layer with normalized set of weight vectors, represented as a 3-node network

Desired state after training:

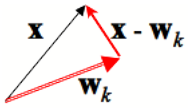
- each weight vector must respond strongly to one of the clusters of input vectors.

This desired state can be derived by:

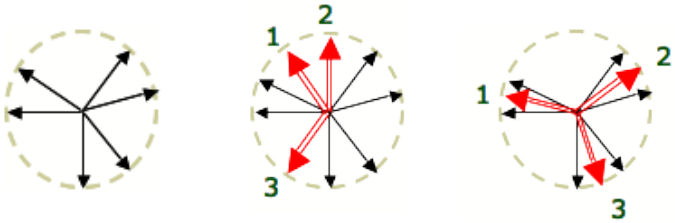
- iteratively applying vectors and finding the winner
  - node with the largest external input; or (in other words)
  - node with weight vector closest to the input vector (Euclidean distance; using argmin)
- adjusting the weights of the winner

Let this "winning" node have index k and weight vector $\boldsymbol{w}_k$. Weight $\boldsymbol{w}_k$ must be rotated towards $\boldsymbol{x}$ by adding a fraction of the difference vector $\boldsymbol{x} - \boldsymbol{w}_k$, i.e. :

$$\Delta \boldsymbol{w}_j = \begin{cases} \eta(\boldsymbol{x} - \boldsymbol{w}_j) & \text{for } j = k \\ 0 & \text{for } j \neq k \end{cases}$$



Example of Competitive learning

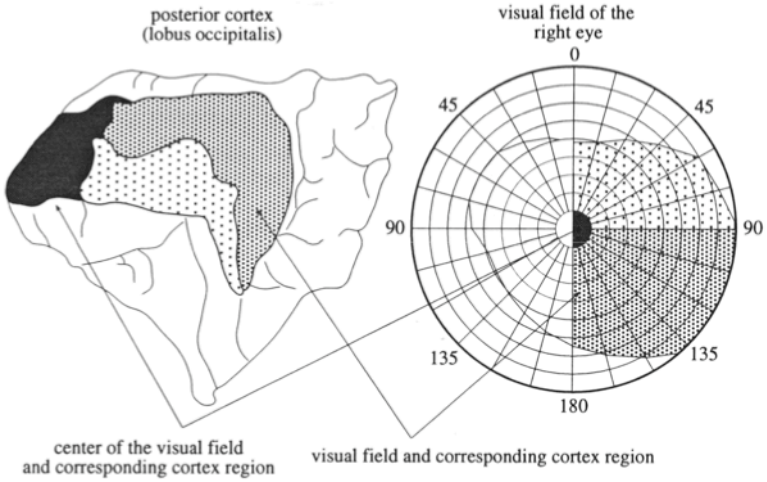


Self-organization in Brain

Formation of feature maps in the brain that have a linear or planar topology

- tonotopic map – sound frequencies are spatially mapped into regions of the cortex in an orderly progression from low to high frequencies.
- retinotopic map – visual field is mapped in the visual cortex (occipital lobe) with higher resolution for the centre of the visual field
- somatosensory map – mapping of touch on the cortex (homunculus)

Mapping of the visual field on the Cortex



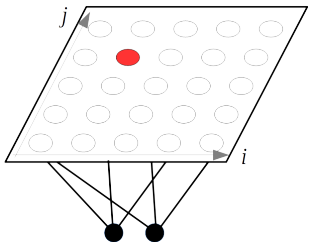
3.7.2 Self-Organizing Maps (SOM)

Purpose: mapping of a multidimensional input space onto a topology preserving map of neurons (often a 2 or 3D space; note the similarity to self-organization in biological systems)

Topology preservation: preserve a topological order so that neighboring output neurons respond to "similar" input patterns

Network structure: similar as described earlier, but:

- no lateral connections
- output units formed in a structure defining neighbourhood
- 1- or 2-dimensional array of units



- Each neuron is assigned a weight vector with the same dimensionality as the input space
- Input patterns are compared to each weight vector and the closest one wins (Euclidean Distance)

$$k = \text{argmin}_i(\|x_i - w_{ij}\|)$$

- Move neuron k and its neighbors closer to the input according to

$$w_{ij} = w_{ij} + \eta(x_i - w_{ij})$$

  for each j such that $D(j,k) < d$ (neighborhood)
- Result of the training process: clustering of data, so that similar inputs appear close in the map space (i.e. implicit categorization discovered without feedback)

Neighborhood function $h_k(j)$

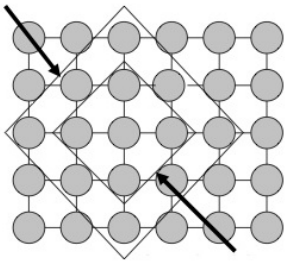
- Neighborhood $N_d(k) = \{j : \ D(j,k) \leq d\}$ defines a set of neurons with distance from k less than d
- In learning algorithm, neighborhood function h_k is used:
  - weight update rule involves neighbourhood relations - the winner as well as the units close to it are changed

$$w_{ij} = w_{ij} + h_k(j)\big(x_i - w_{ij}\big), j \in N_d(k)$$

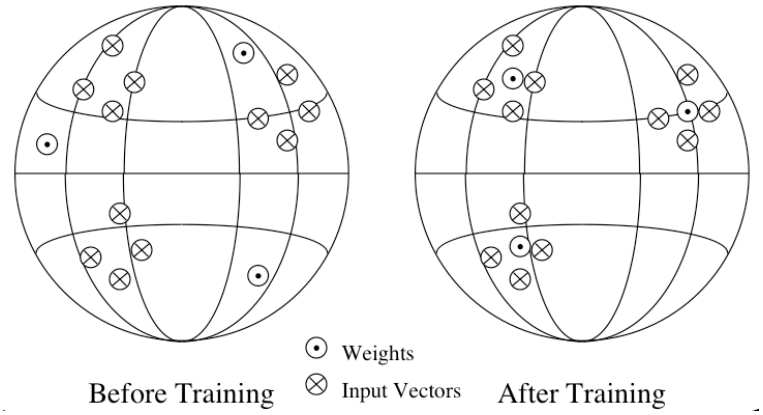
    where $h_k(j) = \eta$ if $j \in N_d(k)$ and $h_k(j) = 0$ otherwise.
  - Gaussian function can be used as $h_k(j)$ so closer units are more affected than those further away

- Neighborhood should get smaller in time (to ensure convergence and specialization of the output nodes), for example when $t_1 < t_2$:

$$t_1 \text{ use } N_2(k), t_2 \text{ use } N_1(k)$$



Typical Convergence

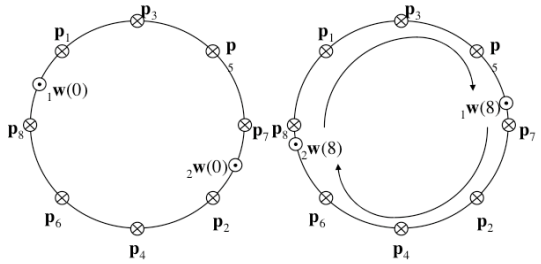


Problems with SOMs

- Stability
- Dead units
- Need to know number of clusters/classes
- Each class must be a convex region

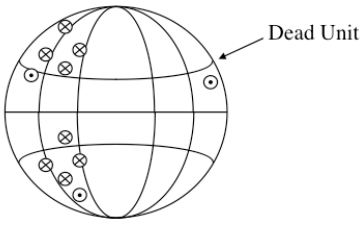
SOM: stability

If the input vectors do not naturally form clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will not converge.



SOM: dead units

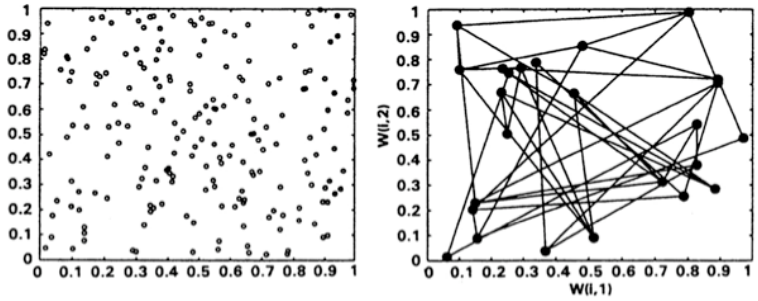
Neurons with initial weights far from any input vector may never win.



Dead unit remedy: “conscience” - neurons are discouraged from being greedy - their probability of being the winner is influenced by how many times they have won previously.

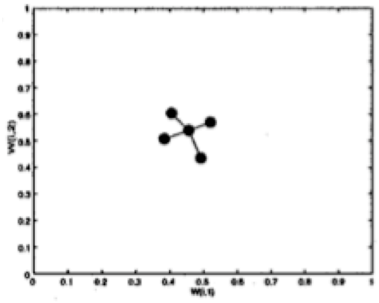
SOM Example – even distribution

200 vectors chosen randomly are shown in the left picture, to train a net of 25 nodes/units on a 5×5 rectangular grid. The initial weights are shown on the right.



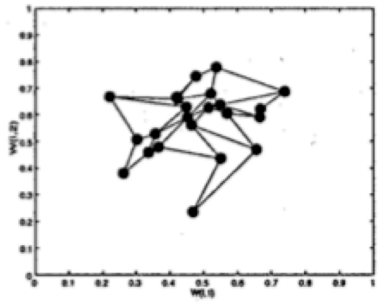
SOM Example – after 60 iterations

All nodes have weights close to the center of the field, which is near the average of the training set. This is the result of selecting a large initial neighborhood, so that nearly all nodes are trained at every step.



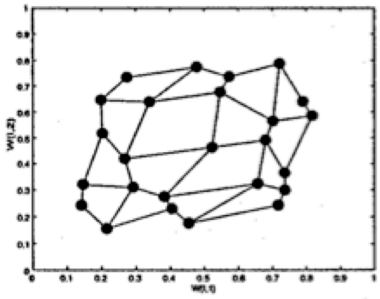
SOM Example – after 300 iterations

As the neighborhood is shrinking, the weight space representation starts to unravel -different parts of the pattern space can start to be encoded by different regions of the net.



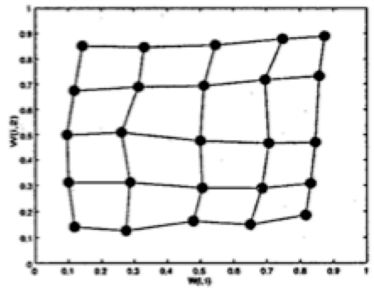
SOM Example – after 600 iterations

The topology is correct – there are no crossing internode connections.



SOM Example – after 6000 iterations

Further training provides fine adjustments, giving the network a better (more even) shape.



3.7.3 Learning Vector Quantization (LVQ)

- SOM algorithm provides an approximate method to compute the Voronoi vectors in an unsupervised manner

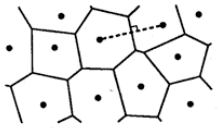


FIGURE 9.4 Voronoi tessellation. The space is divided into polyhedral regions according to which of the prototype vectors (dots) is closest. The boundaries are perpendicular bisector planes of lines joining pairs of neighboring prototype vectors.

- LVQ can be used to add another layer to combine different Voronoi regions to do pattern classification, combining unsupervised and supervised learning (non-convex regions obtained by union of convex regions)

Learning Vector Quantization (LVQ)

- Winning neuron in the competitive layer indicates the subclass which the input vector belongs to
- There may be several different neurons (subclasses) which make up each class
- The second layer of the LVQ network combines subclasses into more complex (non–convex) regions through matrix

$$W^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Subclasses 1,3 and 4 belong to Class 1
Subclass 2 belongs to Class 2
Subclasses 5 and 6 belong to Class 3

LVQ Learning

Combines competitive learning with supervision:

- requires a training set of examples of proper network behavior

$$\boldsymbol{x}_1, \boldsymbol{t}_1, \boldsymbol{x}_2, \boldsymbol{t}_2, \ldots, \boldsymbol{x}_n, \boldsymbol{t}_n.$$

- If the input pattern is classified correctly, then move the winning weight vector toward the input vector according to the Kohonen rule.

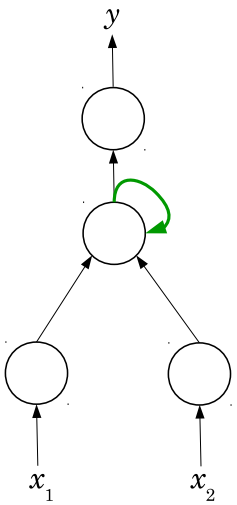
$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \eta[x_i - w_{ij}^{\text{old}}]$$

- If the input pattern is classified incorrectly, then move the winning weight away from the input vector.

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} - \eta[x_i - w_{ij}^{\text{old}}]$$

3.8 Recurrent Neural Networks

Networks with Feedback



The difference this added connection makes:

- activation is now given by
  - current input pattern
  - and previous state of the unit
- States of hidden and output neurons are now functions of everything the network has seen so far (i.e. it has a "sense of history")

RNN Topology

Adding feedback connections – topology becomes very free (any unit to any unit)
Two basic assumptions (so far) are violated

- for computing activations - knowing activations of all posterior units
- for computing errors – knowing error of all anterior units

Need a new approach for training and operation

FFNNs vs. RNNs

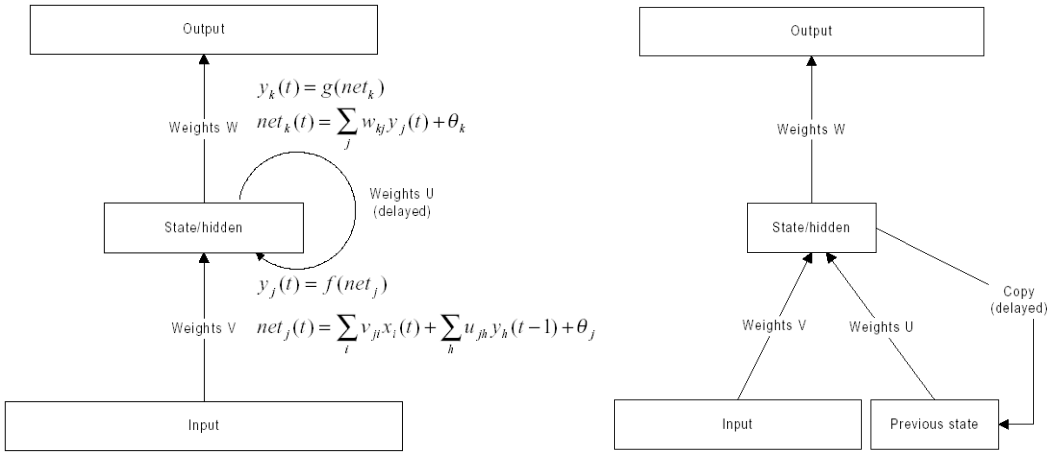
- Feed-Forward Neural Networks (FFNN) can *learn function mappings* (FIR filter)
- Recurrent Networks (RNN) use hidden layer as memory store to *learn sequences* (IIR filter) => RNNs can exhibit virtually unlimited temporal dynamics

RNN Applications

- Grammar induction (recognition of legal strings of symbols)
- Speech recognition (RNN predicts phonemes, predictions fed into HMM for decoding)
- Filtering
- System Control, Identification
- Music Composition

Simple Recurrent NN (Elman, 1990)

Hidden layer activations copied into a context/copy layer



Operation:

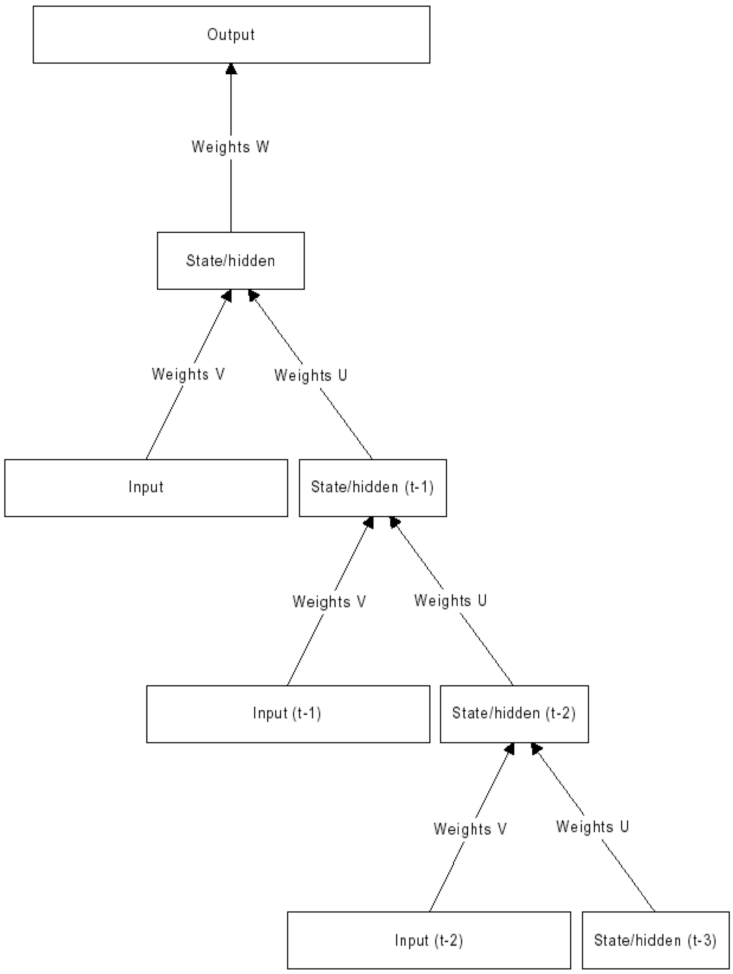
- Copy inputs for time t to the input units
- Compute hidden activations using net input from input units and from copy layer
- Copy new hidden unit activations to copy layer
- Compute output unit activations as usual
- Cycles are eliminated: activations and errors are available according to the original requirements (as for FFNN)

Training:

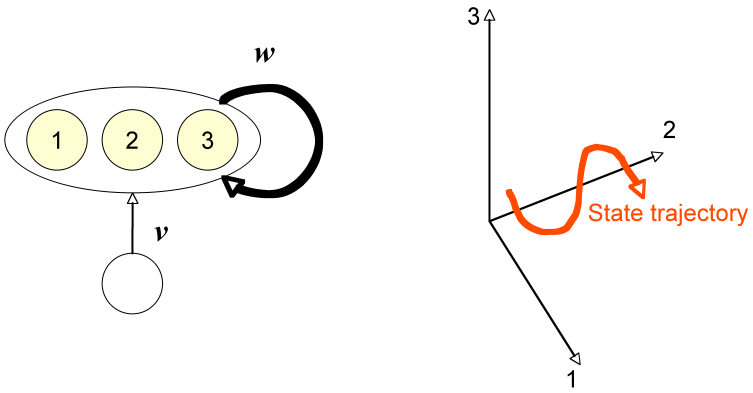
- Need to consider ∇E determined by the activations at the *present* and the *previous* time steps
- Otherwise, standard backpropagation algorithm is retained

Back-Propagation Through Time (BPTT)

Considering more time steps (practical limit 10-12)



Dynamics of RNN



Long-Term and Short-Term Memory

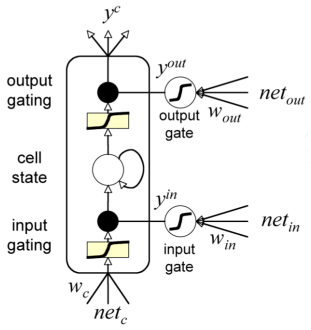
RNN stores information in two different ways:

- Activations (states) – short term
- Weights – long term

. . . each has a different time scale.

