Chapter 1: Introduction to Soft Computing

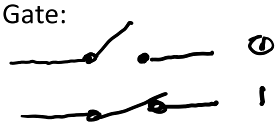
What is Soft Computing?

Soft computing is an emerging approach to computing, which parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision.
(Dr. Lotfi A. Zadeh)

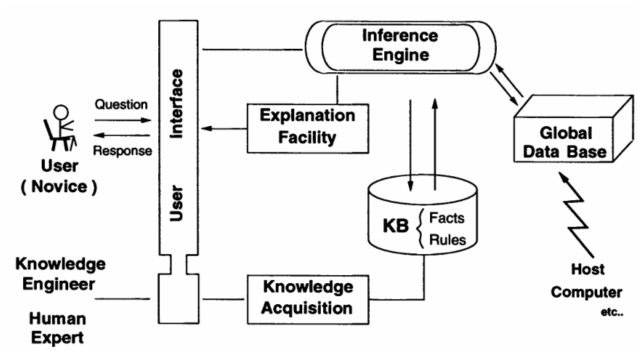
Conventional Artificial Intelligence (AI) and Soft Computing (SC)

- Symbolic tools
- Crisp calculation < 1 : true, 0 : False

Gate:



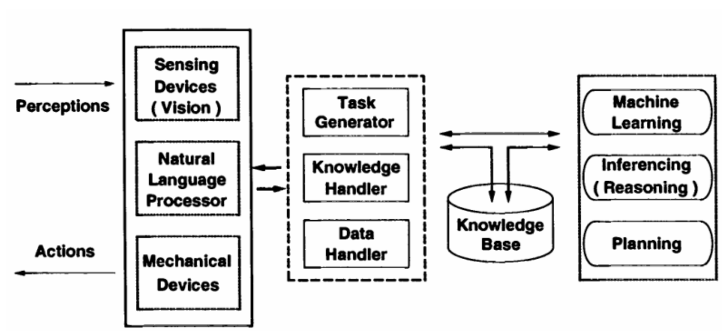
AI Intelligent System (expert system)



Advantages:
- Easy to set up
- Still commonly used today

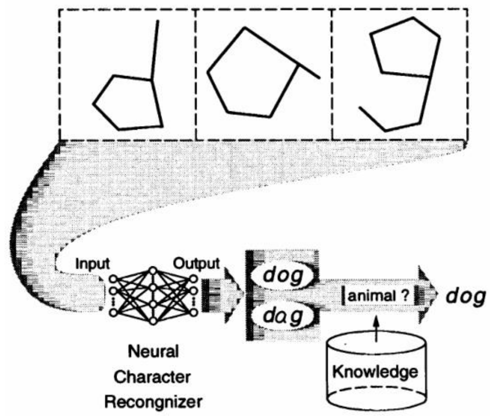
Disadvantages:
- Difficult to improve, how do you implement new knowledge to the system?
- How do you represent knowledge to the users?

SC Intelligent System



It is intelligent because it can sense the environment (temperature, acceleration, etc.)

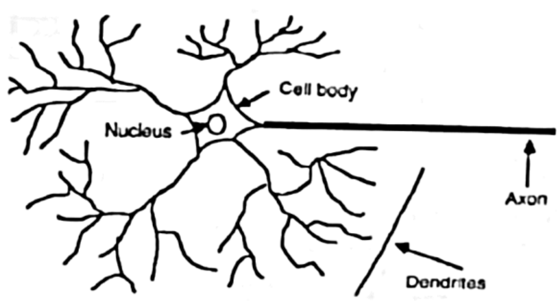
Example of a SC Intelligent System:



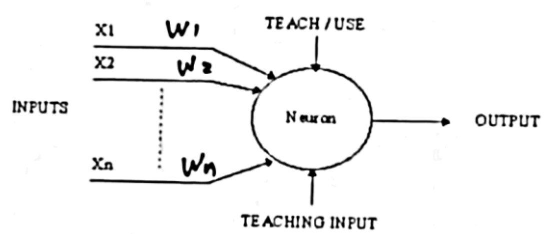
Two systems co-operate with each other to determine the written character.

Neural Networks

Consider a neuron:



This can be used to adapt the following model:



- Mimic the human brain to make decisions
- The functional speed of nerve cells is significantly slower than an electronic gate
- However, the brain can process A/V information significantly faster than a computer
    - especially under uncertainty and noise

Advantages:
- Good adaptive capability
- Can use machine learning algorithms to improve its function

Disadvantages:
- Difficult to recognize *how* decisions are made in the model
- Reasoning is complex, not immediately comprehensible

Fuzzy Logic

- Mimic linguistic reasoning to make a decision
    - Specifically, IF-THEN rules are utilized

IF – THEN rules

- Based on inputs, make a decision
    - IF (food quality = good, service = fast) THEN (tip = 20%)
    - IF (food quality = okay, service = slow) THEN (tip = 10%)

Advantages:

- Easy to follow logical process
- Intuitive

Disadvantages:

- Poor learning capability
- Difficult to optimize if application changes, or the environment varies

Table: Soft computing constituents (the first three items) and conventional artificial intelligence

Methodology	**Strength**
Neural network	Learning and adaptation
Fuzzy set theory	Knowledge representation via fizzy if-then rules
Genetic algorithm and simulated annealing	Systematic random search
Conventional AI	Symbolic manipulation

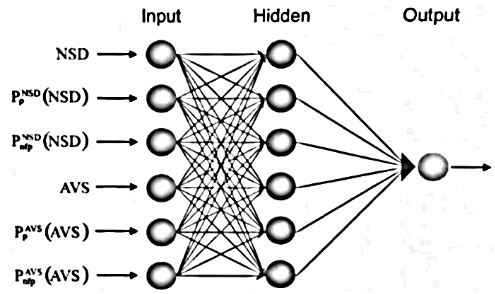
Neuro-fuzzy modeling

- Combining a fuzzy logic system with a neural network leads to a neuro-fuzzy system.

Why neuro-fuzzy?

	Fuzzy Logic	**Neural Networks**
Knowledge Representation	Linguistic description of knowledge	Knowledge distributed within computational unit
Intuitiveness	Explicit and easy to interpret (intuitive)	Implicit and difficult to interpret (“black box”)
Verification	Easy	Not as easy
Adaptation	Manual	Automatic
Learning	None	Excellent tools for imparting learning

Consider:



Historical development of related techniques:

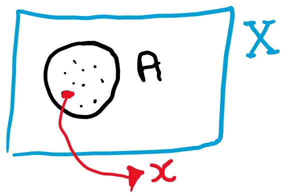
	Conventional AI	Neural networks	Fuzzy systems	Other methodologies
1940s	1947 Cybemetics	1943 McCulloch-Pitts neuron model		
1950s	1956 Artificial Intelligence	1957 Perceptron		
1960s	1960 Lisp language	1960s Adaline Madaline	1965 Fuzzy sets	
1970s	mid- 1970s Knowledge Engineering (expert systems)	1974 Birth of Back-propagation algorithm 1975 Cognitron Neocognitron	1974 Fuzzy controller	1970s Genetic algorithm
1980s		1980 Self-organizing map 1982 Hopfield Net 1983 Boltzmann machine 1986 Backpropagation algorithm boom	1985 Fuzzy modeling (TSK model)	mid- 1980s Artificial life Immune modeling
1990s			1990s Neuro-fuzzy modeling 1991 ANFIS 1994 CANFIS	1990 Genetic programming

Chapter 2: Fuzzy Sets

2.1 Review of Conventional AI (Crisp Logic or Crisp Sets)

Logic $\sim$ representation of processing of knowledge

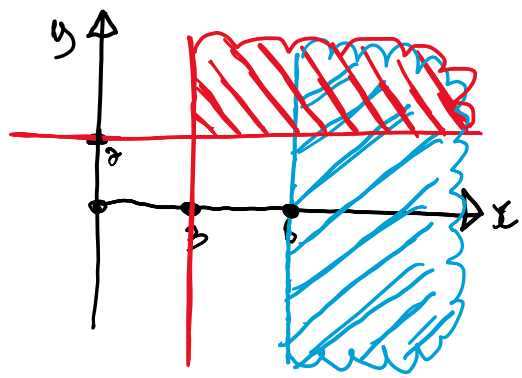
$A \sim$ a set
$X \sim$ universe of discourse



$\phi \sim$ null set
$x \sim$ element in set A
$x \in A$

Consider:

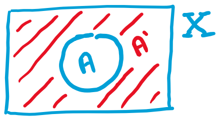
$A = \{\, x \mid x \geq 6 \,\}$
$B = \{\, x, y \mid x \geq x \geq 3, y \geq 2 \,\}$



1) Operations of Sets

1. <u>Complement</u>

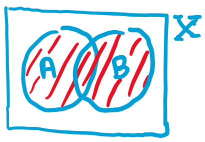
Given set A, the complement is everything outside of $A \to A'$



2. <u>Union</u>

Given sets A and $B \to A \cup B$

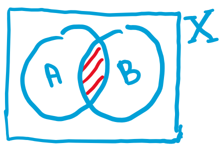
union



3. <u>Intersection</u>

Given sets A and $B \to A \cap B$

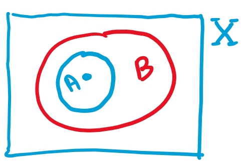
intersection



4. <u>Subset</u>

subset

Set A is a subset of B, if the elements of A are contained within $B \to A \subset B$ (or $A \subseteq B$)



2) Conventional Logic

Knowledge statement is represented by proposition.

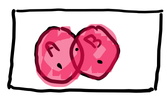
Truth table: True (T) / False (F)

1. Negation (NOT)

$$\sim A$$

$\boldsymbol{A}$	$\boldsymbol{\bar{A}}$
T	F
F	T

2. Disjunction (OR)

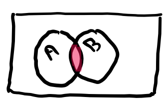


no X (logic representation)

$$A \lor B$$

$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{A \lor B}$
T	T	T
T	F	T
F	T	T
F	F	F

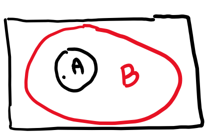
3. Conjunction (AND)



$$A \land B$$

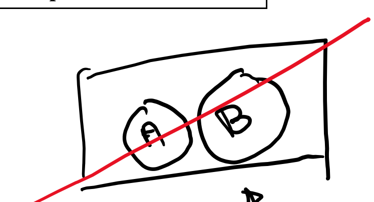
$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{A \land B}$
T	T	T
T	F	F
F	T	F
F	F	F

4. Implication (IF-THEN)

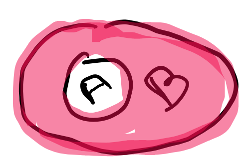


IF A THEN B

$A \rightarrow B$



$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{A \rightarrow B}$
T	T	T
T	F	F *contradicts logic*
F	T	T *does not contradict logic* *could be T, could be F*
F	F	T *(same reason as above)*



Boolean Algebra: 0 or 1

Table 1.2: Isomorphism between set theory, logic, and Boolean algebra

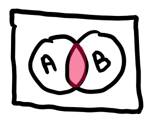
Set theory concept	Set theory notation	Binary logic concept	Binary logic notation	Boolean algebra notation
Universal set	X	(Always) true	(*Always*) T	1
Null set	$\emptyset$	(Always) false	(*Always*) F	0
Complement	A'	Negation (*NOT*)	$\overline{A}$ or $\sim A$	$\overline{A}$
Union	$A \cup B$	Disjunction (*OR*)	$A \vee B$	$A + B$
Intersection	$A \cap B$	Conjunction (*AND*)	$A \wedge B$	$A \cdot B$
Subset	$A \subseteq B$	Implication (if-then)	$A \rightarrow B$	$A \leq B$

3) Logic Processing

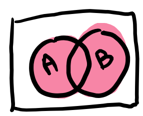
Knowledge is usually represented by a proposition.
Consider 3 propositions; A, B, and C.
$X(always\ T)$ and $\phi(always\ F)$

1. Commutativity

$$A \wedge B = B \wedge A$$

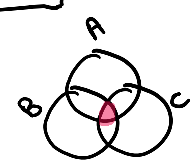


$$A \vee B = B \vee A$$

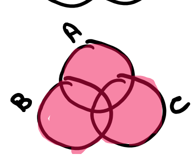


2. Associativity

$$(A \wedge B) \wedge C = A \wedge (B \wedge C)$$



$$(A \vee B) \vee C = A \vee (B \vee C)$$



3. Distributivity

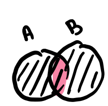
$$A \wedge (B \vee C) = (A \wedge B) \vee (B \wedge C)$$

$$A \vee (B \wedge C) = (A \vee B) \wedge (B \vee C)$$

4. Absorption

$$A \vee (A \wedge B) = A$$

$$A \wedge (A \vee B) = A$$

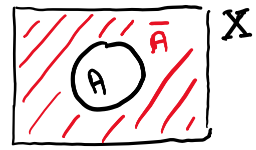


5. Idempotency

$$A \vee A = A$$

$$A \wedge A = A$$

6. Exclusion



$$A \vee \overline{A} = X = (always\ T)$$

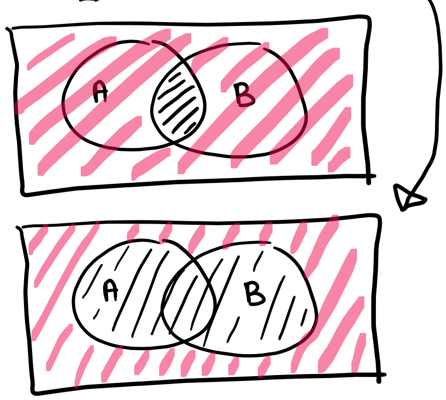
7. Contradiction

$$A \wedge \overline{A} = \phi = (always\ F)$$

8. De Morgan's Law

$$\overline{A \wedge B} = \bar{A} \vee \bar{B}$$

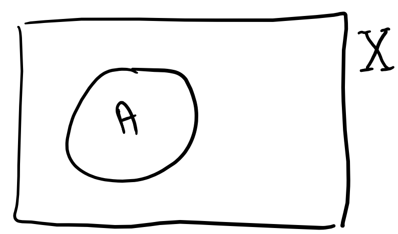
$$\overline{A \vee B} = \bar{A} \wedge \bar{B}$$



9. Boundary Conditions

$$A \vee X = X = T$$
$$A \wedge X = A$$

$$A \vee \phi = A$$
$$A \wedge \phi = \phi = F$$



Consider:
$A =$ "food is good"
$B =$ "service is good"