LSTM network - attempt to retain information in unit activations (states) for longer period of time (LSTM application for music composition is described at http://www.iro.umontreal.ca/~eckdoug/blues/index.html)

LSTM Memory Block

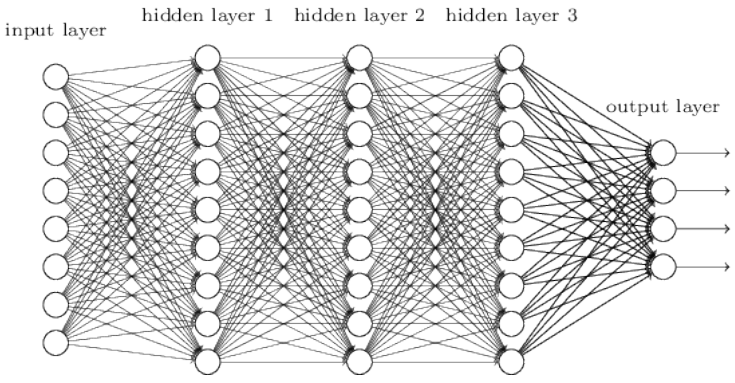


$$y_j^c(t) = y_j^{out}(t)h(s_{c_j}(t))$$
$$s_{c_j}(0) = 0$$
$$s_{c_j}(t) = s_{c_j}(t-1) + y_j^{in}(t)g(net_{c_j}(t))$$

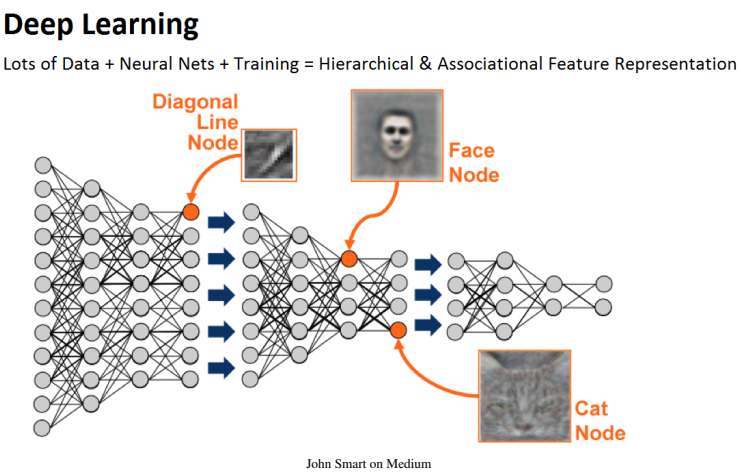
3.9 Deep Neural Networks

Introduction

- In short, a Deep Neural Network (DNN) is a Multi-Layer Perceptron (MLP) with many hidden layers.
- However the field of DNN has developments on its own to improve processing and learning abilities.



Neural Networks and Deep Learning by Michael Nielsen



Early DNNs

Relied on the back propagation algorithm to update each neuron weight individually.

Problems:

- Vanishing gradient error,
- Costly computationally and memory-wise,
- Slow learning,
- And required a large number of training data samples.

Early DNNs - solutions

Solutions to vanishing gradient error:

- Amplify error for each layer by a factor,
- Layer-by-layer learning,
- Use of Rectified Linear Unit in the hidden layers.

Amplification speeds up learning, but the amplification constant is selected manually and its optimal value is unknown. Learning layer by layer also speeds up learning to some extend, however, stopping learning for each layer and adding new layers is done *manually* and there is no target for the hidden layers.

Solutions for other problems:

- None.

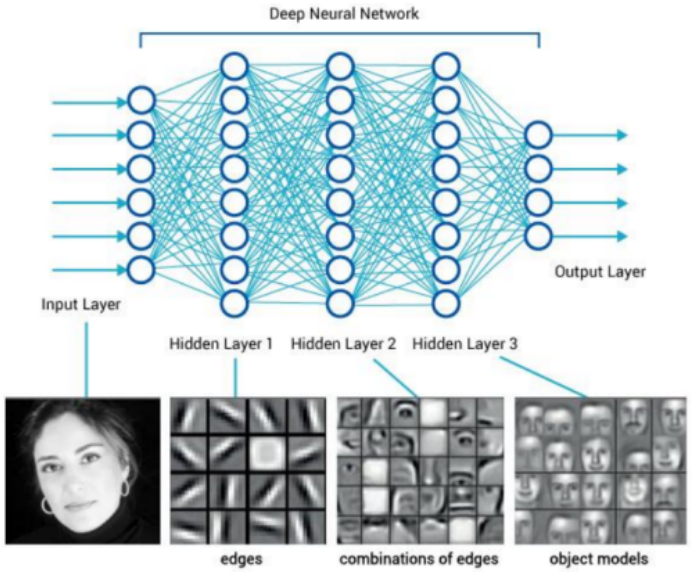
Convolutional Neural Networks (CNNs)

CNN use a shared weight distribution for each layer section, where each section learns a specific pattern/feature of the previous layer. Updates for each section are done based on the overall section error.

- Significantly reduce the memory and computational costs,
- and further accelerates learning.

Additionally, it becomes largely immune to:

- Linear translations,
- and uniform scaling.



Artificial Inteligence by Leonardo Araujo Santos

However, CNNs are still sensitive to rotations, and non-linear scaling and distortions.

Pooling

A lossy compression (subsampling) commonly executed by either averaging the pooled values or by taking the maximum value within the pool. This step is done between convolutional layers (*i.e.* I, C, P, C, P, ..., C, P, O).

Why Reduce the amount of information?

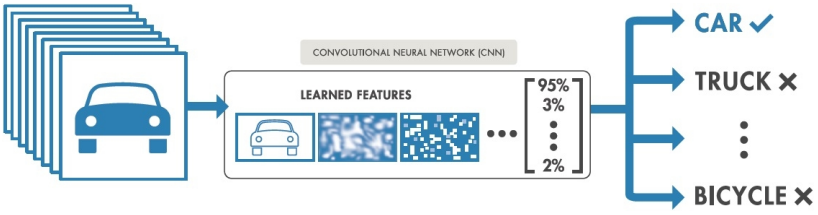
- Increase immunity to non-linear translations and distortions,
- and reduce the likelihood of over-fitting.

How is this different from a convolutional layer with $M \times m < n$?

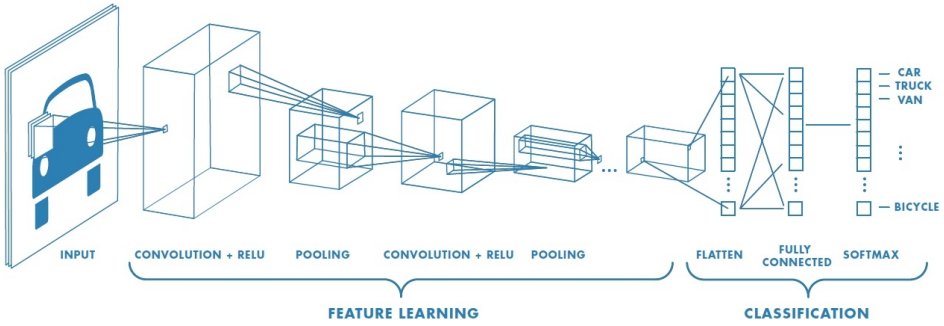
- No learning takes place,
- And the format of each Convolutional layer tends to be identical with some overlapping sources for each neuron. Pooling layers tend to also be identical to one another but do not have overlapping sources for each output.

Deep learning workflow

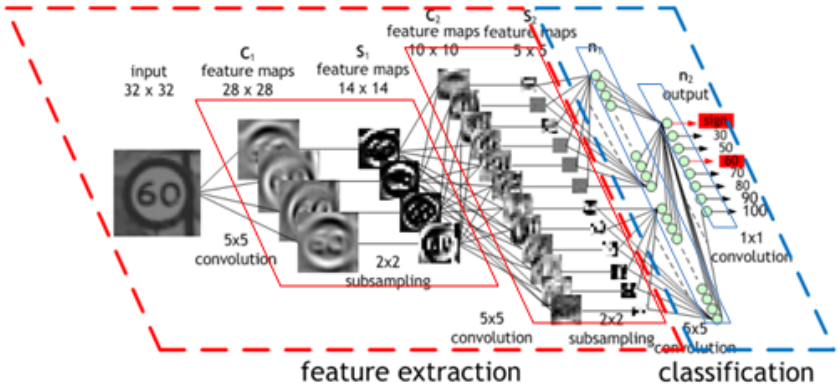
Images are passed to the CNN, which automatically learns features and classifies objects.



https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html



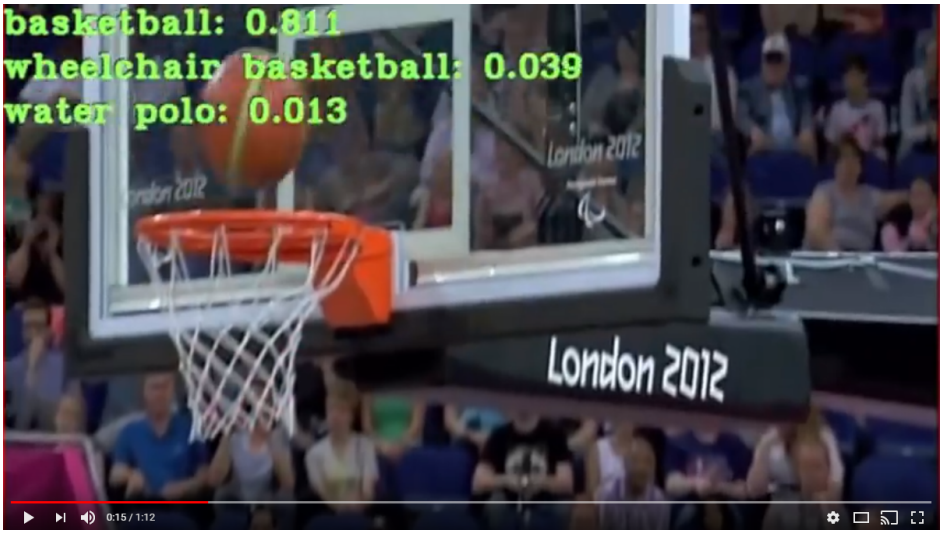
CNN for traffic sign classification



Deep Learning in a Nutshell: Core Concepts by NVIDIA

An image of a traffic sign is filtered by 4 5x5 convolutional kernels to create 4 feature maps subsampled by max pooling. The next layer applies 10 5x5 convolutional kernels to these subsampled images and again we pool the feature maps. The final layer is a fully connected layer where all generated features are combined and used in the classifier.

Example: Video Classification with CNNs



Large-scale Video Classification with Convolutional Neural Networks, CVPR 2014

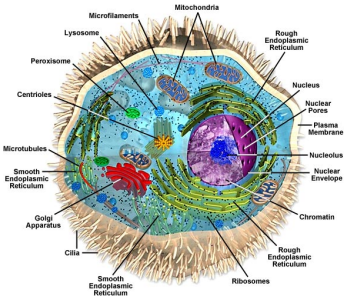
Chapter 4

Evolutionary Computing

4.1 Biological Background

Cells, nuclei

- small factories working together
- the center of each cell is cell nucleus
- nucleus contains genetic information



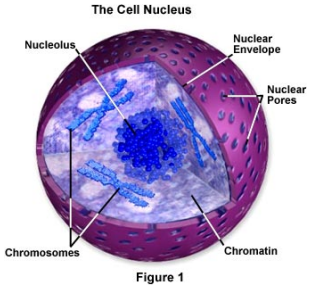


Figure 1

Chromosomes

- structures found in the nucleus of a cell, which contain the genes
- come in pairs
- store information
- built of DNA
- divided into *genes*

Gene

- a unit of inheritance
- a working subunit of DNA
- each of the body's 50,000 to 100,000 genes contains code for a specific product (typically, a protein such as an enzyme)

Genes - alleles

- any of the alternative forms of a gene that may occur at a given position in a chromosome (locus) are called allele
- different alleles produce variations in inherited characteristics such as eye color or blood type

Genetics

Genotype

- the entire combination of genes is called genotype, in other words it is a genetic constitution of an individual

Phenotype

- the visible properties of an organism that are produced by the interaction of the genotype and the environment

Expression

- dominant alleles are expressed from genotype to phenotype
- recessive alleles can survive in the population for many generations without being expressed

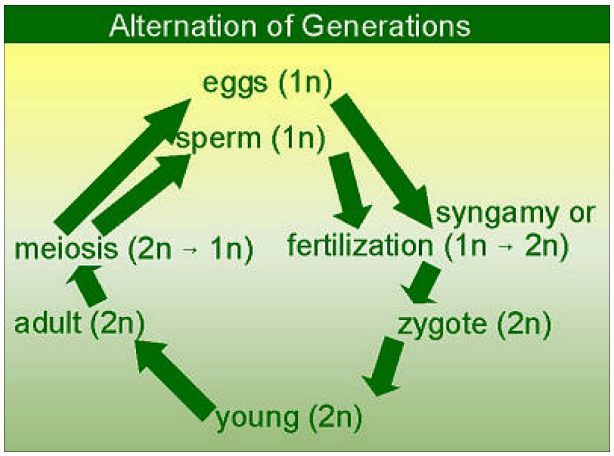
Reproduction

Mitosis: the process that takes place in the nucleus of a dividing cell, and results in the formation of two new nuclei each having the same number of chromosomes as the parent nucleus (no exchange of information)

Meiosis: the process of cell division that reduces the number of chromosomes in reproductive cells from diploid to haploid, leading to the production of reproductive cells – gametes (the nucleus divides into four nuclei each containing half the chromosome number)

Reproduction mechanism: syngamy or fertilization is the union of the egg and sperm to restore the 2n; this results in a zygote, the first cell formed by fertilization, a completely new and different organism with unique genetic information different from either parent

Reproduction cycle



Reproduction "errors"

Recombination (cross-over): a phenomenon that sometimes occurs during the formation of sex cells (meiosis); a pair of chromosomes (one from the mother and the other from the father) break and trade segments with one another

Mutation: a change in the number, arrangement, or molecular sequence of a gene

Natural selection

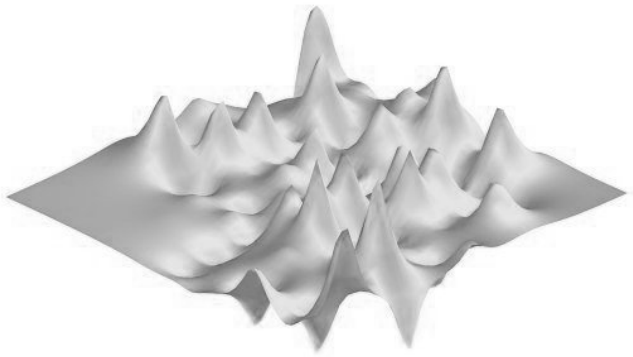
The origin of species: "preservation of favorable variations and rejection of unfavorable variations"

- there are more individuals born than can survive, so there is a continuous struggle for life
- individuals with an advantage have a greater chance for survive – survival of the fittest

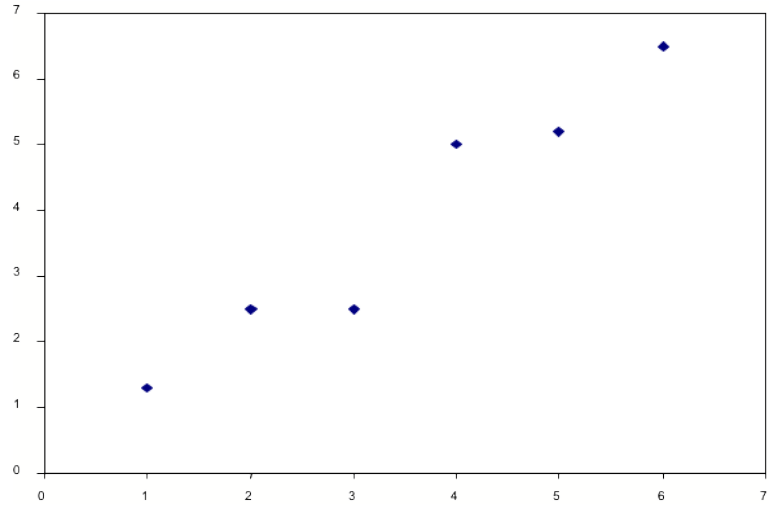
4.2 Overview of EC

Search and Optimization
Optimization: search for the best set of values of adjustable parameters

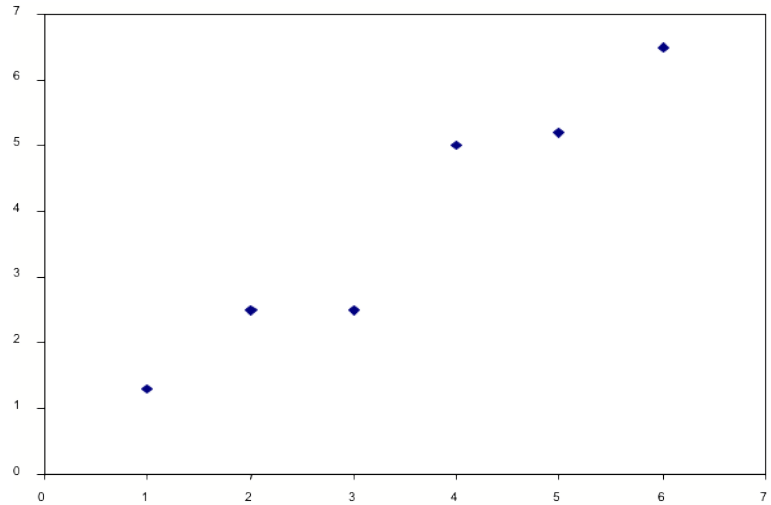
Search space: representation of relationship between a measure of success of the search and the values of the parameters – e.g. error landscape



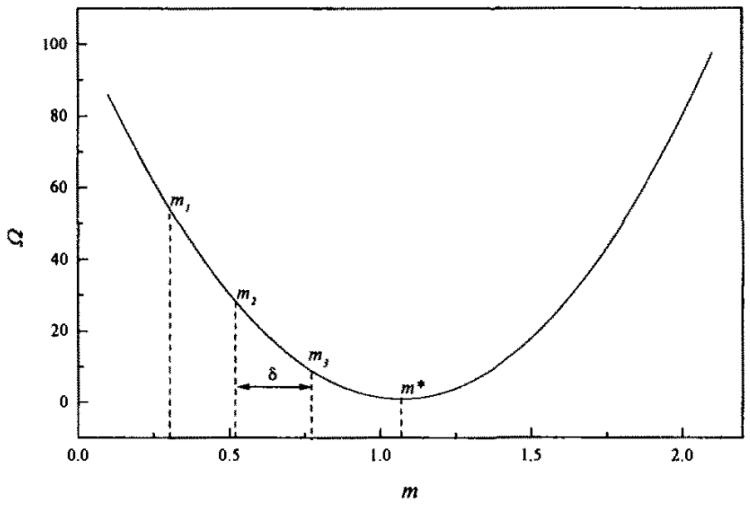
Fitting the experimental data



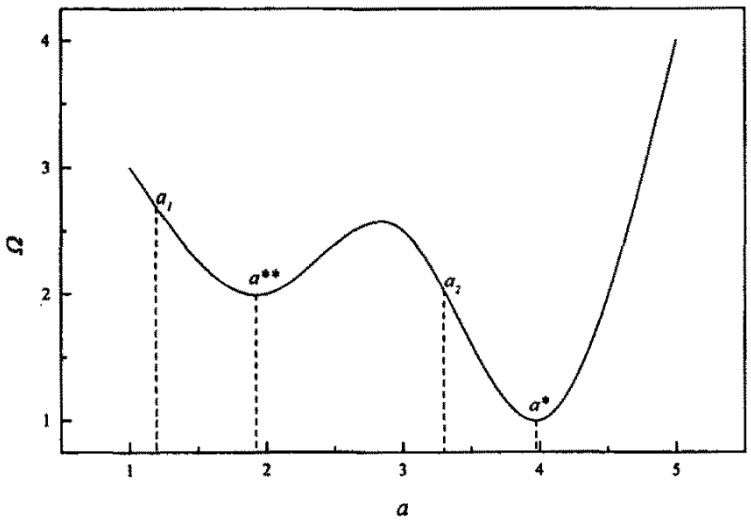
Graphical search: use ruler and visually estimate the best line through the points



Enumerative search: use estimation (e.g. least-squares) at many points and select value of parameter(s) that yields minimal error

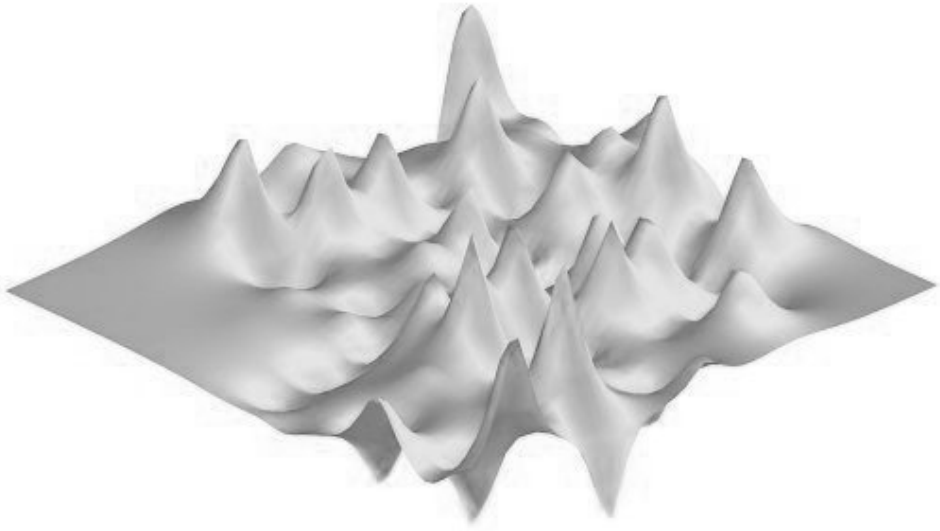


Direct search: estimate at two points and depending on the relation between their quality, go to third point at distance d left or right of the second/first point



What if we have local minima? Start direct or derivative based search from multiple points and assume that the lowest result is the global minimum

What if we have a complex search space?



Evolutionary computing

Evolutionary Algorithms
Stochastic search algorithms based on abstraction of the processes of Darvinian evolution.

There are several types of EAs, all sharing the following basic components:

- they work with population of individuals (candidate solutions), rather than a single candidate solution at a time
- they use selection method biased by fitness (measure of quality of the candidate solutions)
- they generate new individuals via mechanisms of inheritance from existing individuals, mainly through the use of probabilistic operators of crossover and mutation.

Darwin's hypotheses (Origin of Species, 1859)

1. Earth is very old, and organisms have been changing steadily through the history of life.
2. All organisms are descendants of a single common ancestor.
3. Species multiply by splitting into daughter species; such speciation has resulted in the great diversity of life found on Earth.
4. Evolution proceeds via gradual changes in populations, not by the sudden production of individuals of dramatically different types.
5. The major agent of evolutionary change is natural selection.

Types of EAs
Evolution Strategies (ES) individuals: real value vectors; mutation and recombination are used; individuals represent also self–adaptive parameters controlling mutation distribution.

Evolutionary Programming (EP) individuals: originally FSMs; recently also real–valued vectors; do not use crossover; mutation also self-addaptive.

Genetic Algorithms (GA) individuals: binary or real–valued vectors; crossover is the main search operator, mutation minor; follow more the natural genetic mechanisms (ES and EP) are more concerned with behavioral aspects of individuals).

Genetic programming (GP) individuals: computer programs or functions; otherwise quite similar to GAs.

Pseudocode of EAs

GA and GP

```
create initial population
compute fitness of individuals
REPEAT
  select ind. based on fitness
  apply gen. operators to
    selected individuals,
    creating offspring
  compute fitness of offspring
  update the current population
UNTIL (stopping criteria)
```

ES and EP

```
create initial population
compute fitness of individuals
REPEAT
  apply gen. operators to
    individuals, creating
    offspring
  compute fitness of offspring
  select ind. based on fitness
  update the current population
UNTIL (stopping criteria)
```

Key design issues of EAs

Fitness function: should measure quality of individuals as precisely as possible; fine grained.

Individual representation: suitable for given problem (binary, integer, real, tree, ...)

Genetic operators: must be appropriate and balanced with other components of the system

Exploration vs. exploitation: appropriate tradeoff must be chosen

Selective pressure: various selection methods put different pressure on the population (how long does it take until the population is taken over by strong candidates)

4.3 Genetic Algorithms

- search algorithms based on the mechanics of natural selection and natural genetics (Goldberg '89)
- exploit the principle of the survival of the fittest and natural evolution to create a novel and innovative search strategy;
- generally characterized by:
  - Population-based optimization techniques
  - Stochastic reproduction
  - Fitness function

GAs Applications

- Scheduling problems (e.g, job-shop scheduling, traveling salesman problem)
- Optimization of network topologies
- Resource allocation over a distributed system
- Electronic circuit design
- Aircraft design
- Game playing
- Training of fuzzy systems or artificial neural networks

4.3.1 Terminology

GAs: Terminology

Population of potential solutions: the collection of solution points (also called individuals)
Chromosome: an individual solution represented as an array or sequence of strings (also called genotype)
Gene: a string in the chromosome
Alleles: the values, or the states, that genes might have (0 or 1 in the case of binary encoding)
Locus: the position of a gene in a chromosome.

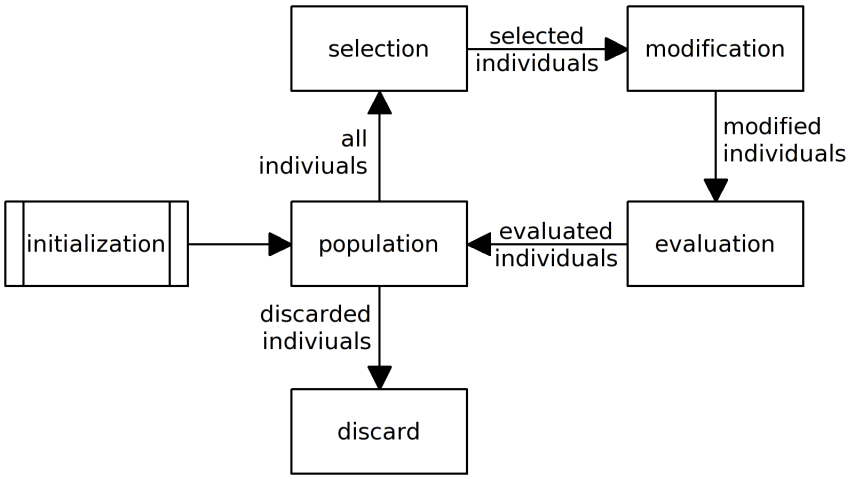
Main Components of a GA
A problem to solve, and ...

- Encoding method (*genes, chromosomes*)
- Initialization procedure (*population creation*)
- Evaluation function (*fitness*)
- Selection of parents (*reproduction*)
- Genetic operators (*mutation, recombination*)
- Parameter settings (*practice and art*)

Simple Genetic Algorithm

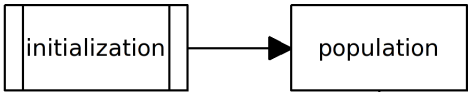
```
create initial population;
compute fitness of individuals;
REPEAT
  select ind. based on fitness;
  perform reproduction and mutation;
  update the current population;
UNTIL (stopping criteria)
```

GA - execution cycle



Population

At the beginning of the cycle, population is initialized; its mebers represent candidate solutions to the problem at hand

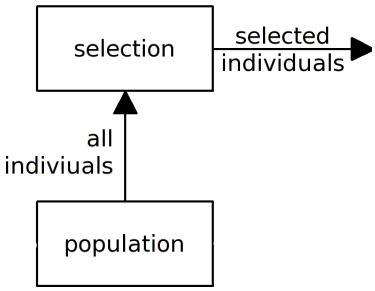


The solutions are represented in form of chromosomes, e.g.

- Bit strings
- Real numbers
- Permutations of element
- Lists of rules
- Program elements
- ... any data structure ...

Selection

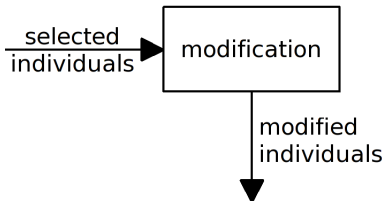
Parents to undergo modification (reproduction) are selected at random with chance biased in relation to their fitness (chromosome evaluation)



Selection methods include

- Fitness proportional selection (roulette wheel)
- Ranking
- Tournament

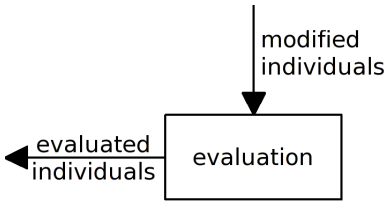
Modification



Modifications are stochastically triggered and based on the following (genetic) operators:

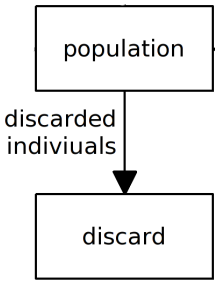
- Crossover (recombination)
- Mutation

Evaluation



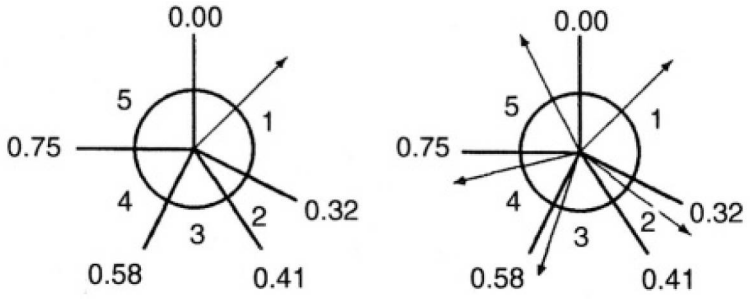
- Evaluator decodes a chromosome and assigns it a fitness measure
- It is the only link between the GA and the problem it is solving (genotype–phenotype mapping)

Deletion



- Low–fit members of the population are discarded to make space for the high–fit children while keeping the population size constant (so caled *steady–state GA*)
- In *generational GA*, entire population is replaced in each generation (the entire old population is discarded)

Selection – roulette wheel
Parents are selected randomly with bias corresponding to their fitness – similar to a biased roulette wheel with uneven sectors



Alternative to speed up the selection process: SUS (Stochastic Universal Selection) with multiple evenly–spaced pointers

Selection – alternative methods
Roulette wheel – simple and intuitive, but:

- assumes non-negative (positive or zero) fitness values
- assumes that fitness must be maximized
- requires computation of global statistic (total sum of fitness values)
- fitness–distribution related problems
  - a "super–performer" will take over population leading to premature convergence
  - most individuals with about the same fitness values – selection becomes almost random

... alternative methods are needed

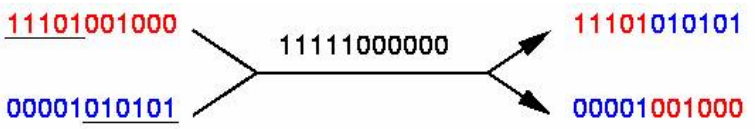
Ranking selection
Consists of two steps:

- all individuals are ranked according to their fitness and sorted (ascending or descending)
- selection is applied based on the rankings (similar to roulette wheel selection, but based only on relative performance of the individuals – absolute values of fitness are discarded)

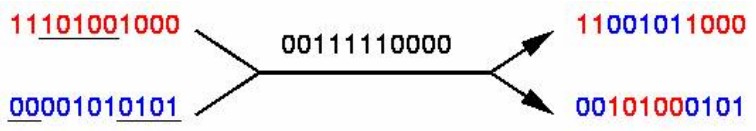
Tournament selection
Consists of two steps:

- random choice of k individuals from population (k - tournament size)
- performing tournament within the subpopulations
  - deterministic (more common)
  - fitness proportional (or ranked)

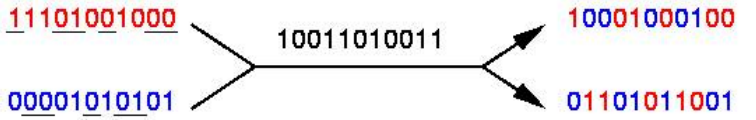
Crossover
Single point crossover



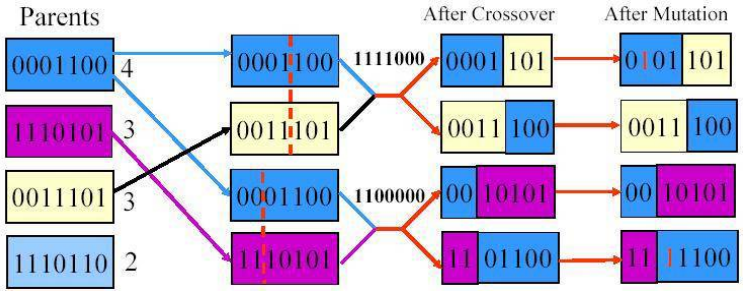
Two point crossover



Uniform crossover – applied at each locus with certain p



Example of selection and modifications



Example: Designing a can
Problem: Design a cylindrical can

- volume of at least 300ml
- minimal cost of can material

Consider 2 parameters: diameter d and height h

NLP problem:

Minimize	$f(d,h) = c\left(\frac{\pi d^2}{2} + \pi dh\right)$
subject to	$g(d,h) = \frac{\pi d^2 h}{4} \geq 300$
variable bounds	$d_{min} \leq d \leq d_{max}$, and $h_{min} \leq h \leq h_{max}$

In–class solved example ...