It is not true that both "food is good" and "service is good" ?!
Either "food is not good" or "service is not good"

4) Rules of Inference

1. <u>Conjunction rule of inference (CRI)</u>

$$(A, B) \rightarrow A \wedge B$$

IF proposition A is TRUE and the second proposition B is TRUE
THEN the combined proposition A ∧ B is also TRUE

2. <u>Modus Ponens</u>

$$A \ \wedge \ (A \ \rightarrow \ B) => \ B$$

IF A $\rightarrow B$, then B
IF A is T, then B is T

Consider:
$A =$ "food is good"
$B =$ "tips are generous"

Thus, IF "food is good" THEN "tips are generous"

3. <u>Modus Tollens</u>

$$\bar{B} \wedge (A \rightarrow B) => \bar{A}$$

IF proposition B is not T, and A $\rightarrow B$ holds
THEN A is NOT TRUE

4. <u>Hypothetical Syllogism</u>

$$[(A \rightarrow B) \wedge (\mathrm{B} \rightarrow \mathrm{C})] => (A \rightarrow C)$$

Consider:
$A =$ "machine performs well"
$B =$ "vibration level is low"
$C =$ "manufacturing accuracy is high"

If both terms hold, then
IF "machine performs well" THEN "manufacturing accuracy is high"

2.2 Concepts of Fuzzy Sets

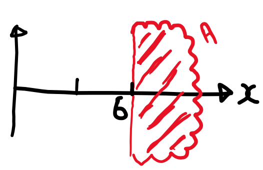
Conventional crisp logic utilizes 0 /1 (or true/false)

- Good for math operations
- Good for computers, chip designs

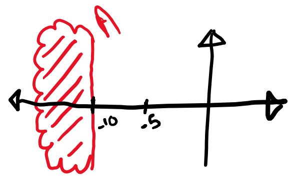
However, real applications have many abstract and imprecise concepts (think "grey" areas)

Crisp logic $\begin{cases} T = 1 \\ F = 0 \end{cases}$

Set:
$A = \{x | x \geq 6\}$



Temperature "cold"



Temp:

$A = \text{"}cold\text{"}$

$A = \{t \leq -10\ °C\}$

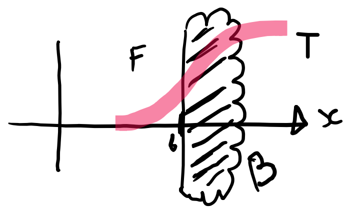
But what about:

$temp = -10.1\ °C$

$temp = -9.99\ °C$

$A \rightarrow T$

$A \rightarrow F$



$B =$ a person is tall

$B = \{x | x \geq 6\ ft\}$

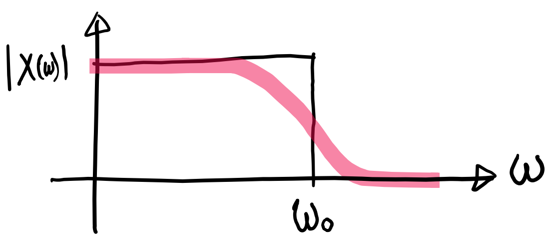
$x = 6.01\ ft$

$B \rightarrow T$

$x = 5.99\ ft$

Fuzzy Logic

Ideal filter:

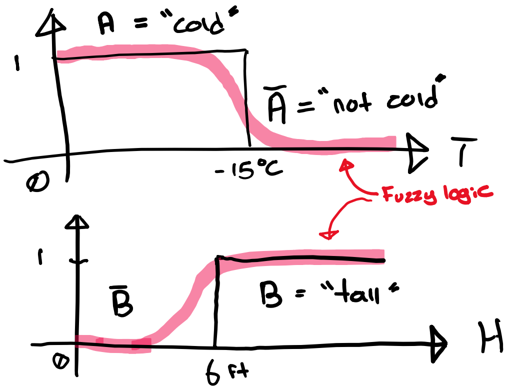


2.2 Fuzzy Sets (Cont'd)

Crisp sets: $A\begin{cases}1\\0\end{cases}$

Concepts and thoughts are abstract and imprecise $\neq$ random.

Fuzzy logic → approximate knowledge



Membership function (MF) grade.

Consider:
Height:
$H = 6.001\ ft$
"Tall" MF grade: 99.9%

$H' = 5.999\ ft$
"not Tall" MF grade: 0.01%

Fuzzy set:
$A = \{x, \mu_A(x)\}$

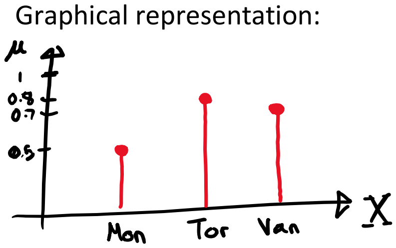
$x =$ variable $\in X$
$\mu_A = MF$
$X =$ universe of discourse

1) Fuzzy sets with a discrete non-ordered universe

$X = \{Montreal, Toronto, Vancouver\}$

$C = \text{"desired city to live in"}$
$C = \{(Mon, 0.5), (Tor, 0.8), (Van, 0.7)\}$

Graphical representation:



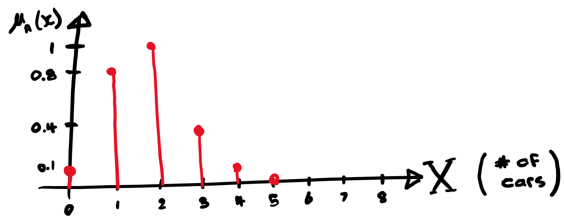
2) Fuzzy set with a discrete ordered universe

$X = \{0, 1, 2, 3, 4\}$

Fuzzy set A = "sensible number of cars in a family"

$A = \{(0, 0.1), (1, 0.8), (2, 1.0), (3, 0.4), (4, 0.1)\}$

Graphical representation:



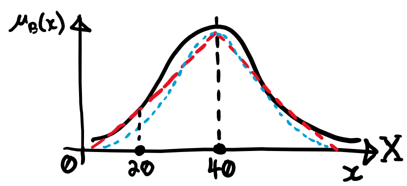
3) Fuzzy sets with a continuous space

$X =$ "ages" $(0 \sim 120)$

Fuzzy set B = "about 40 years old"

$B = \{(x, \mu_B(x)\}, x \in X$

Graphical representation:



$$\mu_B(x) = \frac{1}{1 + \left(\frac{x - 40}{10}\right)^4}$$

- Subjective (X, MF)
- Not random

4) Other fuzzy set representations

For a discrete, non-ordered universe:

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_B(x_2)}{x_2} + \frac{\mu_C(x_3)}{x_3} + \cdots$$

e.g.

$$C = \frac{0.5}{Mon} + \frac{0.8}{Tor} + \frac{0.7}{Van}$$

For a discrete, ordered universe:

$$A = \frac{0.1}{0} + \frac{0.8}{1} + \frac{1.0}{2} + \frac{0.4}{3} + \frac{0.1}{4}$$

For a continuous space:

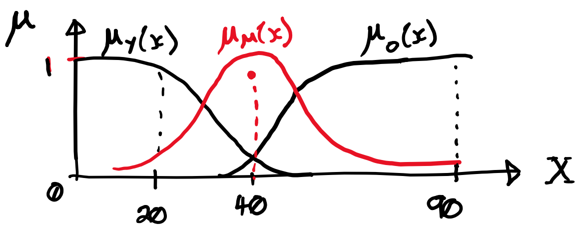
$$B = \frac{\mu_B(x)}{x}$$

$$B = \left[\frac{1}{1 + \left(\frac{x-40}{10}\right)^4}\right] \Big/ x$$

If the universe space X is a continuous space, we can partition X into several fuzzy sets.

Consider:
$X =$ "age"

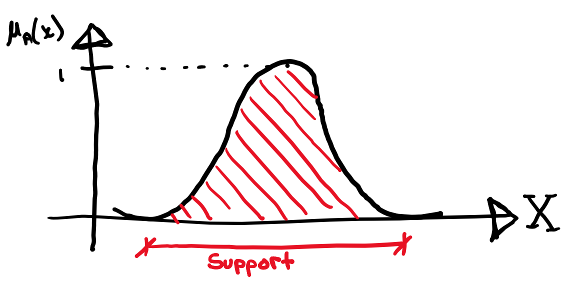
Partitions:
Y M O
"young", "middle aged", "old"
$\mu_Y(x),\ \mu_M(x),\ \mu_O(x),\ where\ x \in X$



2.3 Other Concepts of Fuzzy Sets

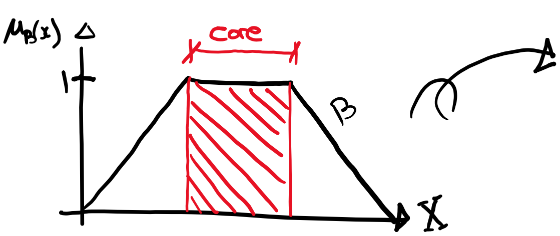
1) Support

$support(A) \rightarrow \{x | \mu_A(x) > 0\}$



2) Core

$core(B) \rightarrow \mu_B(x) = 1$

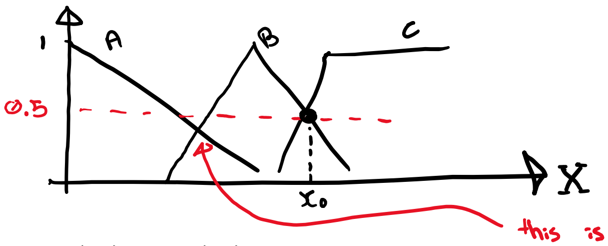


(For a triangular function, only 1 point would be in the core)

3) Normality

$normality(C) \rightarrow \max\{\mu_C(x)\} = 1$

4) Cross-over Points



$\mu_B(x_0) = \mu_C(x_0) = 0.5$

x_0 = a cross-over point

5) Fuzzy singletons

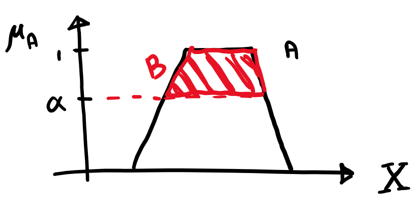
Basically, a fuzzy set in discrete form.

Diagnosis:

Class 1, Class 2, …

6) $\alpha - cut$

$B = \{x, \mu_B(x)|_{\mu_A(x) \geq \alpha}\}$

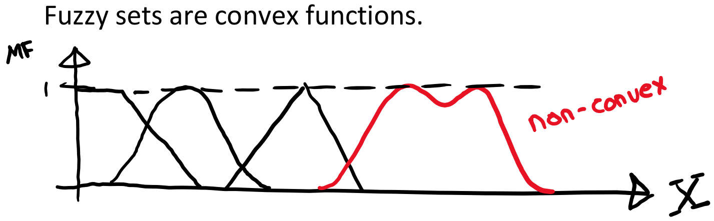


- Strong $\alpha - cut$

  $B = \{x, \mu_B(x) | \mu_A(x) > \alpha\}$

7) Convexity

Fuzzy sets are convex functions.



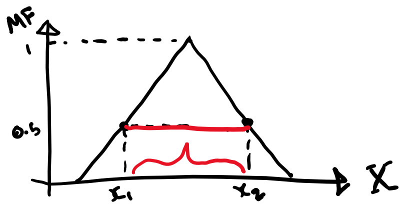
8) Fuzzy Numbers

A fuzzy number is a fuzzy set

→ normality

→ convexity (monotonically increasing, followed by monotonically decreasing, or constant)

9) Bandwith



$x_2 - x_1$ = bandwith

$\mu_A(x_1) = \mu_A(x_2) = 0.5$

10) Symmetry

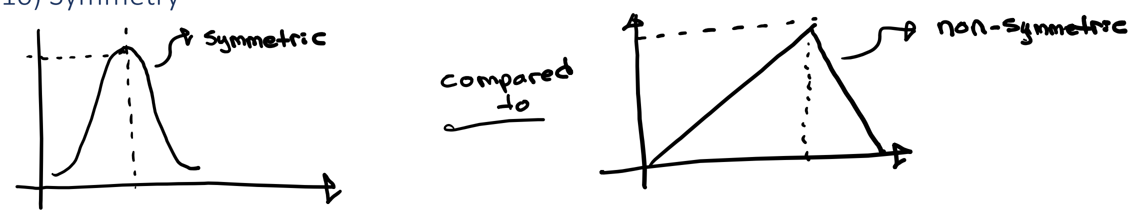


Table 2.1: Some properties of fuzzy sets

Property name	Relation
Commutativity	$A \cap B = B \cap A$ $A \cup B = B \cup A$
Associativity	$(A \cap B) \cap C = A \cap (B \cap C)$ $(A \cup B) \cup C = A \cup (B \cup C)$
Distributivity	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
Absorption	$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$
Idempotency (Idem = same; potent = power) (Similar to unity or identity operation)	$A \cup A = A$ $A \cap A = A$
Exclusion: Law of excluded middle Law of contradiction	 $A \cup A' \subset X$ $A \cap A' \supset \phi$
DeMorgan's Laws	$(A \cap B)' = A' \cup B'$ $(A \cup B)' = A' \cap B'$
Boundary conditions	$A \cup X = X$ $A \cap X = A$ $A \cup \phi = A$ $A \cap \phi = \phi$