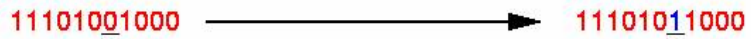
4.3.2 Representations and genetic oprators

A Representation of individuals
First and very important stage of developing GA

Right representation must be chosen for problem on hand:

- Binary representation: simplest and often used (sometimes improperly); suitable for problems with flags or Boolean decision variables; representing integers with or real numbers bitstrings is possible but generally not suitable (significance of bits; can be alleviated by Gray coding)
- Integer representation: e.g. when finding optimal values for a set of integer variables
- Real–valued (floating–point) representation: often the most sensible choice
- Permutation representations: for order–based (e.g. scheduling) or adjacency–based problems (TSP); requirement of no repetition.

B-1 Mutation (binary representations)
Typically, each bit is considered separately and is allowed to flip with certain small probability p_m



Suitable value for the bitwise mutation rate p_m depends on problem. Typically, it is chosen such that on average between one gene per generation and one gene per offspring (child) is mutated.

B-2 Mutation (integer representations)
Two principal types of mutation which mutate each gene with probability p_m

Random reseting – bit flipping of binary mutation is extended to such that a new value (from valid set of integer values) is chosen with probability p_m (suitable for cardinal attributes) 1 2 3 5 7 9 → 1 2 3 8 2

Creep mutation – adding a small value (positive or negative) to each gene with probability p_m (suitable for ordinal attributes); the magnitude of the value is drawn from a distribution around the current allele value at given position.
1 2 3 5 7 9 → 1 2 3 6 6 9

B-3 Mutation (floating–point representations)
Allele value of each gene is changed randomly within its given domain (lower and upper bound). Two types are available according to the distribution from which gene values are drawn.

Uniform mutation – with position-wise probability p_m (analogous to bit–flipping and random resetting)

Nonuniform mutation – analogous to creep mutation for integer representation; Typically, Gaussian distribution is used. Sometimes, p_m is set equal to 1 and the amount of mutation is controlled by changing the standard deviation of the Gaussian (each gene gets mutated, but only some by a significant value due to small SD).

B-4 Mutation (permutation representations)
Genes cannot be considered independently; rather, the allele values are moved around in the chromosome. The mutation parameter p_m thus has the meaning of likelihood that the string as a whole undergoes mutation. There are four major types:

Swap mutation – alleles at two randomly chosen positions are exchanged

1 2 3 4 5 6 7 8 9 → 1 5 3 4 2 6 7 8 9

Insert mutation – two alleles are randomly chosen and one is moved next to the other (others are shuffled to make the room)

1 2 3 4 5 6 7 8 9 → 1 2 5 3 4 6 7 8 9

Scramble mutation – the values of the entire string, or its randomly chosen subset, have their positions scrambled

1 2 3 4 5 6 7 8 9 → 1 3 5 4 2 6 7 8 9

Inverse mutation – two loci are randomly and the order in which the values appear between those positions are reversed (works well for adjacency based problems, because the other mutations could break a large number of links in such problems, e.g. TSP)

1 2 3 4 5 6 7 8 9 → 1 5 4 3 2 6 7 8 9

C-1 Recombination (crossover) for binary representations
Three types of recombination (mentioned earlier) are used:

- One-point crossover

$$0\,0\,0\,0\,|\,1\,0\,0\,0\,0 \rightarrow 0\,0\,0\,0\,|\,0\,0\,0\,0\,1$$
$$1\,1\,0\,1\,|\,0\,0\,0\,0\,1 \rightarrow 1\,1\,0\,1\,|\,1\,0\,0\,0\,0$$

- N–point crossover (e.g. 2–point crossover)

$$0\,0\,0\,0\,|\,0\,0\,0\,|\,0\,0\,0 \rightarrow 0\,0\,0\,0\,|\,1\,1\,1\,|\,0\,0\,0$$
$$1\,1\,1\,1\,|\,1\,1\,1\,|\,1\,1\,1 \rightarrow 1\,1\,1\,1\,|\,0\,0\,0\,|\,1\,1\,1$$

Here, the position(s) to split both parents are randomly chosen from interval $[0, l-1]$, where l is the length of the string.

Recombination (crossover) for binary representations

- Uniform crossover

Here, each gene is considered separately to be inherited from either parent (random string of length l is generated; loci with sub-threshold values are inherited from first parent, otherwise from the second parent). For example

$$0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \rightarrow 0\,1\,0\,0\,1\,1\,0\,1\,0\,0$$
$$1\,1\,1\,1\,1\,1\,1\,1\,1\,1 \rightarrow 1\,0\,1\,1\,0\,0\,1\,0\,1\,1$$

is the result of comparing random vector $\{0.35, 0.62, 0.18, 0.42, 0.83, 0.76, 0.39, 0.51, 0.36, 0.12\}$ to threshold 0.5.

C-2 Recombination for integer representations

Operators analogous to those used for binary representations are used (no blending of allele values is desirable as it could lead to non–integer values)

- One-point crossover
- N–point crossover
- Uniform crossover

C-3 Recombination for floating-point representations

There are two classes of options:

Discrete recombination: analogous to the previous cases

- One-point crossover

- N–point crossover
- Uniform crossover

Arithmetic (intermediate) recombination: blending the alleles from both parents

- Simple arithmetic recombination
- Single arithmetic recombination
- Whole arithmetic recombination

Advantage: Introduction of new genetic information (not just moving existing values around)

Arithmetic (intermediate) recombination
In each case, parameter α is either set to 0.5, or chosen randomly from $[0, 1]$; parents are $\langle x_1, \ldots, x_l \rangle$ and $\langle y_1, \ldots, y_l \rangle$

- Simple arithmetic recombination (k is randomly chosen) Child 1: $\langle x_1, \ldots, x_k, [\alpha x_{k+1} + (1-\alpha) y_{k+1}], \ldots, [\alpha x_l + (1-\alpha) y_l] \rangle$ Child 2: $\langle y_1, \ldots, y_k, [\alpha y_{k+1} + (1-\alpha) x_{k+1}], \ldots, [\alpha y_l + (1-\alpha) x_l] \rangle$
- Single arithmetic recombination Child 1: $\langle x_1, \ldots, x_{k-1}, [\alpha x_k + (1-\alpha) y_k], x_{k+1}, \ldots, x_l \rangle$ Child 2: $\langle y_1, \ldots, y_{k-1}, [\alpha y_k + (1-\alpha) x_k], y_{k+1}, \ldots, y_l \rangle$
- Whole arithmetic recombination Child 1: $x_i^{\text{new}} = \alpha x_i^{\text{old}} + (1-\alpha) y_i^{\text{old}}$ Child 2: $y_i^{\text{new}} = \alpha y_i^{\text{old}} + (1-\alpha) x_i^{\text{old}}, i = 1, 2, \ldots, k$

4.3.3 How do GAs work?

How do GAs work?
GAs work with very simple operations: string copying, substring exchange, bit (or gene) alteration – why does this work?

There is no straightforward general answer; however, for the simple genetic algorithm, there is the following theorem.

Schema theorem
Short, low order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

Schema
Schema is a similarity template describing a subset of strings with similarities at certain string positions; consider:
Alphabet $V = \{0, 1, *\}$
Sample string $*11*0**$ (length = 7)
Symbol $*$ is a wildcard (don't care, either 0 or 1)
Ex 1: $S = *110111 \rightarrow 1110111, 0110111$ (2 strings)
Ex 2: (4 strings) $S = 1*1011* \rightarrow 1010110, 1010111, 1110110, 1110111$
Ex 3: $S = 1***0** \rightarrow$ (? strings)

Schema description
Schema length: $d(S)$, the distance between the first and last specific string position

$1*1011* \rightarrow d = 6 - 1 = 5$
$*11*0** \rightarrow d =$

Schema order: $o(S) = l - m$, the number of not don't care positions

$1*1011* \rightarrow o = 7 - 2 = 5$
$*11*0** \rightarrow o =$

Schema evolution
The schema theory describes how each schema in the population evolves through time:

- Let $\Phi(S, g)$ be the number of instances representing schema S at generation g.
- What to expect of $\Phi(S, g + 1)$?.

Selection and fitness of schemata
Probability of selecting an individual i:

$$P_i(g) = \frac{f_i(g)}{\sum f_j(g)}, \text{ or } \frac{f_i(g)}{n f_{\text{ave}}(g)}$$

Average fitness value of the members of schema S can be defined in the following way

$$f_{\text{ave}}(S, g) = \frac{\sum_S f_i}{\Phi(S, g)},$$

where the sum goes through only the individuals covered by schema S.

Expectation for $g+1$
The expected value of the number of instances in schema S at generation $g+1$ is:

$$E[\Phi(S,g+1)] = \frac{f_{\text{ave}}(S,g)}{f_{\text{ave}}(g)}\Phi(S,g),$$

which means the value is **proportional** to the average fitness of those individuals in schema S at generation g and **inversely proportional** to the average fitness of all individuals at this generation.

Schema growth equation
Assume that fitness of S remains constantly above average by β, i.e.

$$E[\Phi(S,g+1)] = \frac{f_{\text{ave}}(g)+\beta f_{\text{ave}}(g)}{f_{\text{ave}}(g)}\Phi(S,g),$$

which can be simplified as

$$E[\Phi(S,g+1)] = (1+\beta)\Phi(S,g).$$

This means that **better/**worse schemata will receive exponentially **increasing/**decreasing number of trials in the subsequent generations.

Schema theory illustrated

String Processing

String No.	Initial Population (Randomly Generated)	x Value (Unsigned Integer)	f(x) x^2	pselect$_i$ $\frac{f_i}{\Sigma f}$	Expected count $\frac{f_i}{\bar{f}}$	Actual Count from Roulette Wheel
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4.0
Average			293	0.25	1.00	1.0
Max			576	0.49	1.97	2.0

Schema Processing

	schema	Before Reproduction: String Representatives	Before Reproduction: Schema Average Fitness f(H)
H_1	1 * * * *	2,4	469
H_2	* 1 0 * *	2,3	320
H_3	1 * * * 0	2	576

4.4 Genetic Programming

Automatic Programming

- Automatic generation of computer programs
- Saying **WHAT** is wanted but not **HOW** to do it
- The programs can be of the most GENERAL form:
  - Subroutines (e.g. sorting, search)
  - Planning (e.g. sequence of moves of robotic arm)
  - Strategy in games (e.g. Pacman, Quake, Civilization ...)
  - Classification (e.g. character recognition)
  - Prediction (e.g. consumption of electricity)
  - Control (e.g. vehicle steering ...)
  - Design of electronic circuits (e.g. chip layout)

Genetic Programming (GP)

- GP is a branch of EC with a goal to generate a computer program (or just function/expression) that best fits user's requirements.
- The computer program acts as a candidate solution (like a binary string in GA).
- GP is computationally expensive and its application area was limited in 1990s.
- Recently, GP has started delivering human-competitive machine intelligence thanks to exponential growth in CPU power.

Representation of programs

- Programming languages: C, Java, Prolog, machine language, LISP
- Special languages: robots, Turing machines
- Typically, data and programs are treated the same way (lists or S-expressions):
  - `(dotimes i 3 (setq v (* i i)))`
  - `(3 4 5 (to b c))`
  - **Language = functions + terminals**

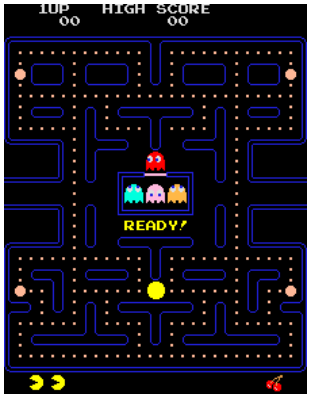
Example: Even parity

- Generate a computer program that takes 10 bits and returns whether the number of 1's is even:
  - `Even-Parity(1,0,0,0,1,0,0,1,1,0)` $\Rightarrow$ `TRUE`
- In terms of:
  - Functions: `AND, OR, NAND, NOR, NOT`
  - Terminals: `B0, B1, B2..., B9`
- Large number of input/output test cases ($2^{10} = 1024$ cases)

Test cases for 10-bit even parity (1024 cases)

B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	OUT
0	0	0	0	0	0	0	0	0	0	TRUE
0	0	0	0	0	0	0	0	0	1	FALSE
0	0	0	0	0	0	0	0	1	1	TRUE
...										
1	1	1	1	1	1	1	1	1	1	TRUE

Example: Strategy for Pacman game



- Functions:
  - `if-obstacle, if-pill, if-power-pill, if-ghost` (they are of the form if-then-else)
  - `sequence2, 3, 4` ...
- Terminals: `advance, turn-left, -right`
  - `advance, turn-left, -right`
- **GOAL**: to eat all pills within a time limit

...

Example of program in Pacman game

```
(if-ghost
    (sequence3 (turn-left)
               (turn-left)
               (advance))
(if-power-pill
    (advance)
    (turn-right)))
```

4.4.1 Genetic Programming Process

GP Terminology

- Parse tree
  - A tree structure representing a program for GP
- Function set
  - Function set is the set of operators used for the program.
  - Functions take the internal points of the parse tree.
  - An example of function set is $\{+, -, *, \%, >, IF\}$.
- Terminal set
  - The set of terminal nodes in the parse tree.
  - A terminal might be a variable, a constant, or a function taking no argument.
  - An example of terminal set is X, Y, random-constants

Preparation Steps of GP

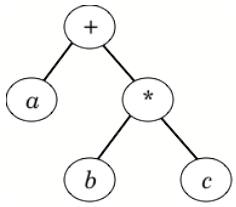
Determination of

- terminal set
- function set
- fitness measure
- parameters for GP run
- termination criterion

Program Representation

The program is generally represented by a parse tree of the function set and terminal set rather than lines of code.

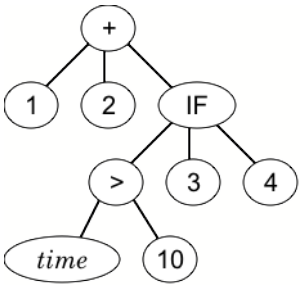
E.g., a simple expression $a + b \cdot c$ would be represented as:



As another example, consider the following simple function in C language:

```
int foo (int time)
{
    int a, b;
    if(time > 10)
        a = 3;
    else
        a = 4;
    b = a + 1 + 2;
    return b;
}
```

The function can be represented by the following tree:



GP Algorithm

1. Formation of an initial random population computer of programs, using functions and terminals
2. Execution (!) of all programs and their evaluation (fitness function)
3. Selection of well-performing programs
4. Creation of a new population by applying genetic operators to selected programs
5. Return to 2 until finding a "good" program

GP as search

- Genetic Programming performs a heuristic search in the space of computer programs (a type of "best-first" search with an opened list of limited size – the population)
- The heuristic function is the *fitness function*
- The search operators are the *genetic operators* (mutation and crossover)

Generation of an initial population

- Choose a function for the root of the tree
- Check the arity of the function
- For each argument of the function, generate:
  - a terminal; or
  - a subtree
- Practically, trees deeper that certain constant should not be generated

Methods to generate individuals

- *Full:* same depth for all branches of a tree
- *Grow:* variable depth (up to a set limit)
- *Ramped half and half:*
  - Mixture of full and grow
  - 50% will be full and 50% grow
- **Objective:** to maximize diversity

4.4.2 Functions and Terminals

Functions and Terminals

- Functions: functions or macros that take arguments
  - e.g. `(+34)`
- Terminals:
  - Functions that do not have arguments; e.g. (advance)
  - Constant: $3, a, \ldots$
  - (Ephemeral) random constant $\mathcal{R}$: for numerical problems and symbolic regression
  - Input variables: `D0,D1`, ...