2.4 Set Operations

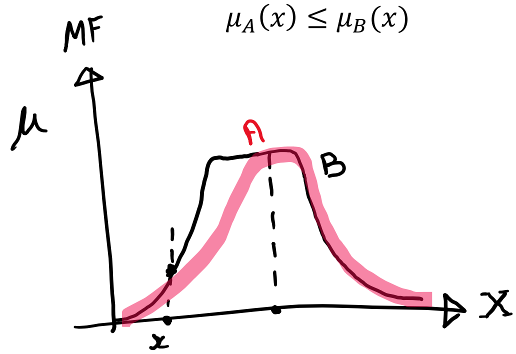
1) Subset

Consider fuzzy set A & B

If A is a subset of B:

$$A \subset B$$

$$\mu_A(x) \leq \mu_B(x)$$

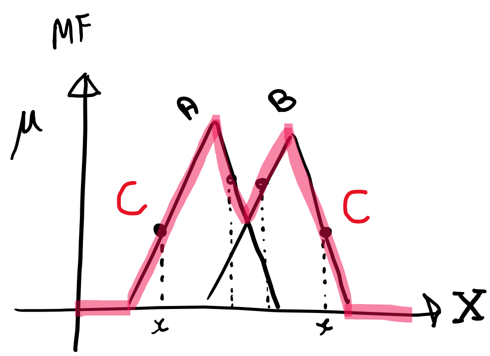


2) Union (Disjunction) - OR

Given A & B

$$C = A \cup B$$

$$\mu_c(x) = \max\{\, \mu_A(x),\ \ \mu_B(x)\, \}$$

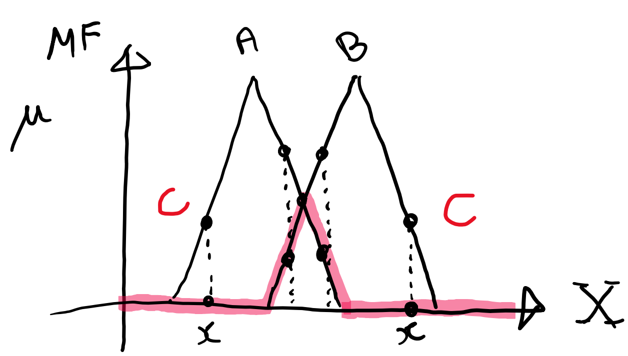


3) Intersection (Conjunction) – AND

Given A & B

$$C = A \cap B$$

$$\mu_C(x) = \min\{\, \mu_A(x),\ \ \mu_B(x)\, \}$$

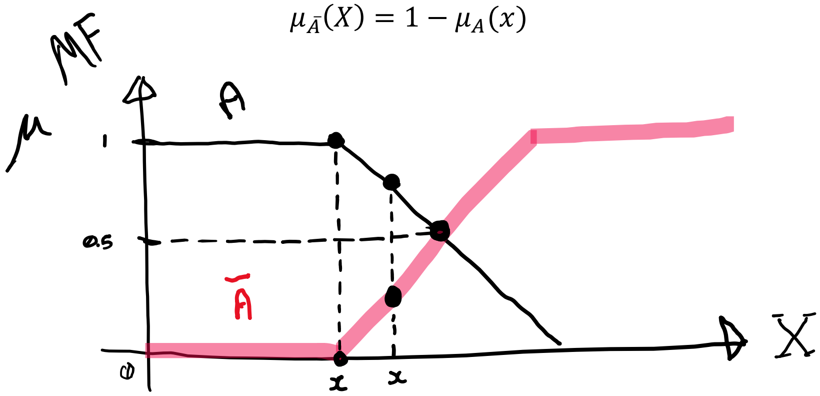


4) Complement (Negation) – NOT

Given A & B

$$not\ A \text{ or } \bar{A} \text{ (fuzzy set)}$$

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



5) Cartesian Product / Co-product

$A \sim$ fuzzy set in X
$B \sim$ fuzzy set in Y } "different universes / domains"

Cartesian product A x B
is in X x Y

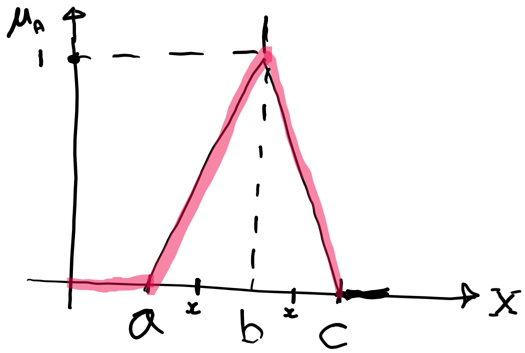
$$\mu_{A \text{ x } B}(x, y) = \min\{\ \mu_A(x),\ \ \mu_B(y)\ \}$$

Cartesian co-product $A + B$
is in $X + Y$

$$\mu_{A+B}(x, y) = \max\{\ \mu_A(x),\ \ \mu_B(y)\ \}$$

2.4 Membership Functions (MF)

1) Triangular Membership Functions

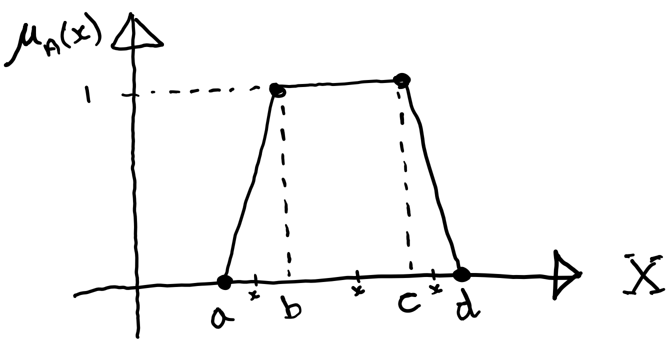


$$\mu_A(x; a, b, c) = \begin{cases} 0 & when\ x \leq a \\ \left(\frac{x-a}{b-a}\right) & when\ a < x < b \\ \left(\frac{c-x}{c-b}\right) & when\ b < x < c \\ 0 & when\ x > c \end{cases}$$

In MATLAB:

$trimf(x, [a\ b\ c])$

2) Trapezoidal Membership Functions



$$\mu_A(x; a, b, c, d) = \begin{cases} 0 & when\ x < a \\ \left(\frac{x-a}{b-a}\right) & when\ a \leq x < b \\ 1 & when\ b \leq x < c \\ \left(\frac{d-x}{d-c}\right) & when\ c \leq x \leq d \\ 0 & when\ x > d \end{cases}$$

In MATLAB:

$trapmf(x, [a\ b\ c\ d])$

NOTE: Triangular, and trapezoidal membership functions are not continuous, which means the derivatives functions do not exist (equal to zero).

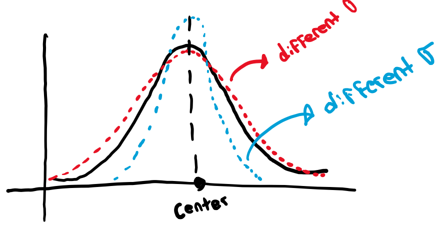
The following membership functions are continuous:

3) Gaussian Membership Functions

$\mu_A = gauss(x; c, \sigma) = G(x) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$

c = center

σ = spread



In MATLAB:

$gaussmf(x; [c, \sigma])$

$$\mu'(x) = DG(x) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2} \cdot \left[-\frac{1}{2} \cdot 2\left(\frac{x-c}{\sigma}\right) \cdot \frac{1}{\sigma}\right]$$

4) Generalized Bell Membership Functions

$$\mu_A = bell(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}$$

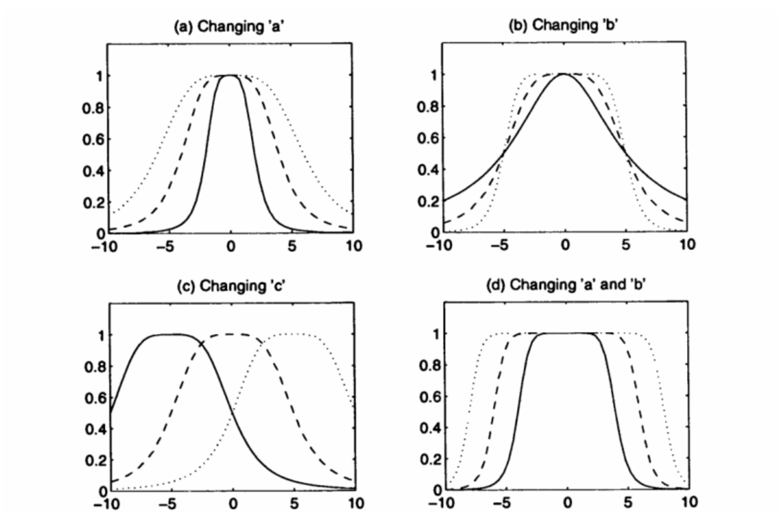


Figure 2.8. *The effects of changing parameters in bell MFs: (a) changing parameter a; (b) changing parameter b; (c) changing parameter c; (d) changing a and b simultaneously but keeping their ratio constant.* (MATLAB file: `allbells.m`)

In MATLAB:

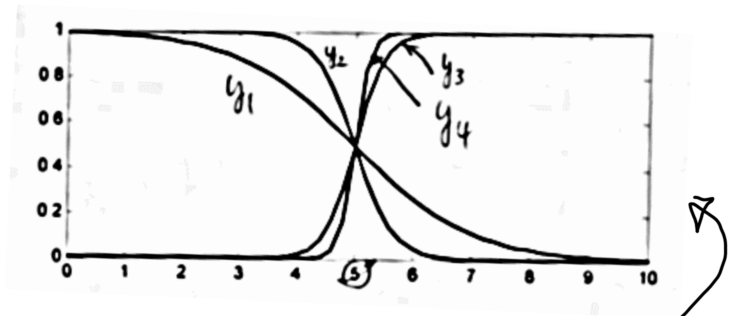
$gbellmf(x; [a, b, c])$

5) Sigmoid Membership Functions

$$\mu(x) = sig(x; a, c) = \frac{1}{1 + exp[-a(x-c)]}$$

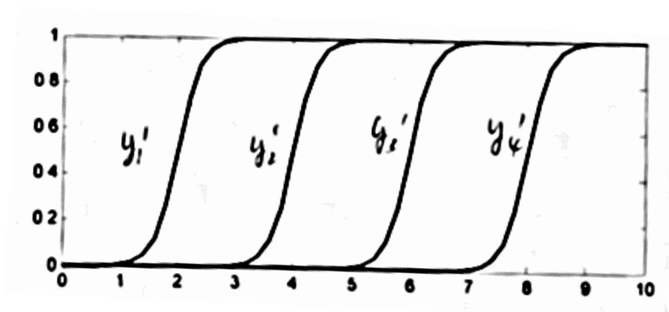
<u>In MATLAB:</u>

$sigmf(x; [a, c])$



If $a > 0$, $sigmf$ opens to the right

If $a < 0$, $sigmf$ opens to the left



$$\mu'(x) = DS(x) = -1\left[1 + \mathrm{e}^{-a(x-c)}\right]^{-2} e^{-a(x-c)} \cdot (-a)$$

$0 \le x < \infty$

2.5 Fuzzy Operations

MFs [0, 1]

$A \sim [0, 1]$
$B \sim [0, 1]$
$C \sim A \cup B$
$D \sim A \cap B$

$$[0, 1] \text{ x } [0, 1] \rightarrow [0, 1]$$

1) Triangular Norm (T-Norm) – Generalized Intersection

$a = \mu_A(x)$
$b = \mu_B(x)$

$T = (a, b), \quad aTb$

Properties in table below.

2) T-Conorm (S-Norm)

Properties in table below.

Table 2.2: Some properties of a triangular norm

Item description	T-norm (triangular norm)	S-norm (T-conorm)
Function	$T: [0, 1] \times [0, 1] \to [0, 1]$	Same
Nondecreasing in each argument	If $b \geq a$, $d \geq c$ then $bTd \geq aTc$	Same
Commutative	$aTb = bTa$	Same
Associative	$(aTb)Tc = aT(bTc)$	Same
Boundary conditions	$aT1 = a$ $aT0 = 0$ with $a, b, c, d \in [0, 1]$	$aS0 = a$ $aS1 = 1$
Examples	Conventional: $\min(a, b)$ Product: ab Bounded max (bold intersection): $\max[0, a + b - 1]$ General: $1 - \min[1, ((1 - a)^p + (1 - b)^p)^{1/p}]$ $p \geq 1$ $\max[0, (\lambda + 1)(a + b - 1) - \lambda ab]$ $\lambda \geq -1$	Conventional: $\max(a, b)$ Set addition: $a + b - ab$ Bounded min (bold union): $\min[1, a + b]$ General: $\min(1, (a^p + b)^p)^{1/p}$ $p \geq 1$ $\min[1, a + b + \lambda ab]$ $\lambda \geq -1$
DeMorgan's Laws	$aSb = 1 - (1 - a)\ T(1 - b)$ $aTb = 1 - (1 - a)\ S(1 - b)$	

Example 2.3 (Similar to Example 2.13)

Use DeMorgan's law to determine the S-norm corresponding to max(x, y), and T-norm corresponding to min(x, y).

Solution 2.3

$$
\begin{aligned}
&xSy = 1 - (1 - x)T(1 - y)\\
&T \rightarrow \min\\
&= 1 - \min[(1 - x), (1 - y)]\\
&= \begin{cases} 1 - (1 - y) = y; & x < y \\ 1 - (1 - x) = x; & x \geq y \end{cases}\\
&xSy = \max(x, y)
\end{aligned}
$$

Example 2.4 (Similar to Example 2.14)

Prove that the min operator is the largest T-norm and the max operator is the smallest S-norm.

Solution 2.4

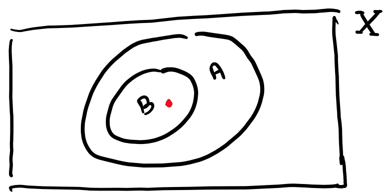
Nondecreasing, boundary conditions

$$
\begin{aligned}
&xTy \leq 1Ty = y\\
&xTy \leq xT1 = x\\
&xTy \leq \min(x, y)
\end{aligned}
$$

2.5 Fuzzy Operations

3) Set Inclusion

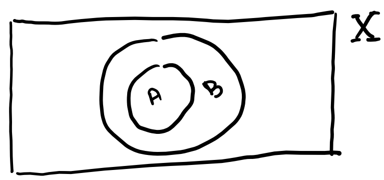
$B \subset A$



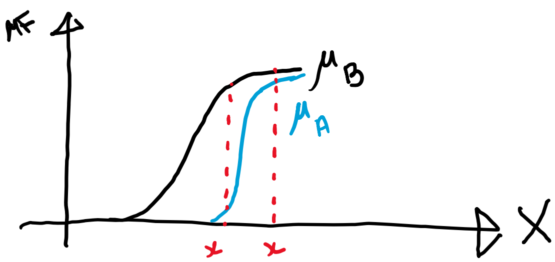
Fuzzy sets A, B

If A is a subset of fuzzy set B,

$A \subset B$

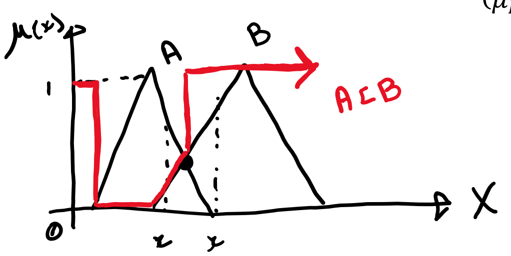


$$\mu_{A \sqsubset B}(x) = \begin{cases} 1 & ; \text{ if } \mu_A(x) \leq \mu_B(x) \\ \mu_A(x) \mathrm{T} \mu_B(x) & ; \text{ Otherwise} \end{cases}$$



min ~ T-norm

$$\mu_{A \sqsubset B}(x) = \begin{cases} 1 & ; \text{ if } \mu_A(x) \leq \mu_B(x) \\ \mu_B(x) & ; \text{ Otherwise} \end{cases}$$



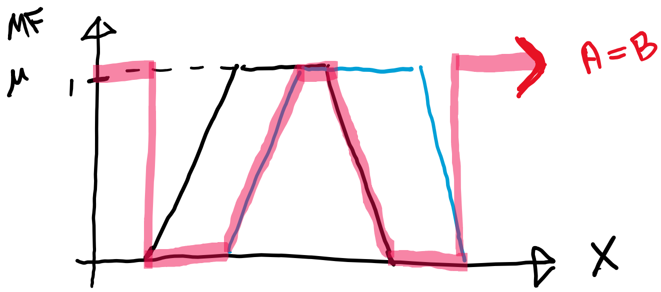
4) Set Equality ($A = B$)

$\mu_A(x) = \mu_B(x)$

$$\mu_{A=B}(x) = \begin{cases} 1 & ; \quad \text{if } \mu_A(x) = \mu_B(x) \\ \mu_A(x) \mathrm{T} \mu_B(x) & ; \quad \text{otherwise} \end{cases}$$

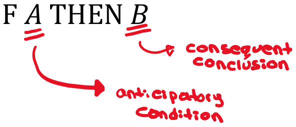
min ~ T-norm

$$\mu_{A=B}(x) = \begin{cases} 1 & ; \quad \text{if } \mu_A(x) = \mu_B(x) \\ \min\big(\mu_A(x), \mu_B(x)\big) & ; \quad \text{if } \mu_A(x) \neq \mu_B(x) \end{cases}$$



2.6 Implication (IF – THEN)

$$A \rightarrow B$$
$$\text{IF } A \text{ THEN } B$$



$A \sim X$

$B \sim Y$

$A \rightarrow B, X \times Y$

1) Method 1 (Mamdani implication)

$$\mu_{A \rightarrow B}(x, y) = \min[\mu_A(x), \mu_B(y)]$$
$$x \in X, y \in Y$$

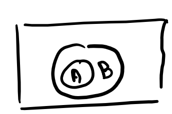
2) Method 2 (Larson implication)

$$\mu_{A \rightarrow B}(x, y) = \mu_A(x) \cdot \mu_B(y)$$

3) Method 3 (Bounded sum implication)

$$\mu_{A \rightarrow B}(x, y) = \min[1, \{1 - \mu_A(x) + \mu_B(y)\}]$$

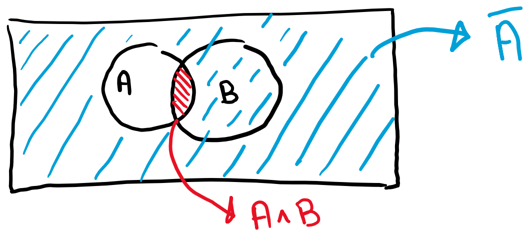
$\mu_{\bar{A}}(x)$



Proof:

A	B	$A \rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

$$A \rightarrow B = (A \wedge B) \vee \bar{A}$$



$$A \rightarrow B = (A \vee \bar{A}) \wedge (B \vee \bar{A})$$
$$= X \wedge (B \vee \bar{A})$$
$$= B \vee \bar{A}$$
$$= 1 - (B \vee A)$$

4) Method 4 (Zadeh implication)

$$\mu_{A \rightarrow B}(x, y) = \max[\min\{\mu_A(x), \mu_B(y)\},\ 1 - \mu_A(x)]$$

$\mu_{\bar{A}}(x)$

5) Method 5 (Dienes-Rescher implication)

$$\mu_{A \rightarrow B}(x, y) = \max[1 - \mu_A(x), \mu_B(y)]$$

$$\text{IF } \mu_A(x) = 0.6$$
$$\text{IF } \mu_B(x) = 0.5$$

Method 1 (Mamdani):
$= \min(0.6, 0.5)$
$= 0.5$

Method 2 (Larson):
$= \text{product}(\mu_A(x), \mu_B(x))$
$= 0.6 * 0.5$
$= 0.3$

Method 3:
$= \min[1, \{1 - 0.6 + 0.5\}]$
$= \min\ [1, 0.9]$
$= 0.9$

Method 4:
$= \max[\min\{0.6, 0.5\}, 1 - 0.6]$
$= \max[0.5, 0.4]$
$= 0.5$

Method 5:
$= \max[1 - 0.6, 0.5]$
$= \max[0.4, 0.5]$
$= 0.5$

Example 2-6 (Problem 2.16)

Example 2.16

Consider the membership functions of fuzzy sets A and B as shown in Figure 2.10, and expressed below:

$$\mu_A(x) = \max\left\{0, \frac{10x-3}{2}\right\} \quad 0.3 \le x \le 0.5$$
$$= \max\left\{0, \frac{7-10x}{2}\right\} \quad 0.5 < x \le 0.7$$
$$= 0 \quad \text{otherwise}$$

$$\mu_B(y) = \max\left\{0, \frac{10y-3}{2}\right\} \quad 0.3 \le y \le 0.5$$
$$= \max\left\{0, \frac{7-10y}{2}\right\} \quad 0.5 < y \le 0.7$$
$$= 0 \quad \text{otherwise}$$

The resulting expressions for the combined membership functions, which represent the five implication relations, are given in (a)–(e) below, and sketched in Figure 2.11.

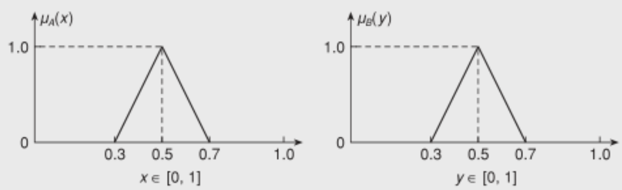


Figure 2.10: Membership functions of fuzzy sets A and B

Solution

(b) Mamdani implication (min operation)

$$\mu_{A\to B}(x,y) = \begin{cases} \min\left[\frac{10x-3}{2}, \frac{10y-3}{2}\right] & \text{if } 0.3 \le x \le 0.5 \text{ and } 0.3 \le y \le 0.5 \\ \min\left[\frac{10x-3}{2}, \frac{7-10y}{2}\right] & \text{if } 0.3 \le x \le 0.5 \text{ and } 0.5 < y \le 0.7 \\ \min\left[\frac{7-10x}{2}, \frac{10y-3}{2}\right] & \text{if } 0.5 < x \le 0.7 \text{ and } 0.3 \le y \le 0.5 \\ \min\left[\frac{7-10x}{2}, \frac{7-10y}{2}\right] & \text{if } 0.5 < x \le 0.7 \text{ and } 0.5 < y \le 0.7 \\ 0 & \text{otherwise} \end{cases}$$

(a) Larsen implication (product or dot operation)

$$\mu_{A\to B}(x,y) = \begin{cases} \frac{(10x-3)(10y-3)}{4} & \text{if } 0.3 \le x \le 0.5 \text{ and } 0.3 \le y \le 0.5 \\ \frac{(10x-3)(7-10y)}{4} & \text{if } 0.3 \le x \le 0.5 \text{ and } 0.5 < y \le 0.7 \\ \frac{(7-10x)(10y-3)}{4} & \text{if } 0.5 < x \le 0.7 \text{ and } 0.3 \le y \le 0.5 \\ \frac{(7-10x)(7-10y)}{4} & \text{if } 0.5 < x \le 0.7 \text{ and } 0.5 < y \le 0.7 \\ 0 & \text{otherwise} \end{cases}$$

2.7 Extension Principle and Fuzzy Relations

$f \sim$ from X to Y

↳A ↳B

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \cdots + \frac{\mu_A(x_n)}{x_n}$$

For fuzzy sets A and B

$$B = f(A)$$

$$y = f(x)$$

$$= \frac{\mu_A(x_1)}{y_1} + \frac{\mu_A(x_2)}{y_2} + \cdots + \frac{\mu(x_n)}{y_n}$$

Example:

$$A = \frac{0.1}{-2} + \frac{0.4}{-1} + \frac{0.8}{0} + \frac{0.9}{1} + \frac{0.3}{2}$$

→ MF grade

→ x

$$y = f(x) = x^2 - 3$$

$A \sim x \in X$
$B \sim y \in Y$

$$B = \frac{0.1}{1} + \frac{0.4}{-2} + \frac{0.8}{-3} + \frac{0.9}{-2} + \frac{0.3}{1}$$

Many to one mapping $\sim$ max

$$B = \frac{0.1 \vee 0.3}{1} + \frac{0.4 \vee 0.9}{-2} + \frac{0.8}{-3}$$

$$= \frac{0.7}{1} + \frac{0.9}{-2} + \frac{0.8}{-3}$$

Given fuzzy sets (X, Y)
Where: $x \in X, y \in Y$

$$\mu(x), \quad \mu(y), \quad 0\sim1 \text{ (binary relations)}$$

Binary fuzzy sets
Let X and Y be two universes of discourse.

$$R = \{(x, y), \mu_R(x, y)|_{XxY}\}$$

$\mu_R(x, y) \sim$ 2D membership function

$R =$ "y is greater than x"

$$\mu_R(x, y) = \begin{cases} \dfrac{y - x}{x + y - 2} & ; if\ y > x \\ 0 & ;\text{if } y \leq x \end{cases}$$

- $X = \{3, 4, 5\}$
- $Y = \{3, 4, 5, 6, 7\}$

$$R = \begin{array}{c} \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 0.111 & 0.200 & 0.273 & 0.353 \\ 0 & 0 & 0.091 & 0.167 & 0.231 \\ 0 & 0 & 0 & 0.077 & 0.143 \end{bmatrix} \end{array}$$

1) Max-Min Composition

$R_1 \sim$ fuzzy relation on X x Y
$R_2 \sim$ fuzzy relation on Y x Z
R_1 and $R_2 \sim$ fuzzy set X and Z

Max-Min Composition:

$$\mu_{R_1 \circ R_2}(x, z)$$
$$= \max\min[\mu_{R_1}(x, y), \mu_{R_2}(y, z)]$$
$$= V_y\left[\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z)\right]$$

Where:
$\vee \sim$ max (or)
$\wedge \sim$ min (and)

- Properties:

$$R: X \text{ x } Y$$
$$S: Y \text{ x } Z$$
$$T: Z \text{ x } W$$

1) Associativity

$$R \circ (S \circ T) = (R \circ S) \circ T$$

2) Distributivity

$$R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$$

3) Weak distributivity over intersection

$$R \circ (S \sqcap T) \sqsubseteq (R \circ S) \sqcap (R \circ T)$$

4) Monotonicity

$$S \sqsubseteq T \rightarrow R \circ S \sqsubseteq R \circ T$$

$T - \text{norm} \sim \text{min product}$
$S - \text{norm} \sim \text{max product}$

2) Max-Product Composition

$R_1 \sim X\text{x}Y$
$R_2 \sim Y\text{x}Z$
$\mu_{R_1 \circ R_2}(x, z) = \max_y [\mu_{R_1}(x, y) * \mu_{R_2}(y, z)]$

Example 2-7

Let:

$\mathcal{R}_1 =$ "x is relevant to y"

$\mathcal{R}_2 =$ "y is relevant to z"

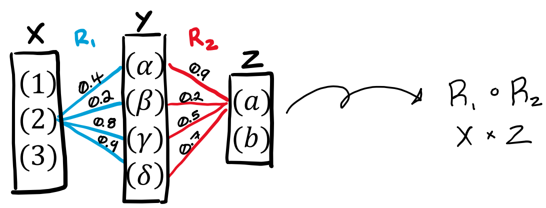
be two fuzzy relationships defined on X x Y and Y x Z, respectively, where $X = \{1, 2, 3\}$, $Y = \{\alpha, \beta, \gamma, \delta\}$, and $Z = \{a, b\}$. Assume that $\mathcal{R}_1$ and $\mathcal{R}_2$ can be expressed as the following matrices:

$$\mathcal{R}_1 = \begin{bmatrix} 0.1 & 0.3 & 0.5 & 0.7 \\ 0.4 & 0.2 & 0.8 & 0.9 \\ 0.6 & 0.8 & 0.3 & 0.2 \end{bmatrix} \quad X \times Y$$

$$\mathcal{R}_2 = \begin{bmatrix} 0.9 & 0{,}1 \\ 0.2 & 0.3 \\ 0.5 & 0.6 \\ 0.7 & 0.2 \end{bmatrix} \quad Y \times Z$$

Now, we want to find $\mathcal{R}_1 \circ \mathcal{R}_2$ which can be interpreted as a derived fuzzy relation "x is relevant to z" based on $\mathcal{R}_1$ and $\mathcal{R}_2$. For simplicity, suppose that we are only interested in the degree of relevance between $2(\in X)$ and $a(\in Z)$. If we adopt max min composition, then:

Solution



$X = \{1, 2, 3\}$

$Y = \{\alpha, \beta, \nu, \delta\}$

$Z = \{a, b\}$

1) Max-min composition operator

$$\mu_{R_1 \circ R_2}(x, z) \to \mu_{R_1 \circ R_2}(2, a)$$
$$= \max_y \min[\mu_{R_1}(x, y),\ \ \mu_{R_2}(y, z)]$$
$$= \max_y [0.4 \wedge 0.9,\ \ 0.2 \wedge 0.2,\ \ 0.8 \wedge 0.5,\ \ 0.9 \wedge 0.7\,)]$$
$$= \max_y [0.4,\ \ 0.2,\ \ 0.5,\ \ 0.7\,)]$$
$$= 0.7$$

2) Max-product composition operator

$$\mu_{R_1 \circ R_2}(x, z) \to \mu_{R_1 \circ R_2}(2, a)$$
$$= \max[0.4 * 0.9,\ \ 0.2 * 0.2,\ \ 0.8 * 0.5,\ \ 0.9 * 0.7]$$
$$= \max[0.36,\ \ 0.04,\ \ 0.14,\ \ 0.63]$$
$$= 0.63$$