Functions

- Funcions/terminals sufficient to express the solution (e.g. for boolean functions, `AND`, `OR` and `NOT` are sufficient)
- It may be useful to include powerful functions (so that the system does not have to rediscover them); e.g. `sin(x)`
- Differentiate between functions (evaluate arguments; e.g. `sum`) and macros (do not evaluate: `if-then-else`);
- Functions have to execute with any argument and without producing errors (closure); e.g. protection against division by zero
- In standard GP there are no data types. All functions must be prepared to receive any type and value of argument

4.4.3 Fitness

Evaluation of programs (fitness)

- *Raw*:
  - e.g. number of correctly guessed cases or hits (even-parity)
  - e.g. function, such as $[0.7 \times \mathsf{points} + 0.3 \times \mathsf{time}]$ (Pacman)
- *Standard*: (to be minimized)
  - Standard Fitness = Maximum - raw

- *Adjusted*: $1/(1 + \text{standard})$; it exaggerates fitness near 0
- *Normalized* (or relative):
  - ajusted / (sum of adjusted fitnesses in population)

4.4.4 GP Operations

GP Operations

- Reproduction (selection)
- Mutation
- Crossover
- Architecture-altering operations

Selection

Selection of capable (well-performing) individuals

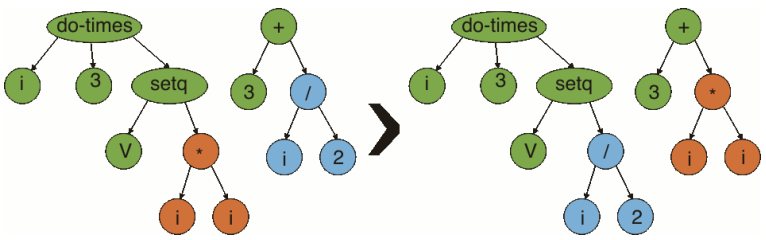
- Fitness proportional (probabilistic): uses normalized fitness
  - Problem: super-individuals
- Ranked
- Tournament
  - Match several individuals to compete among themselves: the best one reproduces
- Greedy Overselection (of the fitter individuals): create a high fitness group, H, and a low fitness group, L
  - 80% of the time select the parent from group H
  - 20% of the time select the parent from group L

Mutation Operation

- Given an individual, randomly pick a (internal or terminal) point in the parse tree.
- Delete subtree at the picked point.
- Grow new subtree at the picked point in the same way as used to generate initial individuals.
- Repair the individual if the associated program is invalid.

Crossover

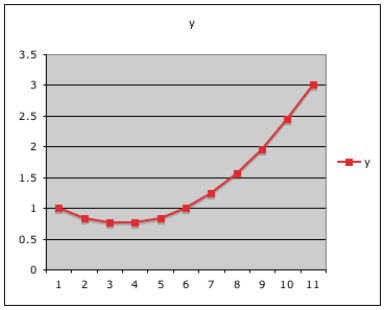
- Given two individuals, pick a point in each parse tree independently.
- Swap the subtrees at the picked points.
- Repair the parse tree if the associated program is invalid.



4.4.5 Examples

Example I: Nonlinear Function Identification

Find a nonlinear function that satisfies the following relationship:



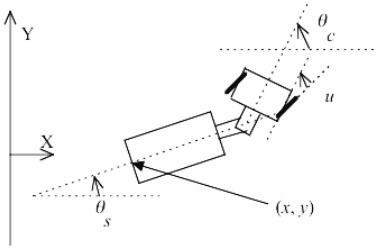
x	y
-1.00	1.00
-0.80	0.84
-0.60	0.76
-0.40	0.76
-0.20	0.84
0.00	1.00
0.20	1.24
0.40	1.56
0.60	1.96
0.80	2.44
1.00	3.00

Example I: Nonlinear Function Identification

Terminal set	$T = \{x, \text{Random-constants}\}$
Function set	$F = \{+, -, *, \%\}$
Fitness	Sum of errors between the candidate program's outputs and y's for all values of x.
Parameter	Population size: 4
Termination	When the sum of errors is less than, e.g. 0.1

Example II: Truck Backer Upper Control Law Design

Objective: Develop a control law $u = (x, y, \theta_S, \theta_C)$ that leads the trailer tail to $(x, y, \theta_S) = (0, 0, 0)$ when the backward linear velocity of the wheels is fixed.



Example II: Truck Backer Upper Control Law Design

There are four input variables of the controller: position, x, y; trailer angle, θ_S, and cap angle, θ_C

Thus, *terminals* $T = \{x, y, \theta_S, \theta_C, \text{Random-constants}\}$

Function set: arithmetic and trigonometric functions

To *evaluate* a solution candidate during evolution process, we need to conduct simulation runs over many initial conditions of the trailer for hundreds of time steps.

4.5 Swarm Intelligence

Motivation: Treasure Hunt

- **Who?** You and a group of friends
- **What do you know?** Knowledge of the approximate area of the treasure, but not exactly where it is located
- **What do you have?**
  - Metal detector
  - Strength and position of signal of your neighbors' metal detectors
  - Your friends!
- **What is your goal?** To find the treasure, or at least part of it

Motivation: Treasure Hunting (cont)

The agreement: You have agreed on some sharing mechanism:

- all who have taken part in the search will be rewarded, but with the person who found the treasure getting a higher reward than all others
- the rest are rewarded based on distance from the treasure at the time when the first one finds the treasure

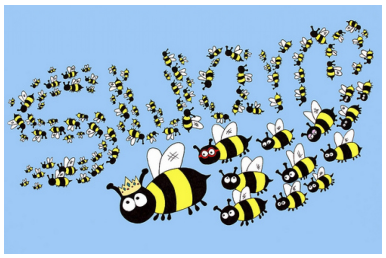
What will be your actions?

- Ignore your friends?
- Use the information from your neighboring friends?

Swarm Intelligence
What is a Swarm?

- A loosely structured collection of interacting agents
- Agents:
  - Individuals that belong to a group (but are not necessarily identical)
  - They contribute to and benefit from the group
  - They can recognize, communicate, and/or interact with each other
- A swarm is better understood if thought of as agents exhibiting a collective behavior

Examples of Swarms in Nature
Classic Example: Swarm of Bees



Can be extended to other similar systems:

- Ant colony [agents: ants]
- Flock of birds [agents: birds]
- Crowd [agents: humans]
- Immune system [agents: cells and molecules]

Swarm Intelligence (SI)

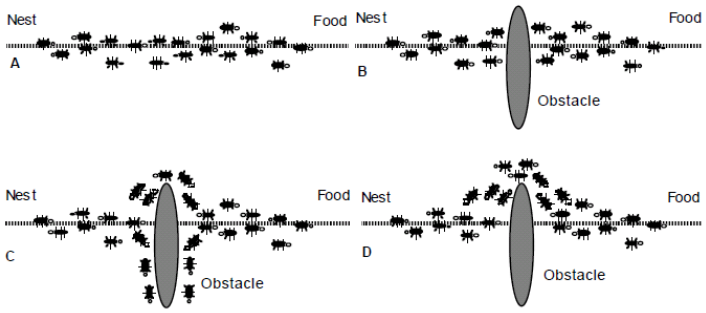
- Computational intelligence (CI) technique based on the collective behavior in decentralized, self-organized systems
- Generally made up of agents who interact with each other and the environment
- No centralized control structures
- No need to have very complex and superior intelligent agents
- Based on group behavior found in nature

4.5.1 Ant Colony Optimization

Ant Colonies in Nature

- Lack of central control
- Intelligent behavior is emergent from the collective actions of simple agents
- Ants communicate via altering the environment (deposit pheromone)
- Ants are more likely to follow routes with more pheromone

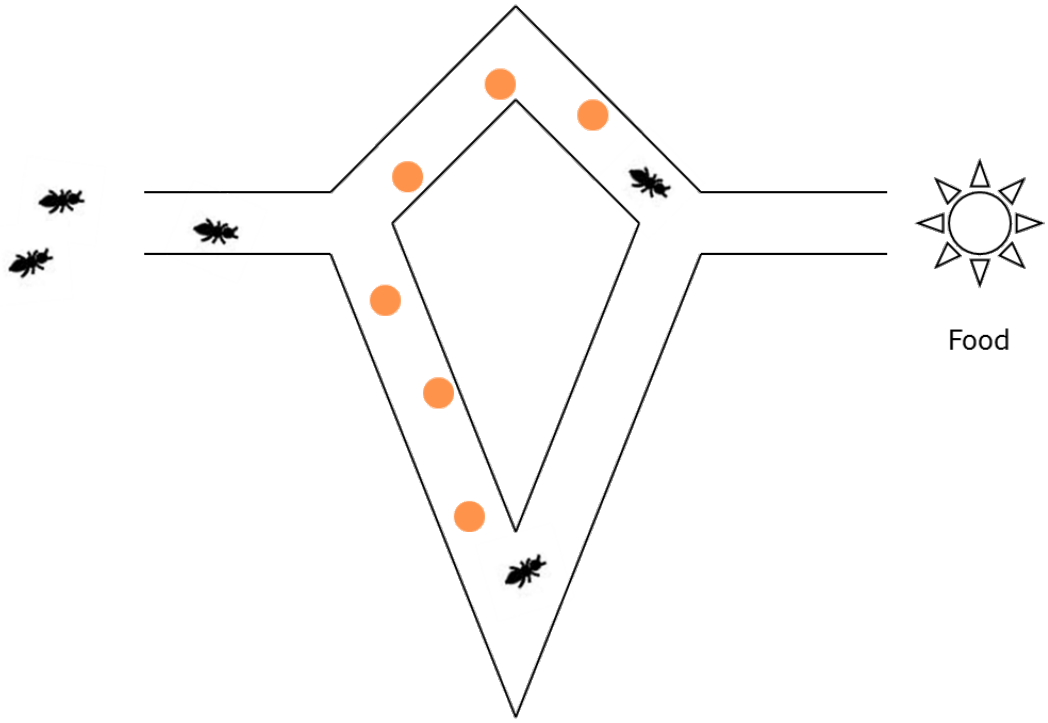
Ant Colonies - Foraging



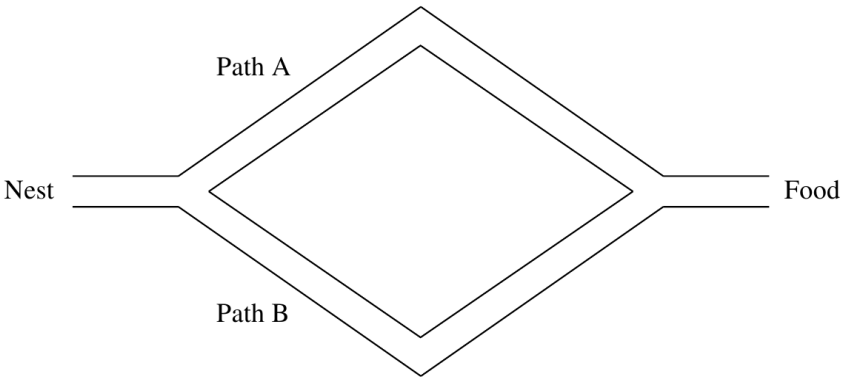
Foraging Strategies

- Ants deposit pheromone on the paths that they cover and this results in the building of a solution (optimal path).
- A way of knowledge sharing and communication through the environment (stigmergy)
- In implementation concept of pheromone evaporation is used – avoiding local optima.

Pheromone



Foraging: Bridge experiment



Probability of the next ant to choose path A:

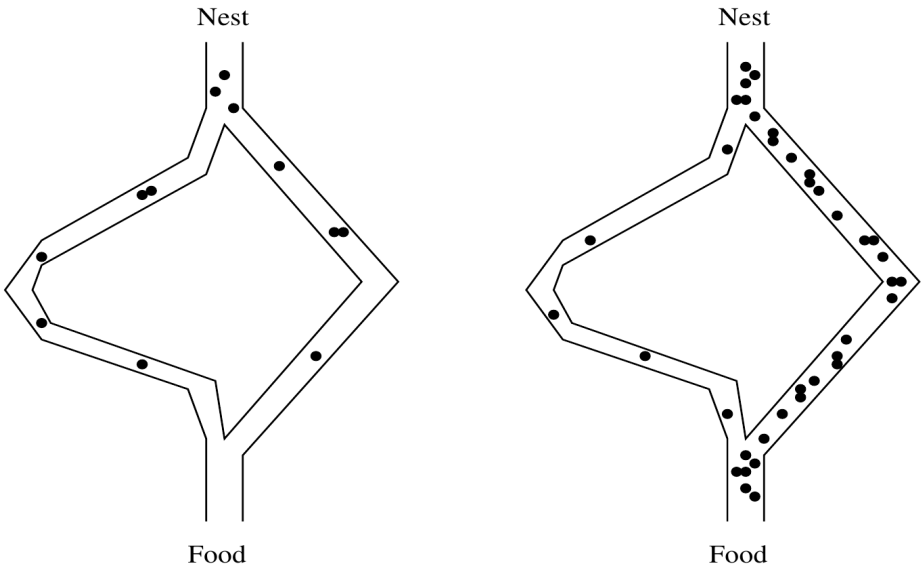
$$P_A(t+1) = \frac{(c+n_A(t))^\alpha}{(c+n_A(t))^\alpha + (c+n_B(t))^\alpha} = 1 - P_B(t+1)$$

- $n_A(t)$ and $n_B(t)$ are the number of ants on paths A and B respectively at time step t
- c quantifies the degree of attraction of an unexplored branch
- α biases towards pheromone deposits in the decision process

Foraging: Bridge experiment

- The larger the value of α, the higher the probability that the next ant will follow the path with a higher pheromone concentration
- The larger the value of c, the more pheromone deposits are required to make the choice of path non-random
- The decision rule: if $U(0,1) \leq P_A(t+1)$ then follow path A otherwise follow path B

Foraging: Extended binary bridge



Differential path length effect:

- The probability of selecting the shorter path increases with the length ratio between the two paths

Foraging Behavior of Ants (cont)

Each ant is a stimulus-response agent, following simple production rules :

1: Let $r \sim U(0,1)$
2: **for** each potential path A **do**
3: Calculate P_A;
4: **if** $r \leq P_A$ **then**
5: Follow path A;
6: Break;
7: **end if**
8: **end for**

Stigmergy and Artificial Pheromone

- Generally stated, stigmergy is a class of mechanisms that mediate animal-to-animal interactions
- A form of indirect communication mediated by modifications to the environment
- Forms of stigmergy:

- *Sematectonic stigmergy* refers to communication via changes in the physical characteristics of the environment - nest building, brood sorting
- *Sign-based stigmergy* facilitates communication via a signaling mechanism, implemented via chemical compounds deposited by ants - foraging

Artificial Stigmergy

Artificial Stigmergy
The indirect communication mediated by numeric modifications of environmental states which are only locally accessible by the communicating agents

- The essence of modeling ant behavior is to find a mathematical model that accurately describes the stigmergic characteristics of the corresponding ant individuals
- Define stimergic variables which encapsulate the information used by artificial ants to communicate indirectly:
  - for foraging behavior, artificial pheromone

Ant Algorithms

- Ant algorithms are population-based systems inspired by observations of real ant colonies
- Cooperation among individuals in an ant algorithm is achieved by exploiting the stigmergic communication mechanisms observed in real ant colonies
- Foraging-based ant algorithms are generally referred to as

Ant Colony Optimization Meta-Heuristics

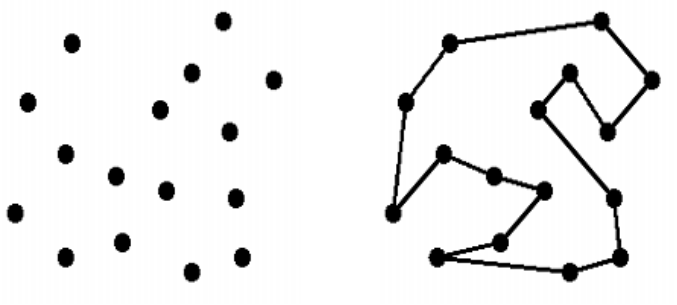
Ant Colony Optimization (ACO)

- Introduced in 1992 by Marco Dorigo
- Population-based meta-heuristic to find approximate solutions to difficult search and optimization problems
- A set of software agents
- Stochastic
- Incrementally build solutions by moving on a graph
- Constraints of the problem are built into the search process of the ants

Traveling Salesman Problem (TSP)

- **Informally:** The problem of finding the shortest tour through a set of cities starting at some city and going through all other cities once and only once, returning to the starting city
- **Formally:** Finding the shortest Hamiltonian path in a fully-connected graph
- TSP belongs to the class of NP-complete combinatorial optimization problems. Thus, it is assumed that there is no efficient algorithm for solving TSPs.