2.7 Fuzzy IF-THEN Rules

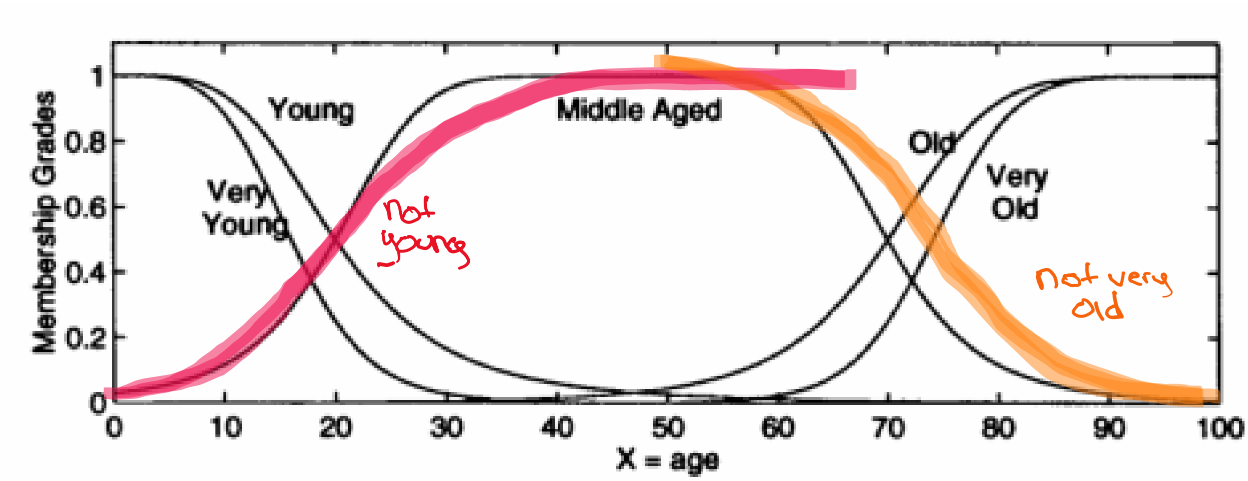
1) Linguistic Variables

$$\{\text{fuzzy set, universe, syntactic rule, semantic rule}\}$$

$$age \sim \text{linguistic variable}$$
$$set\ T\ (age)$$

$$T\ (age) = \left\{ \begin{matrix} young; \\ not\ young; \\ very\ young; \\ middle\ aged; \\ very\ old; \\ not\ very\ old; \\ more\ or\ less\ old \end{matrix} \right\}$$

$$X = [0, 100]$$



- **Primary terms**: young, middle aged, old
- **Negation**: not
- **Hedges**: very, quite, more or less
- **Connectives**: and, or, either, neither
- **Concentration** and **dilation**

Example:
$A \sim$ linguistic term
$MF: \mu_A(x)$
$A^k \sim$ modified version of the linguistic value
$A^k \sim \int \mu_A^k(x)/x$

- **Concentration**
  $CON(A) = A^2$
- **Dilation**
  $DIL(A) = \sqrt{A}$

- **Not**

$$NOT(A) = \neg A = \frac{\int [1 - \mu_A(x)]}{x}$$

Consider two terms A, B:

$$A \text{ AND } B = A \cap B = \frac{\int \mu_A(x) \wedge \mu_B(x)}{x}$$

$$A \text{ OR } B = A \cup B = \frac{\int \mu_A(x) \vee \mu_B(x)}{x}$$

Example:

$\mathrm{T}(age)$

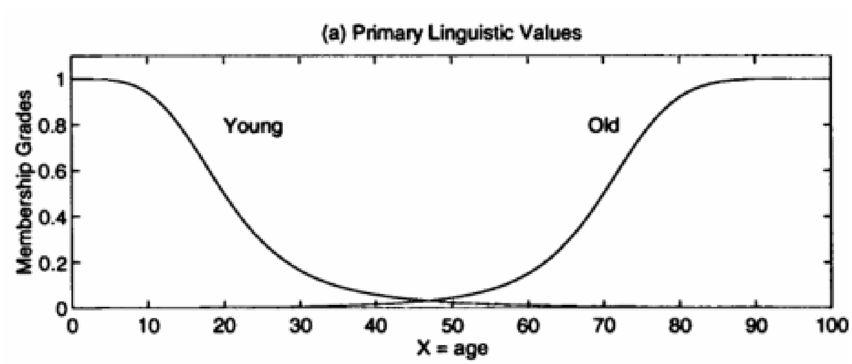
$\mu_{young}(x) = \mathrm{bell}(x, 20, 2, 0)$

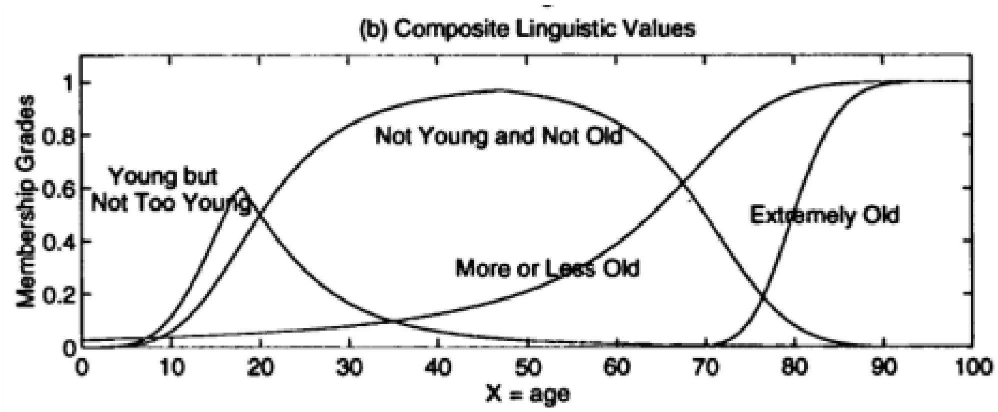
$$= \frac{1}{1 + \left(\frac{x}{20}\right)^4}$$

$\mu_{old}(x) = \mathrm{bell}(x, 30, 3, 100)$

$$= \frac{1}{1 + \left(\frac{x - 100}{30}\right)^6}$$

For $x = [0, 100]$:





- **More or less**

$DIL(old) = old^{0.5}$

$$= \frac{\int \sqrt{\frac{1}{1+\left(\frac{x-100}{30}\right)^6}}}{x}$$

- **Not young AND not old**

$= (\neg\ \text{young}) \sqcap (\neg\ \text{old})$

$$= \frac{\int \left[1 - \frac{1}{1+\left(\frac{x}{20}\right)^4}\right] \wedge \left[1 - \frac{1}{1+\left(\frac{x-100}{30}\right)^6}\right]}{x}$$

- **Young but not very (too) young**

$= \text{young} \sqcap (\neg\ \text{young}^2)$

$$= \frac{\int \left[\frac{1}{1+\left(\frac{x}{20}\right)^4}\right] \wedge \left[1 - \left(\frac{1}{1+\left(\frac{x}{20}\right)^4}\right)^2\right]}{x}$$

- **Extremely old**

$= \text{con}\left(\text{con}\left(\text{con}(\text{old})\right)\right)$

$((old^2)^2)^2 = old^8$

$$= \frac{\int \left(\frac{1}{1+\left(\frac{x-100}{30}\right)^6}\right)^8}{x}$$

2) Orthogonality

$$T = \{t_1, t_2, \dots, t_n\}$$

Universe X

$$\mu_{t_1}(x) + \mu_{t_2}(x) + \cdots + \mu_{t_n}(x) = 1$$
$$\sim \text{orthogonal}$$

2.9 Fuzzy IF-THEN Rules

Fuzzy implication

"If x is A then y is B"

Premise / condition

Conclusion / consequence

"$A \rightarrow B$"

A and B ~ linguistic values

X and Y ~ universe

$R = A \rightarrow B$

- A coupled with B

$R = A \rightarrow B = A \times B$

$$= \int_{X \times Y} \mu_A(x) \tilde{*} \mu_B(y) \Big/ (x, y)$$

$\tilde{*}$ = T-norm operator

- Material implication (A entails B)

$R = A \rightarrow B = \neg A \sqcup B$

And:

$a = \mu_A(x)$

$b = \mu_B(y)$

1) A coupled with B

1. *Mamdani conjunction*

$$R_m = A \to B = A \text{ x } B = \int_{X \text{ x } Y} \mu_A(x) \wedge \mu_B(y) \big/ (x, y)$$

$$f_m(a, b) = a \wedge b$$

2. *Larson (product) implication*

$$R_p = A \text{ x } B = \int_{X \text{ x } Y} \mu_A(x) \cdot \mu_B(y) \big/ (x, y)$$

$$f_p = a \cdot b$$

3. *Bounded product operator*

$$R_{bp} = A \text{ x } B = \int_{X \text{ x } Y} \mu_A(x) \odot \mu_B(y) \big/ (x, y)$$

$$= \int 0 \vee [\mu_A(x) + \mu_B(y) - 1] \big/ (x, y)$$

$$f_{bp} = 0 \vee [a + b - 1]$$

4. *Drastic product operator*

$$R_{dp} = A \text{ x } B = \int_{X \text{ x } Y} \mu_A(x) \hat{\cdot} \mu_B(y) \big/ (x, y)$$

$$f_{dp}(a, b) = a \hat{\cdot} b = \begin{cases} a & ; \quad b = 1 \\ b & ; \quad a = 1 \\ 0 & ; \quad \text{Otherwise} \end{cases}$$

Consider:

$a = \mu_A(x) = \text{bell}(x, 4, 3, 10)$

$b = \mu_B(y) = \text{bell}(y, 4, 3, 10)$

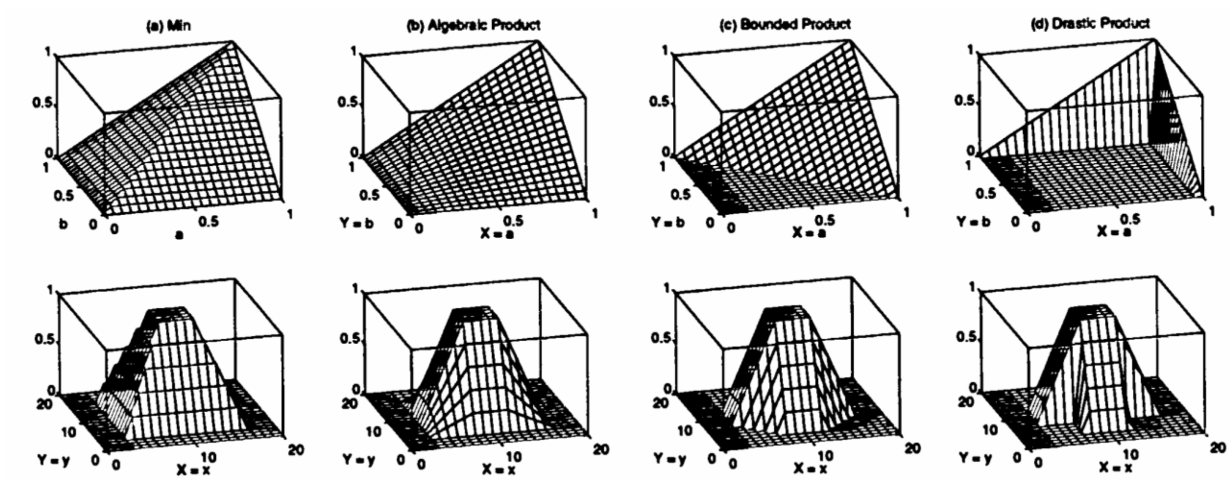


Figure 2.16. *(First row) Four T-norm operators $T_{min}(a,b)$, $T_{ap}(a,b)$, $T_{bp}(a,b)$, and $T_{dp}(a,b)$; (second row) the corresponding surfaces for a = trapezoid$(x,3,8,12,17)$ and b = trapezoid$(y,3,8,12,17)$.* (MATLAB file: `tnorm.m`)

2) A entails B

1. Zadeh's arithmetic rule

$$R_a = A \to B = \neg A \sqcup B$$
$$f_a(a, b) = 1 \wedge (1 - a + b)$$

2. Zadeh's max-min rule

$$R_{mm} = A \to B = \neg A \sqcup (A \sqcap B)$$
$$a = \mu_A(x)$$
$$b = \mu_B(x)$$
$$f_{mm}(a, b) = (1 - a) \vee (a \wedge b)$$

3. Boolean fuzzy implication

$$R_B = A \to B = \neg A \sqcup B$$
$$= \int_{X \times Y} [1 - \mu_A(x)] \vee \mu_B(y) \Big/ (x, y)$$
$$f_B(a, b) = (1 - a) \vee b$$

4. Gogen's fuzzy implication

$$R_\Delta = A \to B$$
$$= \int_{X \times Y} \mu_A(x) \mathrel{\tilde{<}} \mu_B(y) \Big/ (x, y)$$
$$f_\Delta(a, b) = a \mathrel{\tilde{<}} b = \begin{cases} 1 & ; \quad a \le b \\ b/a & ; \quad a > b \end{cases}$$

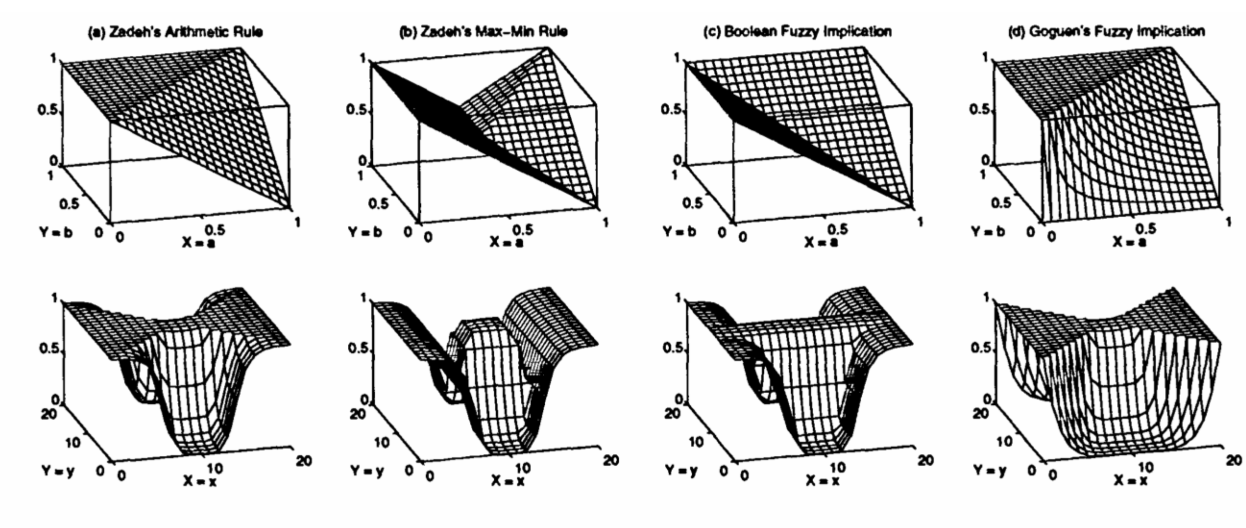


Figure 3.9. *First row: fuzzy implication functions based on the interpretation "A entails B"; second row: the corresponding fuzzy relations.* (MATLAB file: `fuzimp.m`)

2.10 Fuzzy Reasoning Rulebase

2-valued logic, modus ponens

Something like:
fact $\sim x$ is A'
premise (rule) $=$ if x is A then y is B
consequent conclusion $\sim y$ is B'
→ called approximate reasoning
→ or generalised modus ponens (GMP)

Let A, B be fuzzy sets
of X and Y, $A' \sim$ of X'

Rule – fuzzy implication
$R = A \rightarrow B \;\; ; \;\; X \text{ x } Y$

$$\mu_B'(y) = \max\min[\mu_A'(x), \;\; \mu_R(x,y)]$$

$$= \vee_x \, [\mu_A(x) \wedge \mu_B(x,y)]$$

or

$$B' = A' \circ R = A' \circ (A \rightarrow B)$$

" $\circ$ " $=$ composition operator

1) Single rule with single antecedent

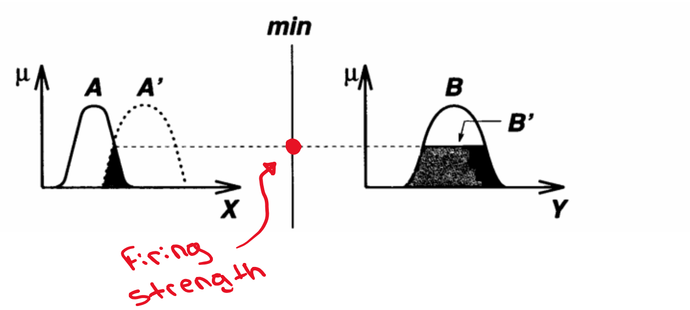
Premise 1 (fact):	x is A'
Premise 2 (rule):	If x is A then y is B
Consequence (conclusion):	y is B'

$$\mu_B'(y) = \vee_x \, [\mu_A'(x) \wedge \mu_R(x,y)]$$
$$A \rightarrow B = A \wedge B$$
$$= \vee_x \, \left[\mu_A'(x) \wedge [\mu_A(x) \wedge \mu_B(y)]\right]$$
$$= \vee_x \, \left[\underbrace{[\mu_A'(x) \wedge \mu_A(x)]}_{\text{degree of match}} \wedge \mu_B(y)\right]$$

degree of match $= \omega$ (intersection)

$$\mu_B'(y) = \vee_x \, [\omega \wedge \mu_B(y)]$$

$$\mu_B'(y) = \underset{\max}{\vee_x} \, \underbrace{[\mu_A'(x) \wedge \mu_A(x)]}_{\omega} \wedge \mu_B(y)$$
$$= \omega \wedge \mu_B(y)$$



2) Single rule with multiple antecedents

antecedent ~ something existing before (or logically proceeding) another.

Premise 1 (fact):	x is A' and y is B'
Premise 2 (rule):	If x is A and y is B then z is C
Consequence (conclusion):	z is C$'$

$$R = A \text{ x } B \rightarrow C$$

and

Mamdani's implication:

$$R_m(A, B, C) = A \text{ x } B \rightarrow C$$
$$= \int \mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z) \Big/ (x, y, z)$$

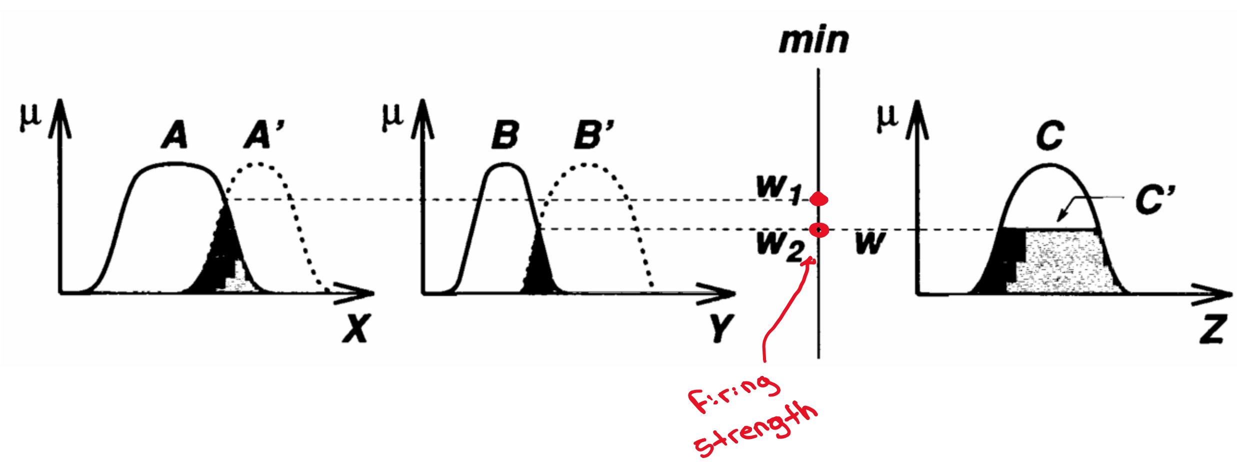
A'x B$'$; $C' = ?$

$$C' = (A' \text{ x } B') \text{ x } R_m$$
$$= (A' \text{ x } B') \cdot (A \text{ x } B \rightarrow C)$$
$$= (A' \text{ x } B') \wedge (A \text{ x } B \text{ x } C)$$

$$\mu'_C(z) = max - min$$
$$= \vee_{x,y} \{[\mu'_A(x) \wedge \mu'_B(y)] \wedge [\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)]\}$$
$$= \vee_{x,y} \{[\mu'_A(x) \wedge \mu_A(x)] \wedge [\mu'_B(y) \wedge \mu_B(y)]\} \wedge \mu_C(z)$$

$$= \underbrace{\vee_x [\mu'_A(x) \wedge \mu_A(x)]}_{\omega_1} \wedge \underbrace{\vee_y [\mu'_B(y) \wedge \mu_B(y)]}_{\omega_2} \wedge \mu_C(z)$$

$$= \omega_1 \wedge \omega_2 \wedge \mu_C(z)$$

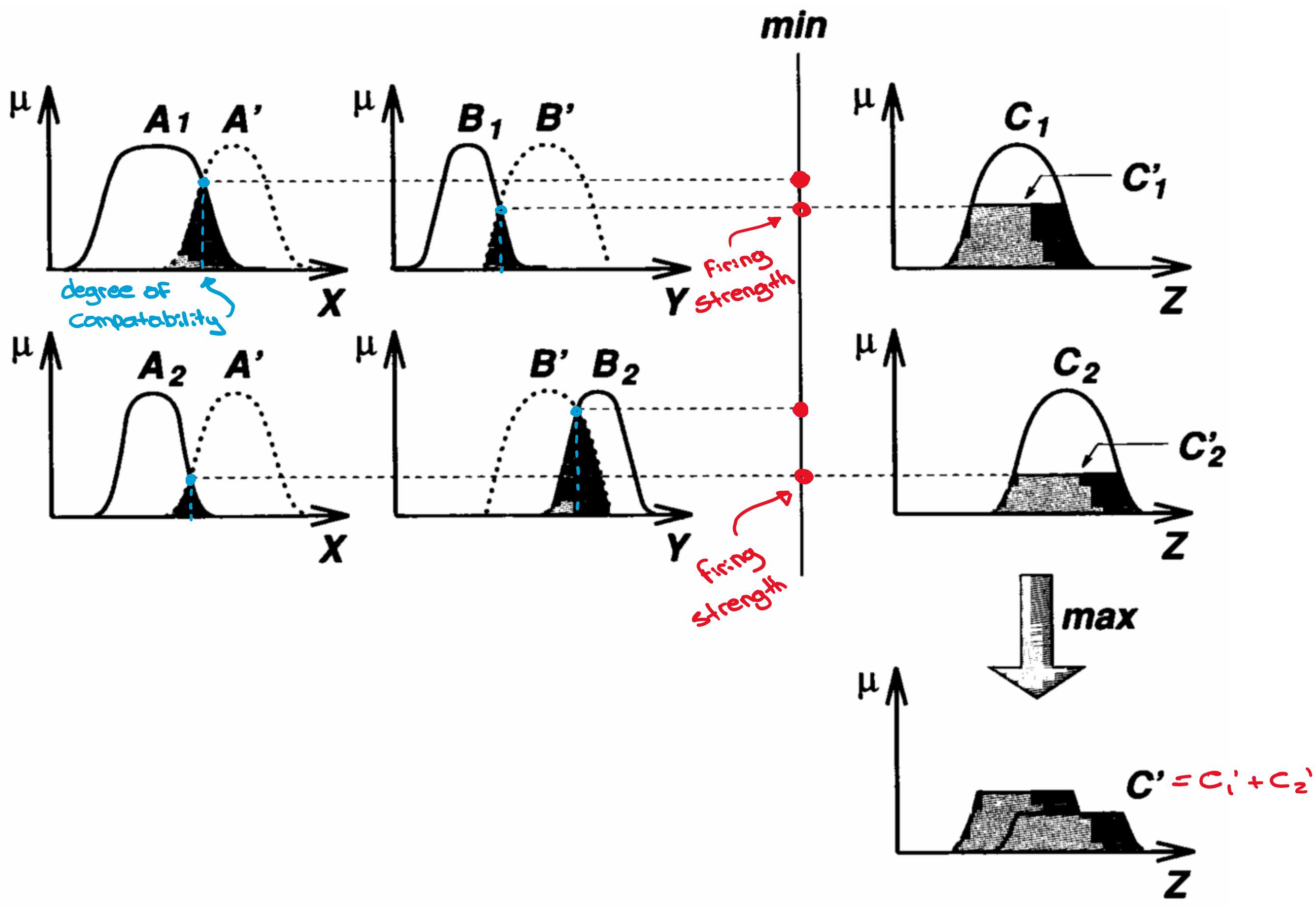


3) Multiple rules with multiple antecedents

Premise 1 (fact):	x is A' and y is B'
Premise 2 (rule 1):	If x is A_1 and y is B_1 then z is C_1
Premise 3 (rule 2):	If x is A_2 and y is B_2 then z is C_2
Consequence (conclusion):	z is C'

Rule 1: $R_1 = A_1 \times B_1 \rightarrow C_1$
Rule 2: $R_2 = A_2 \times B_2 \rightarrow C_2$
Fact: $A' \times B'$

Use max min composition operator " $\circ$ "
$C' = (A' \times B') \circ (R_1 \cup R_2)$



$C' = (A' \times B') \wedge (R_1 \sqcup R_2)$
$= \underbrace{[(A' \times B') \wedge R_1]}_{C_1'} \sqcup \underbrace{[(A' \times B') \wedge R_2]}_{C_2'}$

$= C_1' \sqcup C_2'$

Theorem 2.1 Decomposition Method

$R \to (A \text{ x } B \to C)$

Given fact: $A' \text{ x B}'$
$C' = (A' \text{x } B') \cdot (A \text{ x } B \to C)$
$= [\underbrace{A' \cdot (A \to C)}_{C_1'}] \sqcap [\underbrace{B' \cdot (B \to C)}_{C_2'}]$

$= C_1' \sqcap C_2'$

Proof:
$\mu_{C'}(z) = \vee_{x,y} \{[\mu_{A'}(x) \wedge \mu_{B'}(y)] \wedge [\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)]\}$
$= \vee_x [\mu_{A'}(x) \wedge \mu_A(x) \wedge \mu_C(z)] \wedge \vee_y [\mu_{B'}(y) \wedge \mu_B(y) \wedge \mu_C(z)]$
$= \mu_{A' \circ (A \to C)} \wedge \mu_{B' \circ (B \to C)}$
$= C_1' \wedge C_2'$

In Summary

Degree of compatibility Compare the known facts with the antecedents of fuzzy rules to find the degrees of compatibility with respect to each antecedent MF.

Firing strength Combine degrees of compatibility with respect to antecedent MFs in a rule using fuzzy AND or OR operators to form a firing strength that indicates the degree to which the antecedent part of the rule is satisfied.

Qualified (induced) consequent MFs Apply the firing strength to the consequent MF of a rule to generate a qualified consequent MF. (The qualified consequent MFs represent how the firing strength gets propagated and used in a fuzzy implication statement.)

Overall output MF aggregates all the qualified consequent MFs to obtain an overall MF.