Simple TSP Instance



- Every edge has a distance and the answer would be the shortest tour having the minimum sum of the edges in the tour
- The direct solution will have a running time of $O(n!)$

16 cities: 16! = 209,227,898,88,000

Computing a solution

Approximate methods - iterative methods that try to improve the search over the course of a specific number of steps

- **Exploration:** Making choices contrary to previous experience in order to explore new tours
- **Exploitation:** Making choices considering previous experience in order to find tours close to the ones we already have

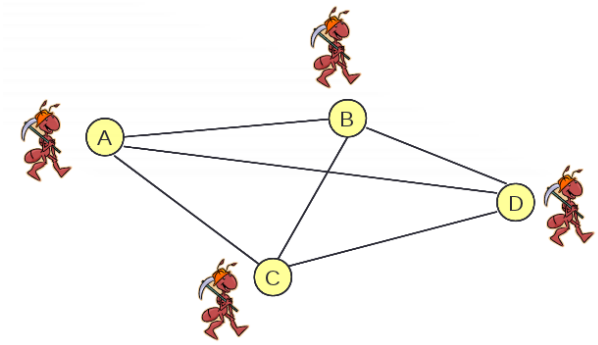
Ant System (AS)

Ant Systems for TSP

Graph (N,E): where N = cities/nodes, E = edges

d_{ij} = the tour cost from city i to city j (edge weight)

Ants "Independently" move from one city i to the next j with certain transition probability. Let the number of ants be m and cities n.



Transition from city i to j depends on

- Pheromone intensity $\tau_{ij}(t)$ for each edge – represents the learned desirability to visit city j when in city i
- Heuristic information (visibility) $\eta_{ij} = 1/d_{ij}$ represents local information – heuristic desirability to visit city j when in city i.

Generally, have several ants searching the solution space, e.g.

$$(m = n)$$

- Transition Rule
- Probability of ant k going from city i to j
- α and β are adjustable parameters tuning the balance between exploration and exploitation

$$p_{ij}^k(t) = \begin{cases} \dfrac{\tau_{ij}^\alpha(t)\eta_{ij}^\beta(t)}{\sum_{u\in\mathcal{N}_i^k(t)}\tau_{iu}^\alpha(t)\eta_{iu}^\beta(t)} & \text{if } j \in \mathcal{N}_i^k \\ 0 & \text{if } j \notin \mathcal{N}_i^k(t) \end{cases}$$

State Transition Rule: example

$$\begin{aligned} \tau(A,B) &= 150;\ \ \eta(A,B) = 1/10 \\ \tau(A,C) &= 35;\ \ \eta(A,C) = 1/7 \\ \tau(A,D) &= 90;\ \ \eta(A,D) = 1/15 \end{aligned}$$

$$p_{ij}^k(t) = \frac{\tau_{ij}(t)^\alpha\eta_{ij}^\beta}{\sum_{u\in\mathcal{N}_i^k(t)}\tau_{iu}^\alpha(t)\eta_{iu}^\beta(t)}; p_{AB}^k(t) = \frac{15}{15+5+6}$$

i.e. $p_{AB} = 15/26, p_{AC} = 5/26, p_{AD} = 6/26$

- A balance between pheromone intensity, τ_{ij} , and heuristic information, η_{ij}
- If $\alpha = 0$:
  - no pheromone information is used, i.e. previous search experience is neglected
  - the search then degrades to a stochastic greedy search
- If $\beta = 0$:
  - the attractiveness of moves is neglected
  - the search algorithm is similar to SACO
- Heuristic information adds an explicit bias towards the most attractive solutions, e.g.

$$\eta_{ij} = \tfrac{1}{d_{ij}}$$

Pheromone update:

$$\Delta\tau_{ij}^k = Q/d_{ij}^k(t) \ \text{ if } \ (i,j) \in T^k(t) \ \text{ else } \ 0$$

where T is the tour done at time t by ant k, d is the deistance, Q is a heuristic parameter (positive constant).
Pheromone decay:

$$\tau_{ij}(t) = (1-\rho)\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^k(t)$$

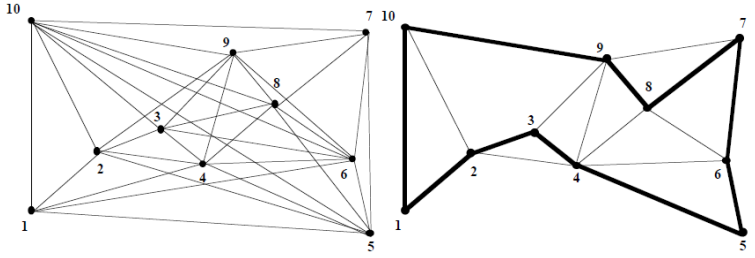
where $\rho = (0,1]$ is the evaporation rate, m is the number of ants and $\Delta\tau_{ij}^k$ is the quantity of pheromone laid on edge (i,j) by ant k.

ACO for TSP

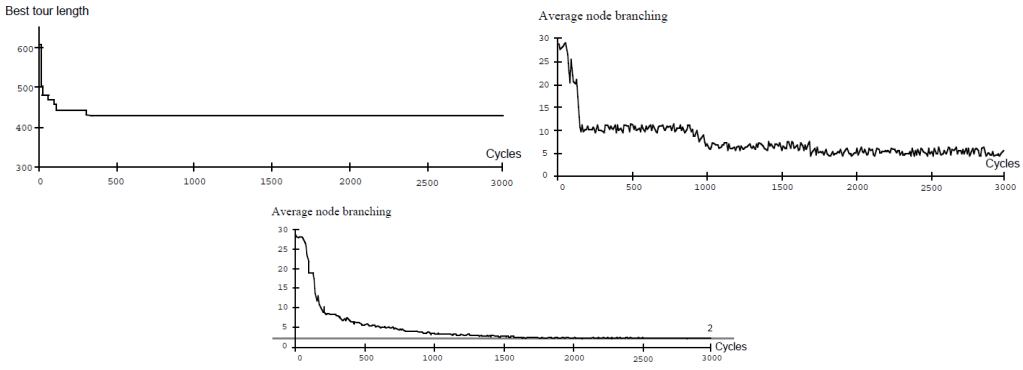
```
Loop
   Randomly position m artificial ants on n cities
   For city=1 to n
      For ant=1 to m
        {Each ant builds a solution by adding one city after
        the other}
         Select probabilistically the next city according to
                    exploration and exploitation mechanism
         Apply the local trail updating rule
      End for
   End for
   Apply the global trail updating rule using the best ant
Until End_condition
```

Stopping Criteria

- Stagnation
- Max Iterations
- Desired objective (lower than a threshold)



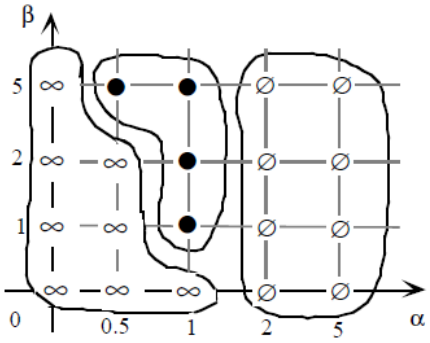
Search Process



Role of Parameters

- Experimentally tuned
- Bad solutions and stagnation-(Φ)

- Bad solutions and no stagnation- (∞)
- Good solutions ($\bullet$)



ACO Applications

- TSP
- Quadratic Assignment Problems (QAP)
- Scheduling
- Vehicle Routing Problems (VRP)
- Telecommunication Networks
- Graph Coloring
- Data Mining (classification)
- etc.

4.5.2 Particle Swarm Optimization

Particle Swarm Optimization

- Introduction
- Overview of the basic PSO
- Global Best PSO
- Local Best PSO
- Aspects of Basic PSO
- Basic Variations of PSO
- PSO Parameters
- Particle Trajectories
- Single-Solution Particle Swarm Optimizers

Introduction

Introduction

Particle swarm optimization (PSO):

- developed by Kennedy & Eberhart,

- first published in 1995,
- exponential increase in the number of publications since then.

What is PSO?

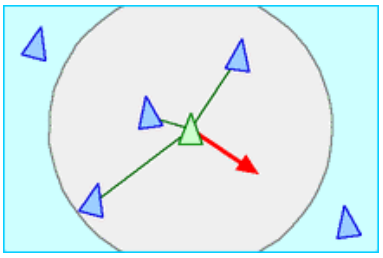
- a simple, computationally efficient optimization method
- population-based, stochastic search
- based on a social-psychological model of social influence and social learning
- individuals follow a very simple behavior: emulate the success of neighboring individuals
- emergent behavior: discovery of optimal regions in high dimensional search spaces

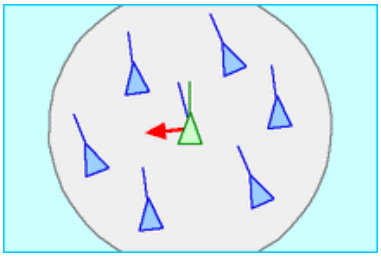
Introduction: What are the origins of PSO?

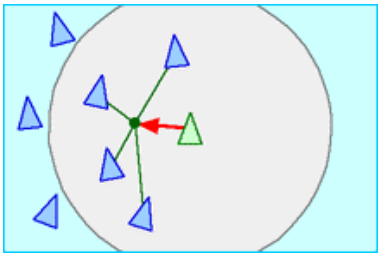
- In the work of Reynolds on "boids": Flocking is an emergent behavior which arises from the interaction of simple rules:
  - Collision avoidance
  - Velocity matching
  - Flock centering
- The work of Heppner and Grenander on using a "roost" as an attractor for a bird flock
- Simplified social model of determining nearest neighbors and velocity matching

Origins of PSO (cont)

Boids: flocks, herds, and schools







Separation: steer to avoid crowding local flockmates

Alignment: steer towards the average heading of local flockmates

Cohesion: steer to move toward the average position of local flockmates

(http://www.red3d.com/cwr/boids/ ©Craig Reynolds)

Origins of PSO (cont)

- Initial objective: to simulate the graceful, unpredictable choreography of collision-proof birds in a flock
- At each iteration, each individual determines its nearest neighbor and replaces its velocity with that of its neighbor
- Resulted in synchronous movement of the flock
- Random adjustments to velocities prevented individuals to settle too quickly on an unchanging direction
- Adding roosts as attractors:
  - personal best
  - neighborhood best $\rightarrow$ particle swarm optimization

Overview of basic PSO

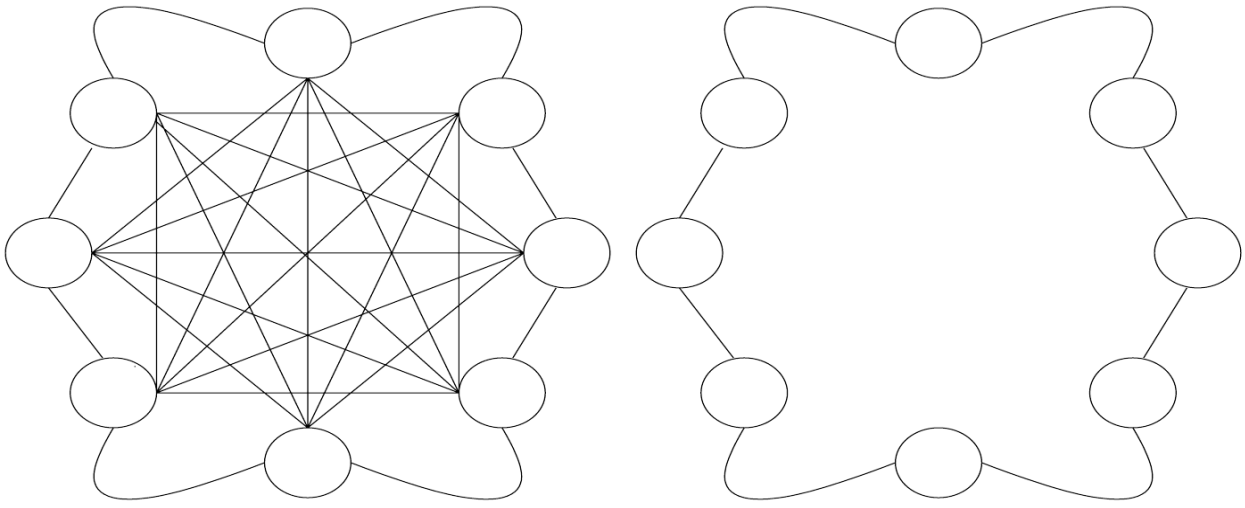
What are the main components?

- A swarm of particles
- Each particle represents a candidate solution
- Elements of a particle represent parameters to be optimized

The search process:

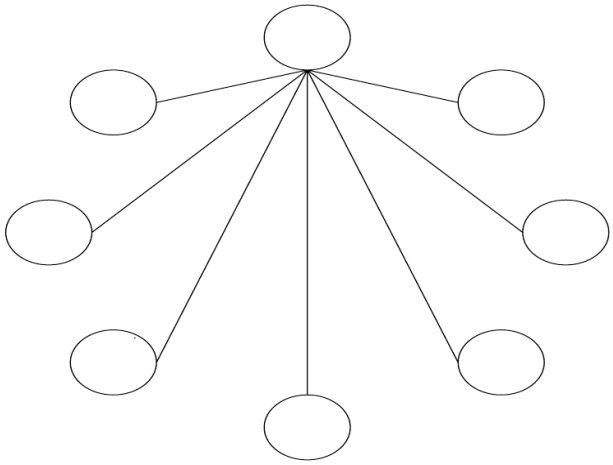
- Position updates $$x_i(t+1) = x_i(t) + v_i(t+1)$$ where $$x_{ij}(0) \sim U(x_j^{\min}, x_j^{\max})$$
- Velocity
  - drives the optimization process
  - step size
  - reflects experiential knowledge and socially exchanged information

Social interaction based on neighborhoods - topologies

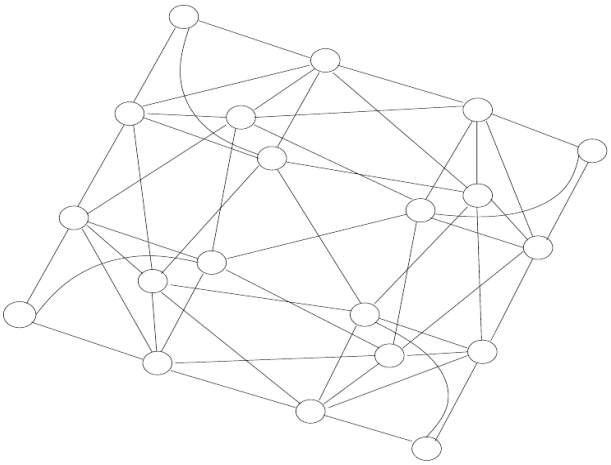


Star Ring

Social interaction based on neighborhoods - topologies

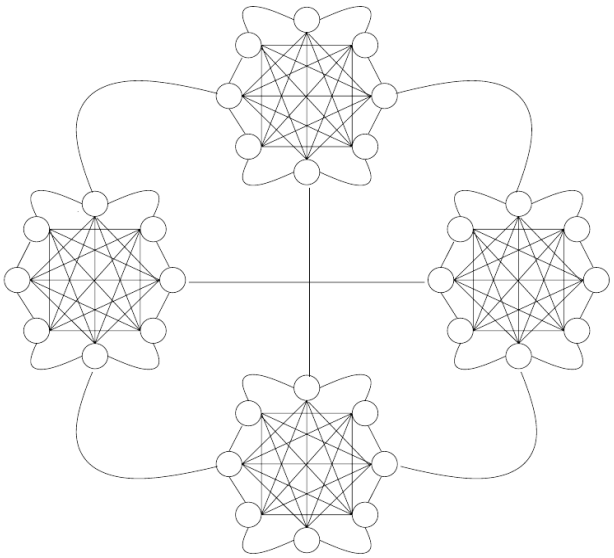


Wheel

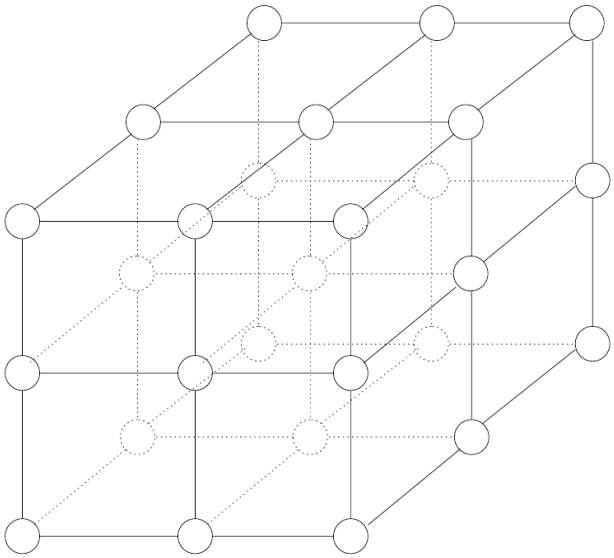


Pyramid (A. P. Engelbrecht, Computational Intelligence, ©2007 Wiley)

Social interaction based on neighborhoods - topologies



Clusters



Von Neumann (A. P. Engelbrecht, Computational Intelligence, ©2007 Wiley)

Global Best PSO

Global Best (`gbest`) PSO

Uses the **star** social network

Velocity update per dimension

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

$v_{ij}(0) = 0$ (usually, but can be random) c_1, c_2 are positive acceleration coefficients $r_{1j}(t), r_{2j}(t) \sim U(0,1)$

Global Best PSO (cont)

$\boldsymbol{y}_i$ (t) is the **personal best** position (for minimization):

$$\boldsymbol{y}_i(t+1) = \begin{cases} \boldsymbol{y}_i(t) & \text{if } f\big(\boldsymbol{x}_i(t+1)\big) \geq f\big(\boldsymbol{y}_i(t)\big) \\ \boldsymbol{x}_i(t+1) & \text{if } f\big(\boldsymbol{x}_i(t+1)\big) < f\big(\boldsymbol{y}_i(t)\big) \end{cases}$$

$\hat{\boldsymbol{y}}(t)$ is the **global best** position calculated as

$$\hat{\boldsymbol{y}}(t) \in \{\boldsymbol{y}_0(t), ..., \boldsymbol{y}_{n_s}(t)\} | f(\hat{\boldsymbol{y}}(t)\big) = \min f\big(\boldsymbol{y}_0(t)\big), ..., f\big(\boldsymbol{y}_{n_s}(t)\big)$$

or

$$\hat{\boldsymbol{y}}(t) \in \{\boldsymbol{x}_0(t), ..., \boldsymbol{x}_{n_s}(t)\} | f(\hat{\boldsymbol{y}}(t)\big) = \min f\big(\boldsymbol{x}_0(t)\big), ..., f\big(\boldsymbol{x}_{n_s}(t)\big)$$

where n_s is the number of particles in the swarm.