Chapter 3: Fuzzy Inference Systems

neuro fuzzy system ~ fuzzy system (the main difference is related to parameter training)

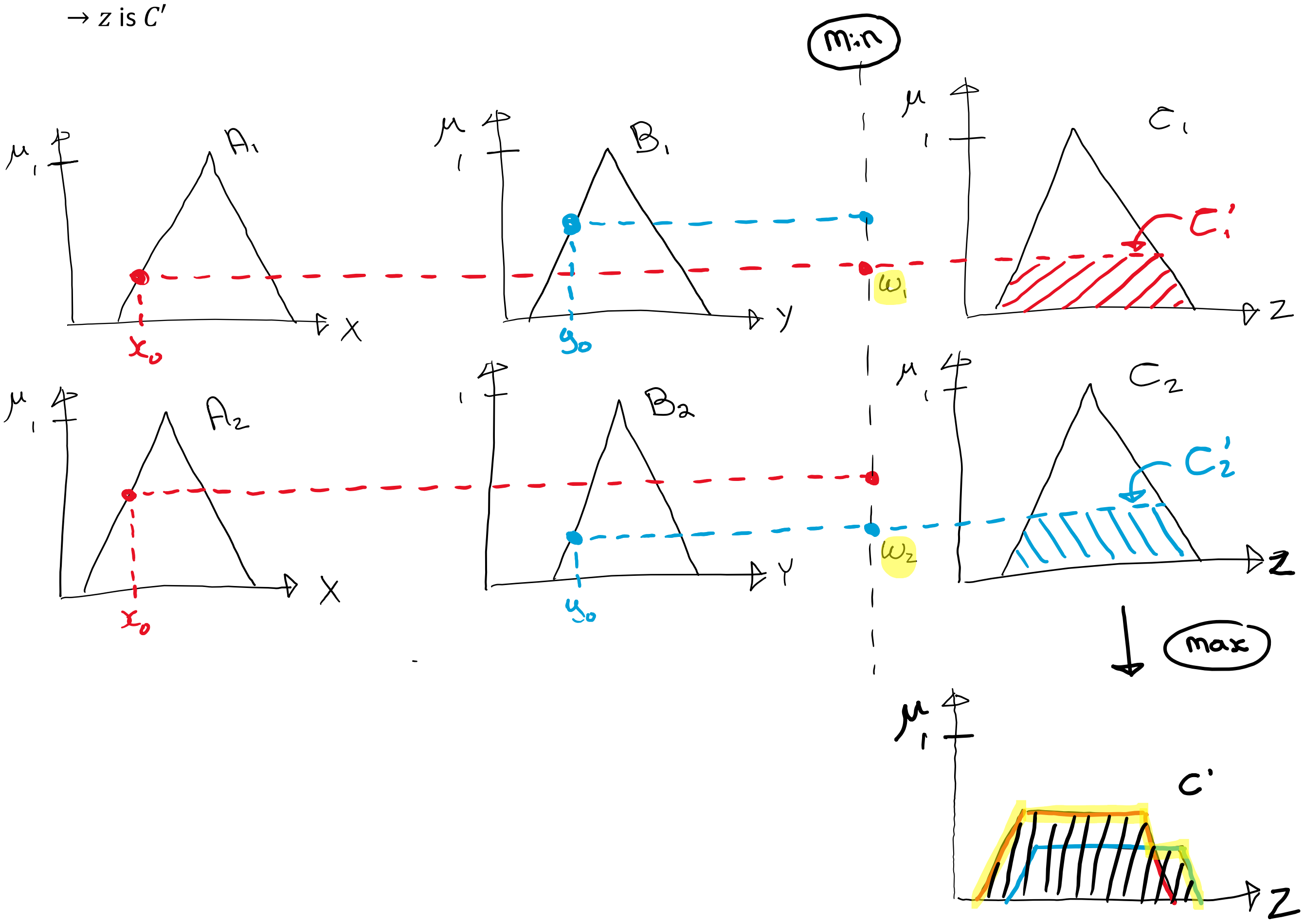
Inputs: fuzzy inputs, crisp inputs (fuzzy singletons)

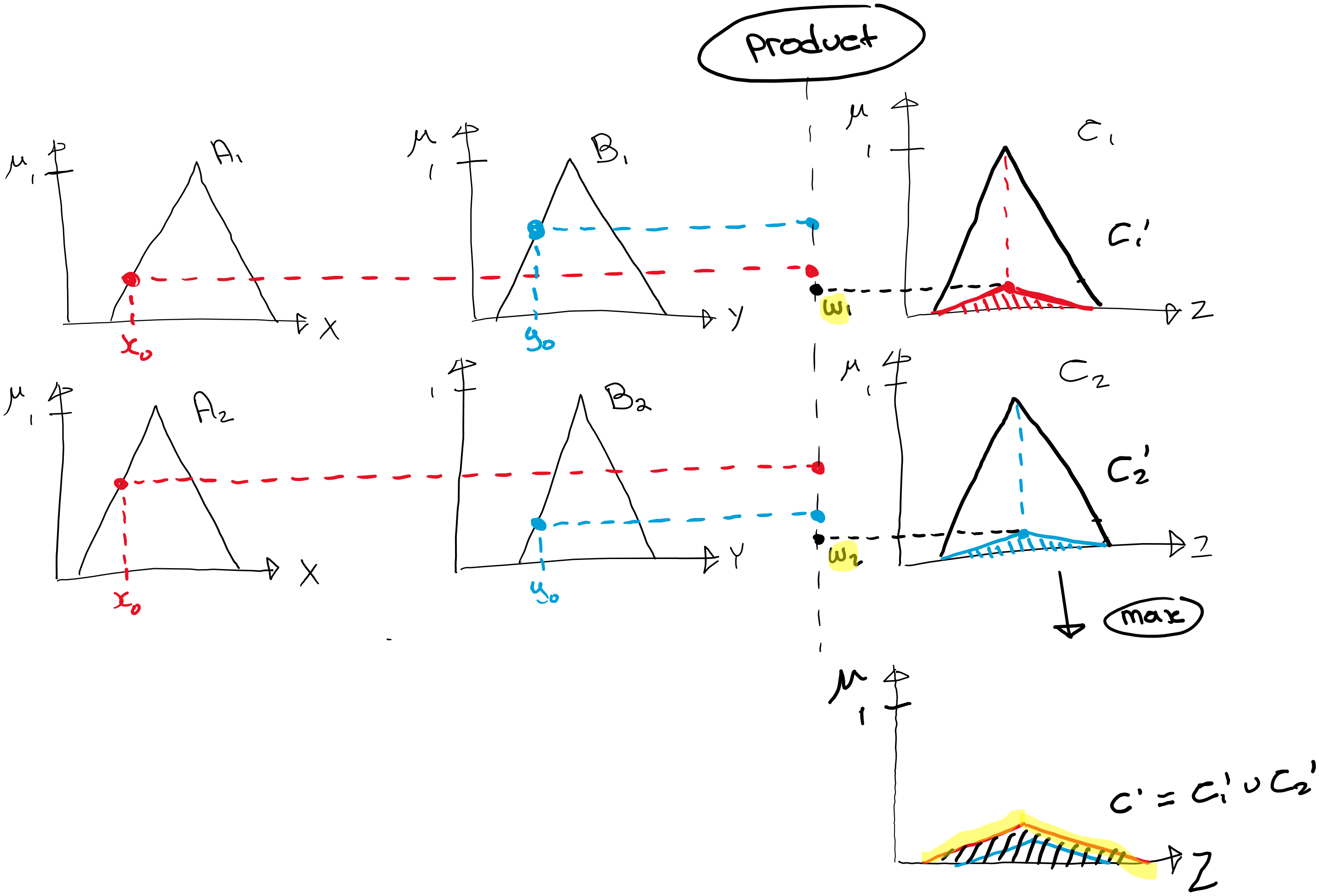
1) Mamdani Fuzzy Models

Rule: (R_1)
If (x is A_1) and (y is B_1)
Then (z is C_1)

Rule: (R_2)
If (x is A_2) and (y is B_2)
Then (z is C_2)

$x = x_0$
$y = y_0$
$z = ?$
$\rightarrow z$ is C'



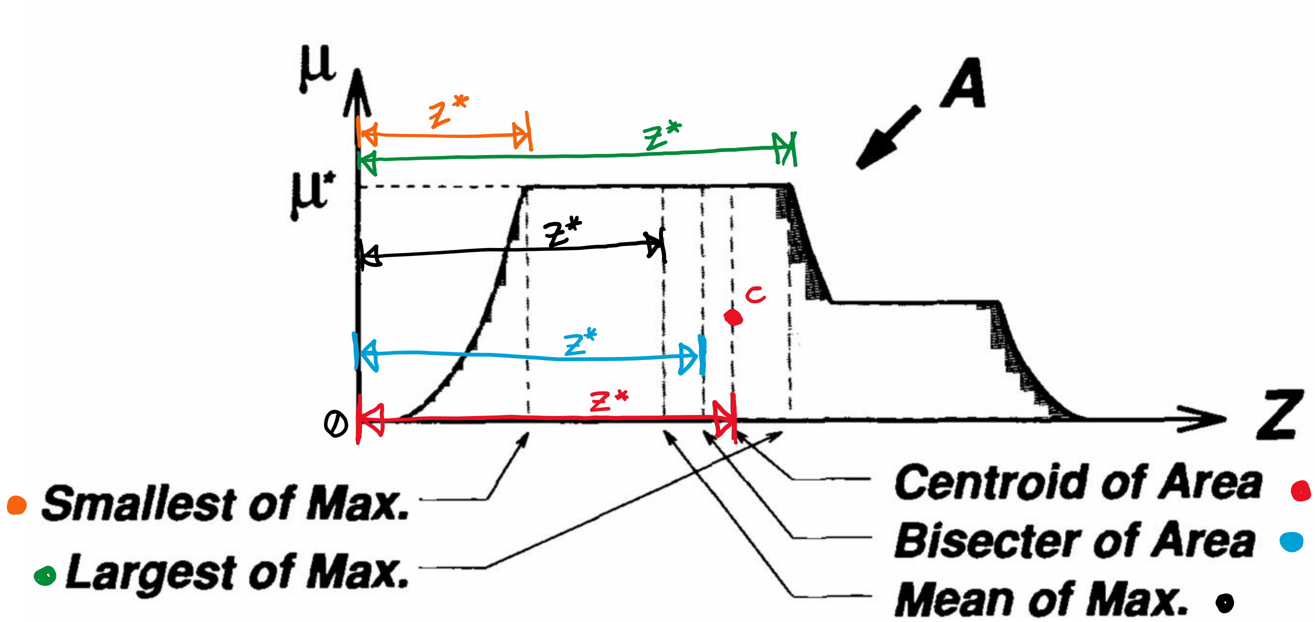


Product
A1
B1
C1
C1'
x0
y0
w1
A2
B2
C2
C2'
x0
y0
w2
max
C' = C1' ∪ C2'

2) Defuzzification

When we want to get back a number, instead of a membership function.

- Centroid of the area (most commonly used, dividing line drawn across the centroid of the MF)
- Bisector of the area (commonly used, dividing line such that area on LHS = RHS)
- Smallest of the maximum
- Largest of the maximum
- Mean of the maximum

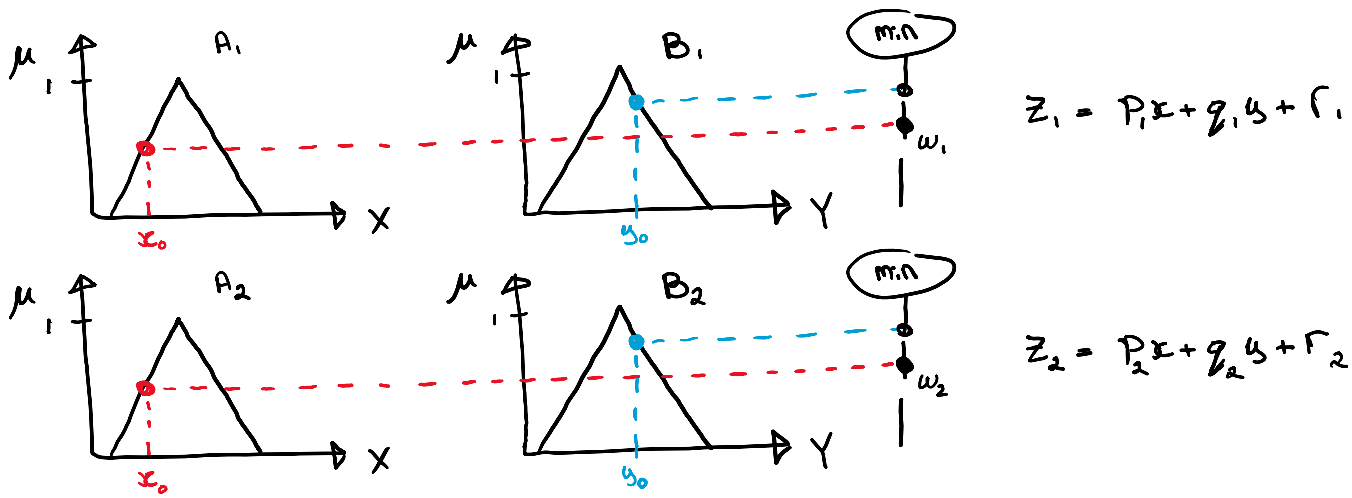


1) Mamdani fuzzy model

$$\frac{\text{max} - \text{min/ product}}{R_1 \cup R_2 \cup \ldots}$$

2) Sugeno fuzzy models

Takagi-Sugeno-Kang (TSK): consequent part of a rule is a polynomial function of inputs.



Defuzzification:

$$z^* = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_2}$$
$$z^* = \frac{w_1(p_1 x + q_1 y + r_1) + w_2(p_2 x + q_2 y + r_2)}{w_1 + w_2}$$

Type 1: TSK model (1st order)

$$z_1 = p_1 x^1 + q_1 y^1 + r_1$$
$$z_1 = p_1 x + q_1 y + r_1$$

Type 0: (or 0^{th} order TSK)

$$z_1 = p_1 x^0 + q_1 y^0 + r_1$$
$$z_1 = p_1 + q_1 + r_1$$
(consequent part is just a number)
$$z_1 = C_1$$

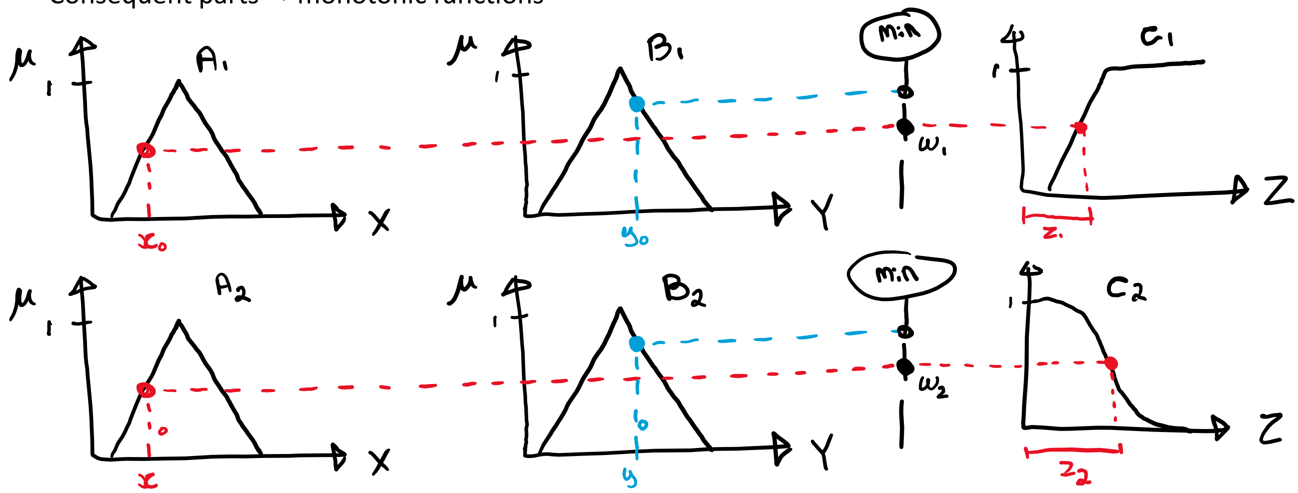
1st order TSK and models are commonly used in modeling (forecasting) applications.