`gbest` **PSO Algorithm**

1: Create and initialize an n_x-dimensional swarm S;
2: **while** stopping condition is not true **do**
3: **for** each particle $i = \ldots, n_s$ **do**
4: **if** $f(\boldsymbol{x}_i) < f(\boldsymbol{y}_i)$ **then**
5: $\boldsymbol{y}_i = \boldsymbol{x}_i$;
6: **end if**
7: **if** $f(\boldsymbol{y}_i) < f(\hat{\boldsymbol{y}})$ **then**
8: $\hat{\boldsymbol{y}} = \boldsymbol{y}_i$;
9: **end if**
10: **end for**
11: **for** each particle $i = 1, \ldots, n_s$ **do**
12: update the velocity and then the position;
13: **end for**
14: **end while**

Local Best PSO

Local Best (`lbest`) PSO

Uses the **ring** social network

Velocity update per dimension (same as `gbest`)

$$\begin{aligned} v_{ij}(t+1) = v_{ij}(t) &+ c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] \\ &+ c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \end{aligned}$$

but $\hat{\boldsymbol{y}}_i$ is now the **neighborhood best**, defined as

$$\hat{\boldsymbol{y}}_i(t+1) \in \{\mathcal{N}_i | f(\hat{\boldsymbol{y}}_i(t+1)) = \min\{f(\boldsymbol{x})\}, \forall \boldsymbol{x} \in \mathcal{N}_i\}$$

with the neighborhood defined as

$$\begin{aligned} \mathcal{N}_i = &\{\boldsymbol{y}_{i-n_{\mathcal{N}_i}}(t), \boldsymbol{y}_{i-n_{\mathcal{N}_i}+1}(t), \ldots, \\ &\boldsymbol{y}_{i-1}(t), \boldsymbol{y}_i(t), \boldsymbol{y}_{i+1}(t), \ldots, \boldsymbol{y}_{i+n_{\mathcal{N}_i}}(t)\} \end{aligned}$$

where $n_{\mathcal{N}_i}$ is the neighborhood size

Local Best PSO (cont...)

Neighborhoods:

- Neighborhoods are based on particle indices, not spatial information
- Spatial approach is possible, but computationally expensive and not that effective in spreading information (indices allow leaps in search space)
- Neighborhoods overlap to facilitate information exchange

`gbest` PSO vs `lbest` PSO:

- Speed of convergence: larger interconnectivity of `gbest` PSO - it converges faster than the `lbest` PSO (but has less diversity)
- Susceptibility to local minima: larger diversity of `lbest` PSO (i.e. it covers larger parts of the search space) makes it less susceptible to being trapped

Velocity Components

Previous velocity, $\boldsymbol{v}_i(t)$

- inertia component
- memory of previous flight direction
- prevents particle from drastically changing direction

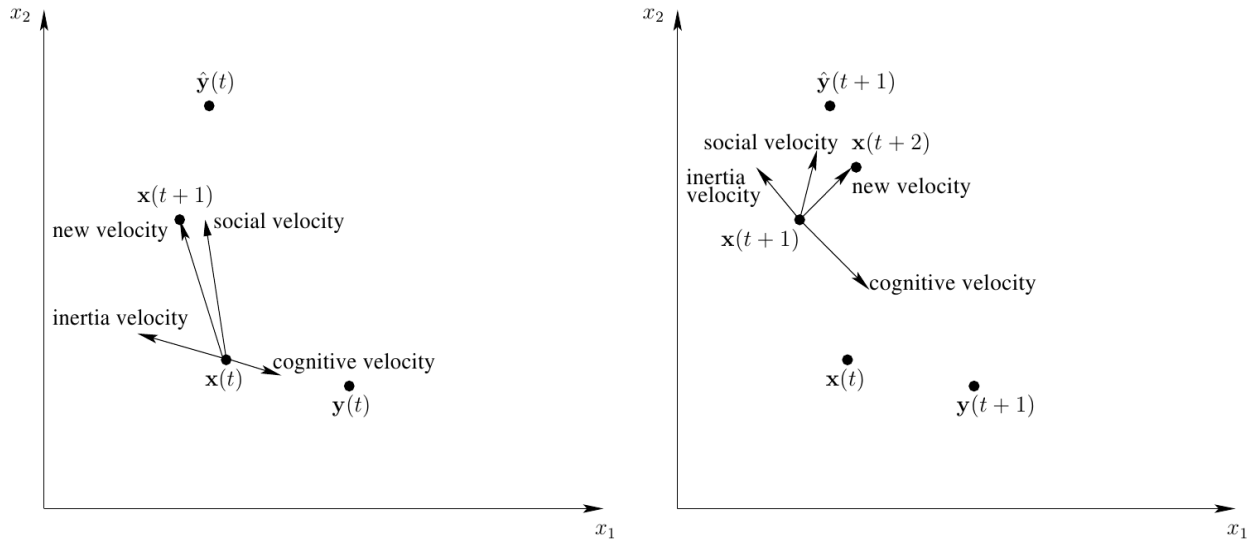
Cognitive component, $c_1\boldsymbol{r}_1(\boldsymbol{y}_i - \boldsymbol{x}_i)$

- quantifies performance relative to past performances
- memory of previous best position
- a.k.a. nostalgia

Social component, $c_2\boldsymbol{r}_2(\hat{\boldsymbol{y}}_i - \boldsymbol{x}_i)$

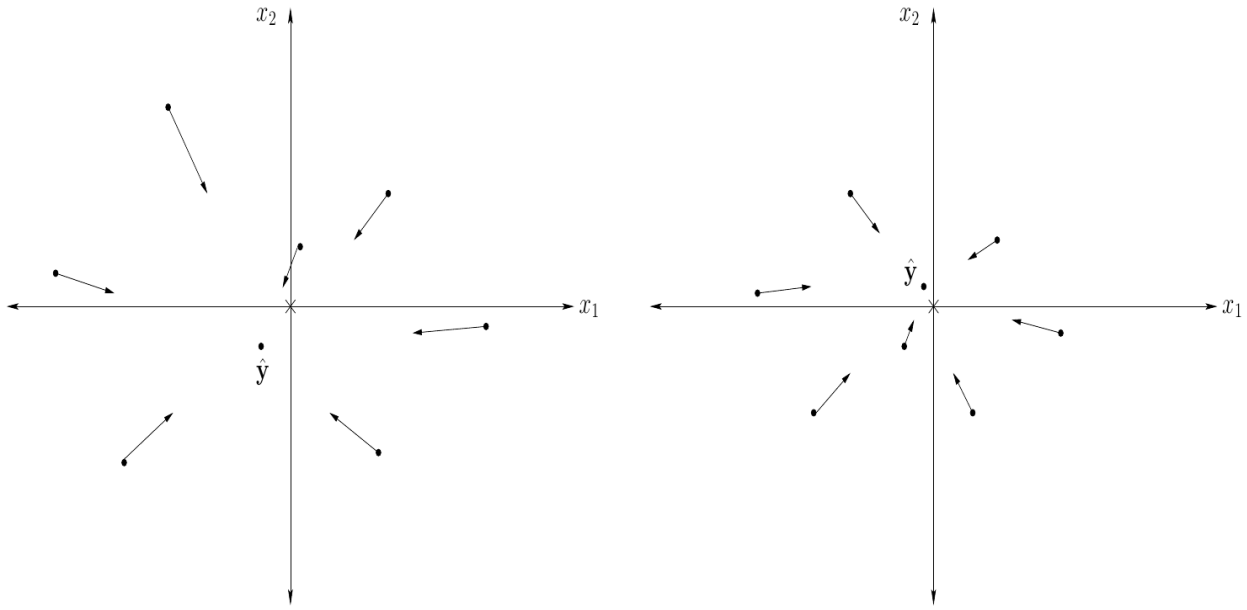
- quantifies performance relative to neighbors
- a.k.a. envy

Geometric Illustration for a Single Two-Dimensional Particle



Time step t Time step $t+1$ (A. P. Engelbrecht, Computational Intelligence, ©2007 Wiley)

Cumulative Effect of Position Updates (multi-particle `gbest` PSO)



Time step t Time step $t+1$ (A. P. Engelbrecht, Computational Intelligence, ©2007 Wiley)

Stopping Conditions

- Terminate when a maximum number of iterations, or function evaluations (FEs), has been exceeded
- Terminate when an acceptable solution has been found, i.e. when (assuming minimization)

$$f(\boldsymbol{x}_i) \leq |f(\boldsymbol{x}^*) - \epsilon|$$

- Terminate when no improvement is observed over a number of iterations

Stopping Conditions (cont)

- Terminate when the normalized swarm radius is close to zero, i.e.

$$R_{\mathsf{norm}} = \frac{R_{\mathsf{max}}}{\mathsf{diameter}\big(S(0)\big)},$$

where
$$R_{\mathsf{max}} = ||\boldsymbol{x}_m - \hat{\boldsymbol{y}}||, m = 1, \ldots, n_s,$$

with
$$||\boldsymbol{x}_m - \hat{\boldsymbol{y}}|| \geq ||\boldsymbol{x}_i - \hat{\boldsymbol{y}}||, \forall i = 1, \ldots, n_s.$$

- Terminate when the objective function slope is approximately zero, i.e.

$$f^{'}(t) = \frac{f(\hat{\boldsymbol{y}}(t)) - f(\hat{\boldsymbol{y}}(t-1))}{f(\hat{\boldsymbol{y}}(t))} \approx 0$$

Chapter 5

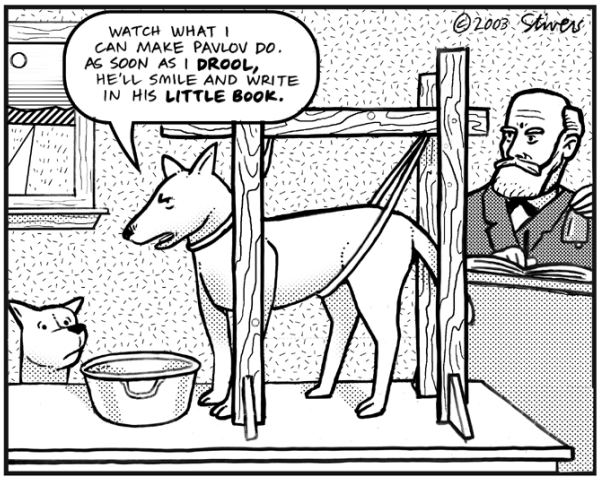
Hybrid Systems

5.1 Neural Networks and Fuzzy Systems

Neural Networks
Problem requirements:

- There must be sample data describing the problem
- If input-output data is available → Supervised Learning
- If input data only is available → Unsupervised Learning
- If input data and error measurement is available → Reinforcement Learning

Reinforcement Learning
We did not really talk about reinforcement learning (RL); here is a simple illustrative example:



Neural Networks
Pros and Cons:

+ No prior knowledge is required.
– We can't interpret the solution, NN is a black-box.
– We can't initialize NN with prior knowledge. (learning from scratch)
+ Fault tolerant to changes in input and network structure.

– If problem data differs too much from training data → NN can't cope → retraining.
– No guarantee for convergence → learn from scratch.
+ No need for mathematical (explicit, analytical) model.

Fuzzy Systems
Problem requirements:

- Prior knowledge about the problem in terms of IF-THEN rules.
- Defining suitable fuzzy sets describing the linguistic variables.

Fuzzy Systems
Pros and Cons

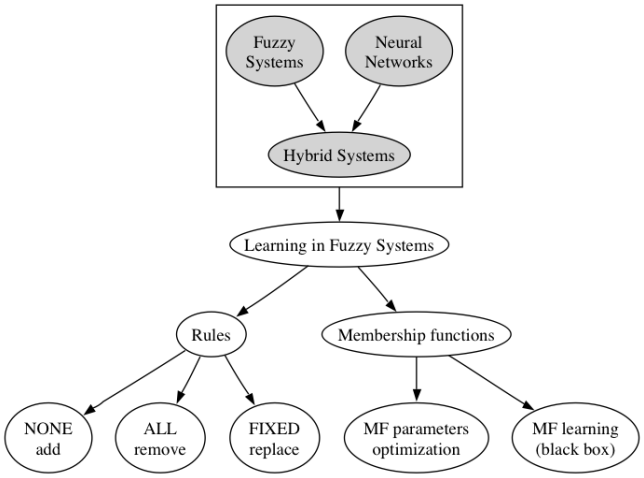
+ No need for mathematical model.
+ No need for training data.
– Definition of fuzzy rules and fuzzy sets requires tuning.
+ Fault tolerant to small changes in input and system parameters.
– Slight changes in MF's can cause changes in performance

5.2 Hybrid Systems

Combining both approaches - Hybrid Systems

- Manual adaptation of FS parameters is not easy.
- Automatic adaptation could be done by taking advantage of NN learning capabilities.
- Can be used to learn new fuzzy rules and MF's, or to optimize existing ones.

Hybrid Systems: Approaches to Learning



5.3 Learning in Fuzzy Systems

Learning Fuzzy Rules (1/3)
First approach

- Start without rules and create new ones through learning.
- Create a new rule if a training pattern is not covered by existing rules.
- Drawbacks:
  - Can lead to large rule base (especially if MF's don't verlap.)
  - Can lead to inconsistent rule base $\rightarrow$ delete some rules.

Learning Fuzzy Rules (2/3)
Second approach

- Start with all possible rules.
- Delete unnecessary rules.
- Drawbacks:
  - Performance evaluation is needed for individual rules.
  - Complex with large number of variables.
  - It's possible to end with too few rules.

Learning Fuzzy Rules (3/3)
Third approach

- Start with a fixed number of rules (possibly random.)
- Replace rules during learning, checking for consistency at each step.
- Drawbacks:
  - Fixed number of rules.
  - Needs evaluation scheme for rule deletion.
  - Needs analysis procedure for acquiring new rules.

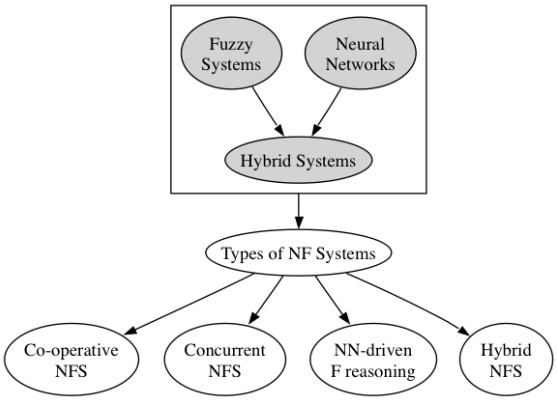
Learning Membership Functions (1/2)

- Less complex than adaptation of rule bases.
- MF's parameters can be optimized with respect to a global error.
- NN learning is used to meet certain conditions (e.g. MF's must overlap).

Learning Membership Functions (2/2)
Two approaches:

(i) MF's are described by parameters which are optimized in a learning process.
(ii) NN learns to produce membership values for given inputs by using sample data. (disadvantage: MF's are not explicitly known – black box)

Hybrid Systems: Classification



5.4 Neuro-Fuzzy Systems

Neuro-Fuzzy Systems

- Basic idea is to find fuzzy system parameters by means of learning using a neural network.
- Common way:
  - Represent the FS in a NN architecture.
  - Apply a learning algorithm (e.g. back-propagation)
  - Problem: NN learning usually depends on gradient descent method, and FS inference process is not differentiable (e.g. min, max)
  - Possible solution:
    a) Replace the FS functions with differentiable ones.
    b) Use better suited procedure other than gradient descent for learning.

Common Models

- ANFIS:
  - Implements a Sugeno-like FS in a NN architecture.
  - Uses back-propagation and LMS for learning.
  - Uses only differentiable functions [solution (a)]
- GARIC:
  - Uses special "soft-min" function which is differentiable [solution (a)]
- NEFCON, NEFCLASS, and NEFPROX:
  - Use Mamdani-type FS.
  - Use special learning algorithm [solution (b)]

Common Properties of Neuro-Fuzzy Systems

- Learning
  - Trained by learning algorithm derived from NN theory.
  - Operates on local info and causes local modifications.
  - Data-driven, not knowledge based.

- Interpretation
  - Can be interpreted as a system of fuzzy rules.
  - Can be created from scratch (using learning data) or initialized with prior knowledge (fuzzy rules).
- Semantics
  - Learning takes the semantics of a FS into account (which results in constraints on modifying system parameters.)

Common Properties of Neuro-Fuzzy Systems (continued)

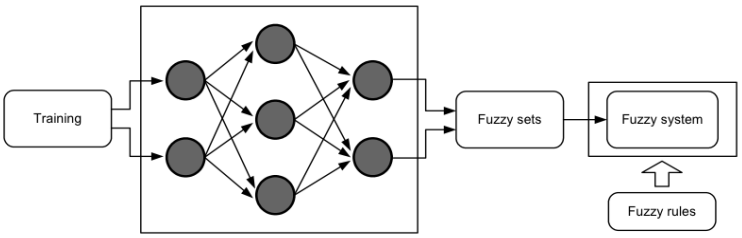
- Function approximation
  - NF system approximates an n-dimensional (unknown) function that is partially given by the training data.
  - The fuzzy rules represent vague samples (or vague prototype of training data.)
- NN-like
  - NF system can be viewed as special 3-layer feedforward network whose neurons are t-norms and t-conorms instead of activation functions.
    - 1st layer: Inputs
    - 2nd layer: Rules
    - 3rd layer: Outputs
    - Connection weights: Fuzzy sets

Types of Neuro-Fuzzy Systems

- Co-operative NF systems
  - NN and FS work independently of each other.
  - NN is used to determine certain parameters of the FS.
- Hybrid NF systems
  - FS is implemented (or interpreted) as a special kind of NN.
  - NN learning algorithm is used to tune the system.

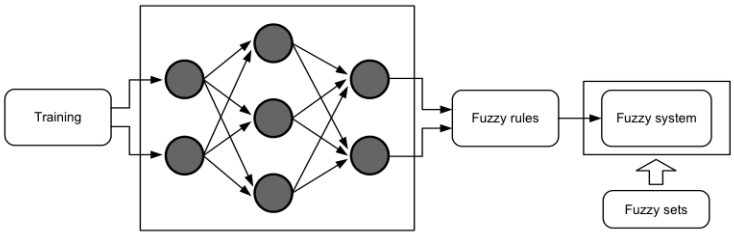
Co-operative NF systems (1/4)

- A NN determines MF's from learning data, either by tuning MF parameters or approximating the MF.
- The tuned fuzzy sets are then used with separately given fuzzy rules.



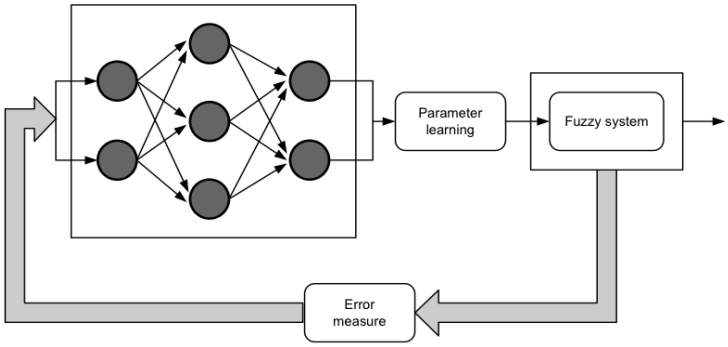
Co-operative NF systems (2/4)

- A NN determines fuzzy rules from training data (usually by clustering).
- MF's are specified separately and then used with the tuned rules.



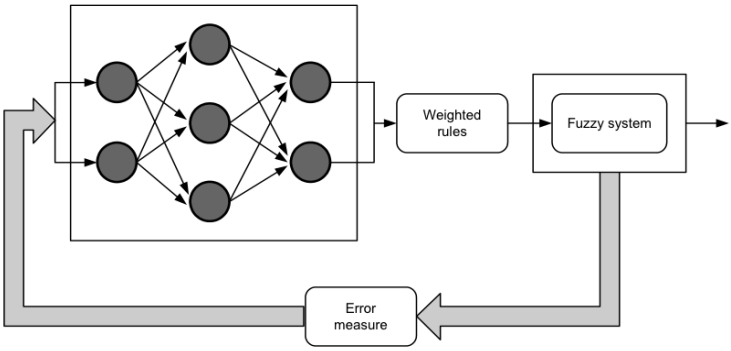
Co-operative NF systems (3/4)

- The system learns parameters online during use of the FS to adapt MF's.
- Fuzzy rules and initial MF's must be given.
- An error measure must be specified that guides learning.



Co-operative NF systems (4/4)

- A NN determines rule weights either online or offline.
- Rule weights are multiplied by rule outputs.
- Often destroys the semantics of rules.

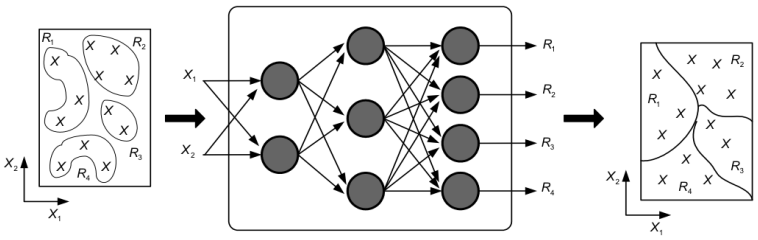


Concurrent neural/fuzzy systems

- Use a NN as a preprocessor or postprocessor of a Fuzzy system.
- These types do not optimize the FS, but improve the overall performance of the combined system.
- Useful if the inputs to a FS have to be combined, or the output is to be combined with other parameters first; i.e. inputs or outputs can not be processed directly.
- Sometimes not considered as neuro-fuzzy systems

Neural Network-driven Fuzzy Reasoning

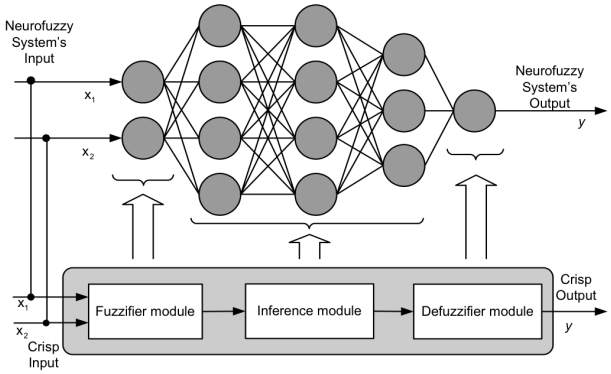
- Large number (¿4) of linguistic variables - construction of FRB can be difficult



- Can be used with automatically partitioned input/output
- Reasoning can be adapted to changes in environment

Hybrid Neuro-Fuzzy Systems

- Architecture integrating NN and FL-based system in appropriate (parallel) structure
- One entity, as opposed to cooperative or concurrent NFS



ANFIS

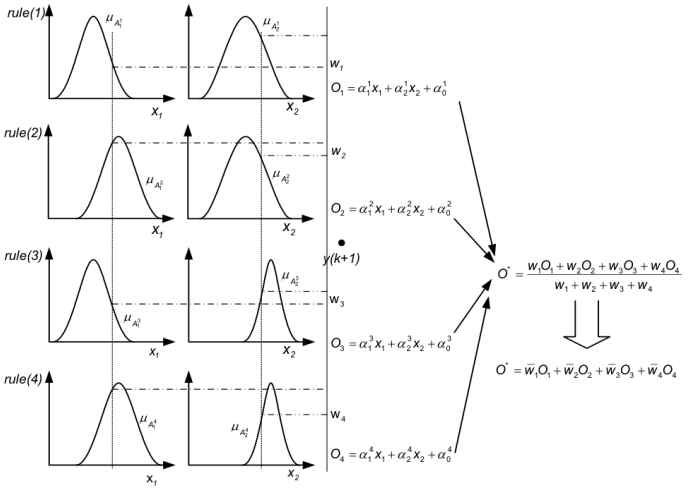
- Adaptive Network based Fuzzy Inference System
- Sugeno-tupe fuzzy system with rules in form

R_p: IF x_1 is A_1^p ANDx_2 is A_2^p AND $\ldots$ AND x_n is A_n^p
THEN $O_p = \alpha_0^p + \alpha_1^p x_1 + \ \ldots \ + \alpha_n^p x_n$

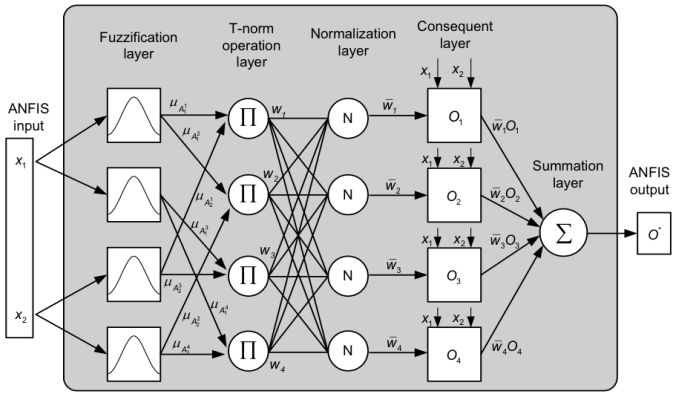
- 5-layer feed-forward network
  - Fuzzification

- **t**-norm operation
- Normalization
- Consequent
- Aggregation

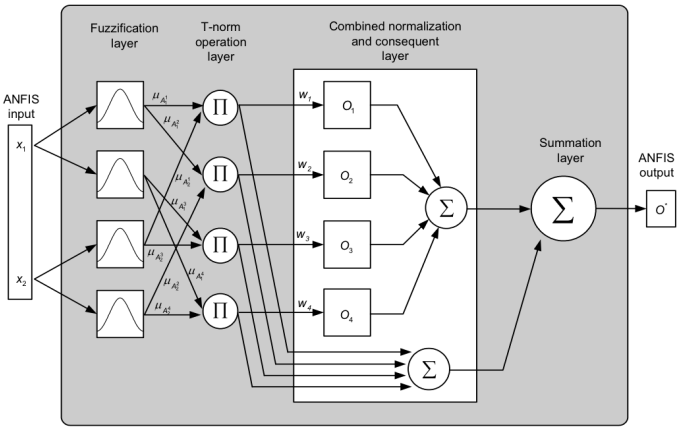
ANFIS example: 2 inputs, 1 output, 4 1^{st}-order rules



ANFIS example: architecture



ANFIS variant: 4-layer network

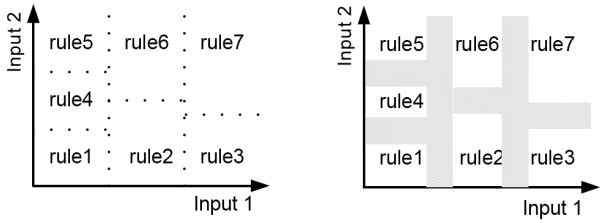


Construction of Neuro-Fuzzy Systems

- Structure identification pahse: I/O space partitioning
  - Each partition/patch represents a rule
- Parameter learning phase
  - Backpropagation learning
  - Hybrid learning

NFS Construction - Structure

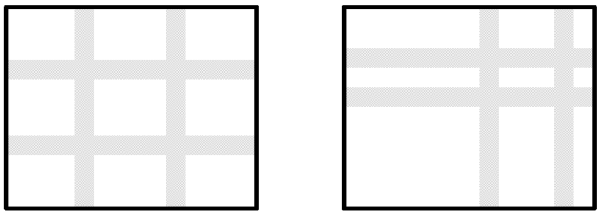
- Each partition/patch represents a rule (crisp vs. fuzzy)



- Partitioning approaches
  - Grid partitioning
  - Clustering
  - Scatter partitioning

Grid Partitioning

- Fixed vs. adaptive grid partitioning



+ Rules can be generated using a (relatively) small number of linguistic variables.
– Size of rule-base grows exponentially (so called *curse of dimensionality*).

Clustering

- Hard vs.soft partitioning (cf. crisp vs. fuzzy partition)

Most important approach: fuzzy c-means which represents $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^d$ using c-clusters with centers at locations

$$v_i = \frac{\sum_{j=1}^{n}(\mu_{ij} x_j)}{\sum_{j=1}^{n} \mu_{ij}}$$

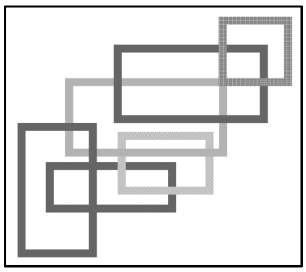
These locations are optimized using the objective function

$$J(U, V) = \sum_{j=1}^{n} \sum_{i=1}^{c} \mu_{ij}^{m} \|x_j - v_i\|^2$$

where U is partition matrix composed of memberships of each data point x_j in each cluster v_i.

Scatter Partitioning

- Antecedent (IF–part) of rules can be positioned at arbitrary locations of input space



+ Limits the number of generated fuzzy rules
– Finding suitable number of rules and their positions and widths is difficult

Parameter learning phase

Learning algorithms - strategies for optimizing weights/biases of NNs, or MFs and rules of Fuzzy rule-based systems

- Backpropagation learning: requires differentiability, suffers from known problems (local minima, plateaus)
- Hybrid learning: combination of two or more learning algorithms (RBF, SOM, LMS, BP, ...)

Bibliography

[1] Karray, F. O. and De Silva, C., 2004, *Soft Computing and Intelligent Systems Design - Theory, Tools and Applications*, Pearson/Addison Wesley.

[2] Bellman, R. E., 1978, *An Introduction to Artificial Intelligence: Can Computers Think?*, San Francisco: Boyd and Fraser

[3] Stottler, R., 1999, http://www.stottlerhenke.com/ai_general/quotations.htm.

[4] Webopedia, 2006, http://webopedia.com/TERM/A/artificial_intelligence.html.

[5] J. C. Bezdek,1994, What is computational intelligence?, in *Computational Intelligence Imitating Life*, New York, IEEE Press, 1994, pp. 1–12.

[6] T. Kohonen, 1988, An Introduction to Neural Computing, *Neural Networks*, 1(1), 3–16

[7] McCulloch, W. S. and W. H. Pitts (1943), "A Logical Calculus of the Ideas Imminent In Nervous Activity," *Bulletin of. Mathematics and Biophysics*, Vol. 5, pp. 115-133.

[8] Hebb, D. O., 1949, *The Organization of Behavior: A Neuropsychological Theory*, New York, NY: John Wiley.

[9] Farley, B. G. and W. A. Clark, 1954, "Simulation of Self-Organization Systems by Digital Computer," *Institute of Radio Engineers – Transactions of Professional Group of Information Theory*, Vol. PGIT-4, pp. 76-84.

[10] Rosenblatt, F., 1958, "The Perceptron: A Probabilistic Model for Information Storage And Organization in the Brain," Psychological Review, Vol. 65, pp. 386-408.

[11] Widrow, B. and M. E. Hoff, Jr., 1960, "Adaptive Switching Circuits," IRE WESCON Convention Record, Part 4, New York, NY: IRE, pp. 96-104.

[12] Minsky, M. and S. Papert, 1969, *Perceptrons*, Cambridge, MA: MIT Press.

[13] Amari, S. I., 1977, "Neural Theory of Association and Concept Formation," *Biological Cybernetics*, Vol. 26, pp. 175-185.

[14] Fukushima, K., 1975, "Cognitron: A Self-Organizing Multi Layered Neural Network," *Biological Cybernetics*, Vol. 20, pp. 121-136.

[15] Kohonen, T., 1982, "Self-Organizing Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, Vol. 43, pp. 59-69.

[16] Grossberg, S., 1982, *Studies of Mind and Brain: Neural Principles of Learning Perception, Development, Cognition, and Motor Control*, Boston, MA: Reidel Press.

[17] Hopfield, J. J. and D. W. Tank, 1985, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141-154.

[18] Werbos, P. J. 1974, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science*, Ph.D. Thesis, Cambridge, MA: Harvard University.

[19] Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986, "Learning Representations by Back-Propagating Errors," *Nature*, Vol. 323, pp. 533-536.

[20] Menger, K. Statistical metrics. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 28, pages 535–537, December 15 1942.

[21] S. Chen, C.F. Cowan, and P.M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. IEEE Trans. Neural Networks, 2:302–309, 1991