Neuro fuzzy models (NF) are fuzzy model – but they are different from conventional fuzzy systems. They can use machine learning algorithms to update parameters.

3) Tsukamoto fuzzy models

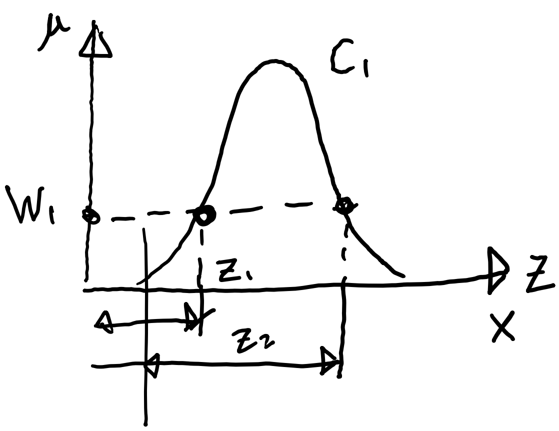
Premise parts → same

Consequent parts → monotonic functions



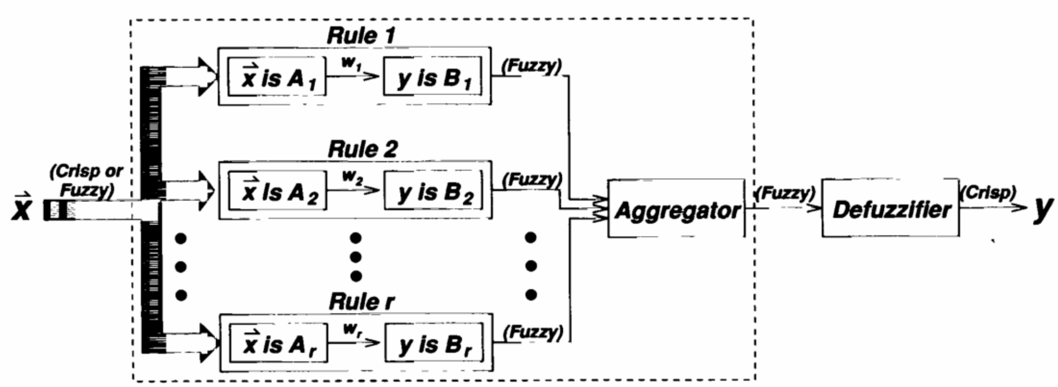
Defuzzification (output):

$$z^* = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_1}$$



(This is why we use monotonic functions - otherwise there is two different results at a single firing strength)

(Read by yourself for more info – Section 4.5, Book 2)



Example 3.1

Consider the room comfort control system schematically shown in Figure 3.3. The temperature (T) and humidity (H) are the process variables that are measured. These sensor signals are provided to the fuzzy logic controller, which determines the cooling rate (C) that should be generated by the air conditioning unit. The objective is to maintain a particular comfort level inside the room.

A simplified fuzzy rule base of the comfort controller is graphically presented in Figure 3.4. The temperature level can assume one of two fuzzy states (*HG*, *LW*), which denote high and low, respectively, with the corresponding membership functions. Similarly, the humidity level can assume two other fuzzy states (*HG*, *LW*) with associated membership functions. Note that the membership functions of T are quite different from those of H, even though the same nomenclature is used. There are four rules, as given in Figure 3.4. The rule base is:

Rule 1:		If	*T*	is	*HG*	and	*H*	is	*HG*	then	*C*	is	*PH*
Rule 2:	else	if	*T*	is	*HG*	and	*H*	is	*LW*	then	*C*	is	*PL*
Rule 3:	else	if	*T*	is	*LW*	and	*H*	is	*HG*	then	*C*	is	*NL*
Rule 4:	else	if	*T*	is	*LW*	and	*H*	is	*LW*	then	*C*	is	*NH*
	end	if											

The nomenclature used for the fuzzy states is as follows:

Temperature (T)	*Humidity* (H)	*Change in cooling rate* (C)
HG = High	*HG* = High	*PH* = Positive high
LW = Low	*LW* = Low	*PL* = Positive low
		NH = Negative high
		NL = Negative low

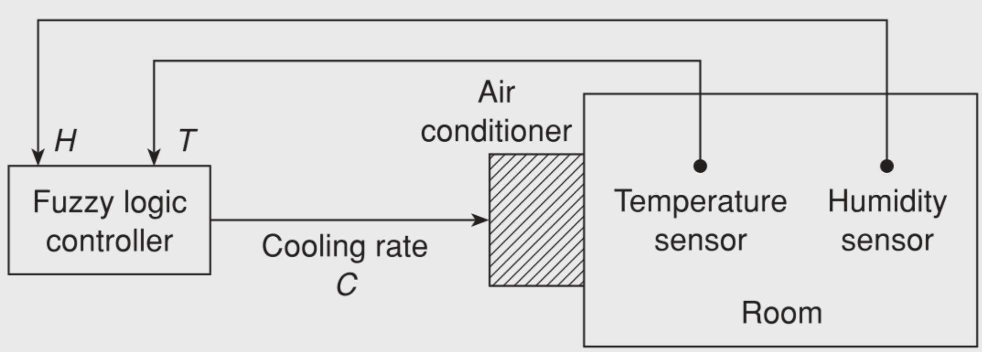
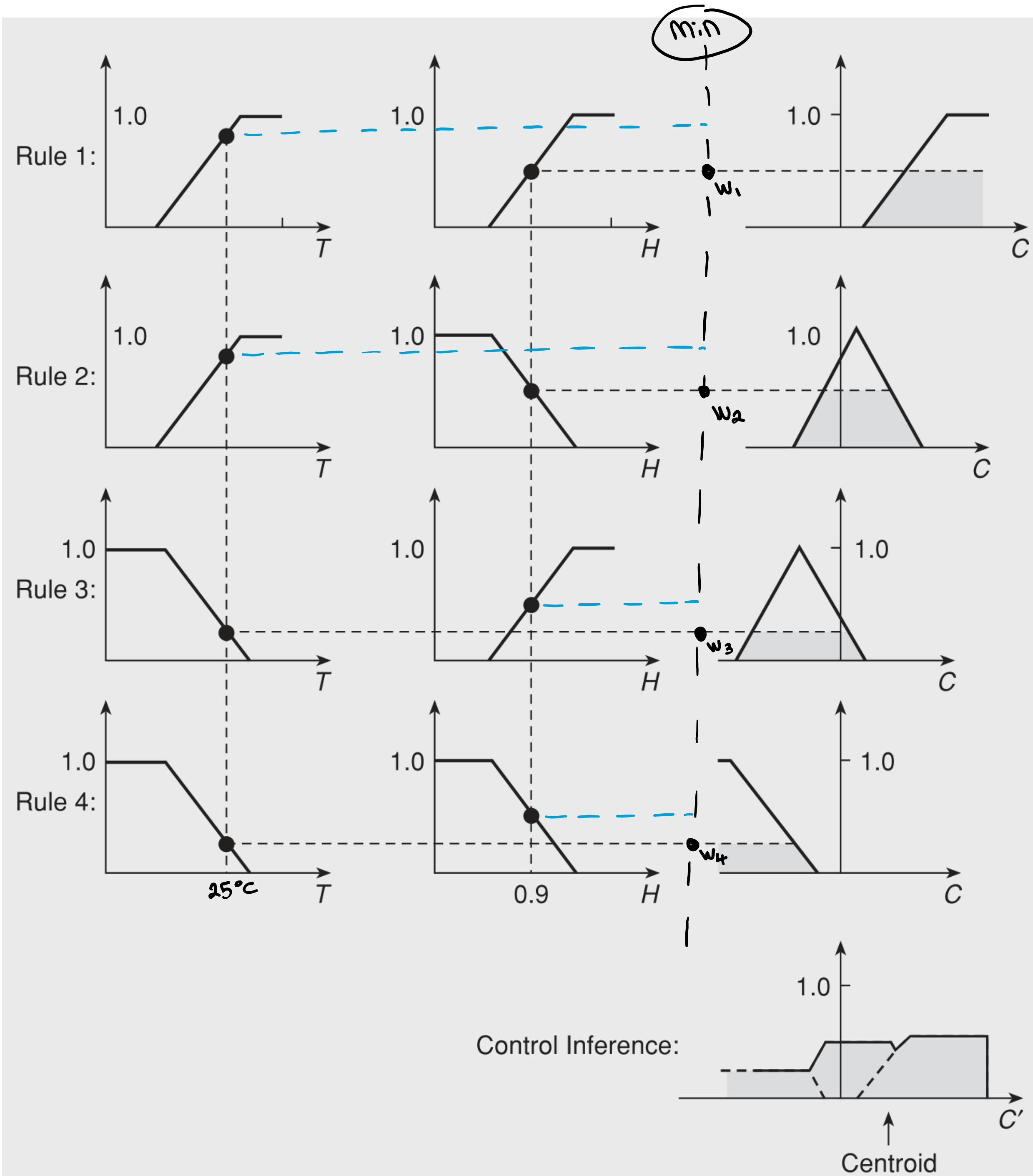


Figure 3.3: Comfort control system of a room



min
Rule 1:
1.0
T
1.0
H
w1
1.0
C
Rule 2:
1.0
T
1.0
H
w2
1.0
C
Rule 3:
1.0
T
1.0
H
w3
1.0
C
Rule 4:
1.0
T
25°C
1.0
H
0.9
w4
1.0
C
Control Inference:
1.0
C′
Centroid

Example 3.2

A schematic diagram of a simplified system for controlling the liquid level in a tank is shown in Figure 3.8(a). In the control system, the error (actually, correction) is given by

$$e = \text{Desired level} - \text{Actual level}.$$

The change in error is denoted by Δe. The control action is denoted by u, where $u > 0$ corresponds to opening the inflow valve and $u < 0$ corresponds to opening the outflow valve. A low-level direct fuzzy controller is used in this control system, with the control rule base as given in Figure 3.8(b).

The membership functions for E, ΔE, and U are given in Figure 3.8(c). Note that the error measurements are limited to the interval $[-3a, 3a]$ and the Δerror measurements to $[-3b, 3b]$. The control actions are in the range $[-4c, 4c]$.

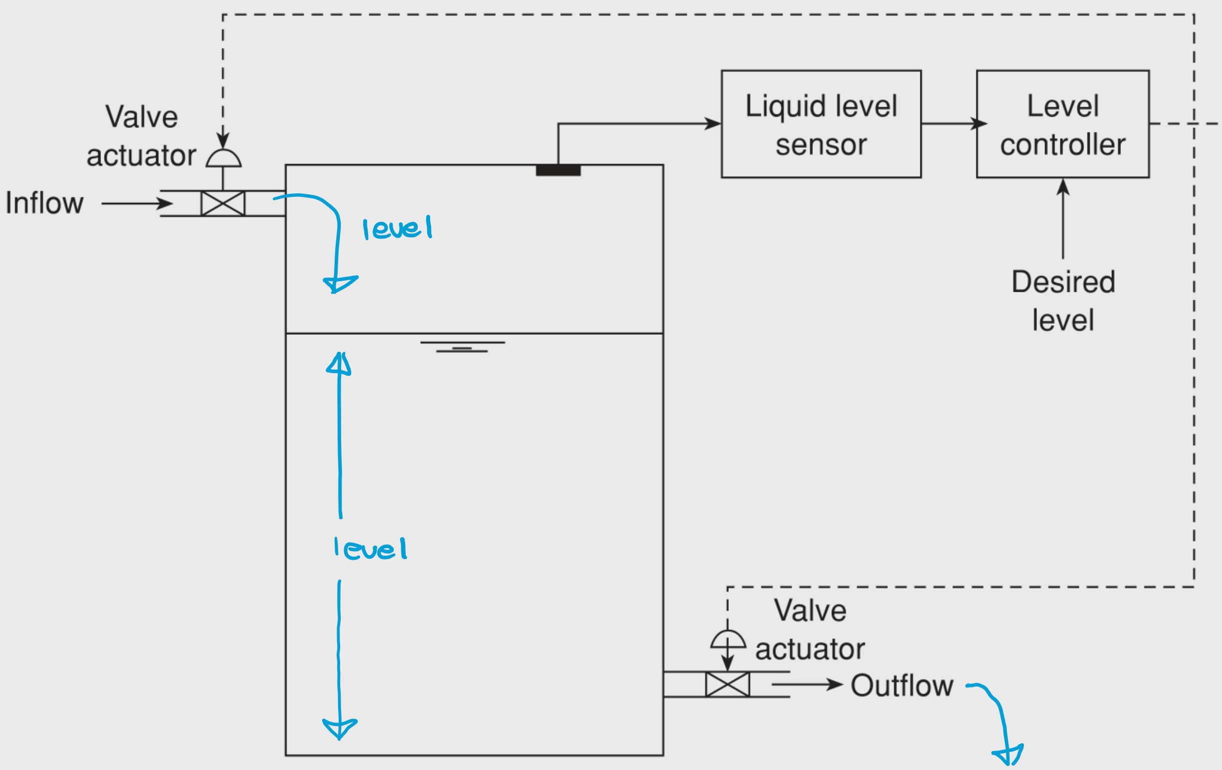


Figure 3.8 (a): Liquid level control system

ΔE / E	NL	NS	ZO	PS	PL
NL	NL	NL	NM	NS	ZO
NS	NL	NM	NS	ZO	PS
ZO	NM	NS	ZO	PS	PM
PS	NS	ZO	PS	PM	PL
PL	ZO	PS	PM	PL	PL

Figure 3.8 (b): The control rule base

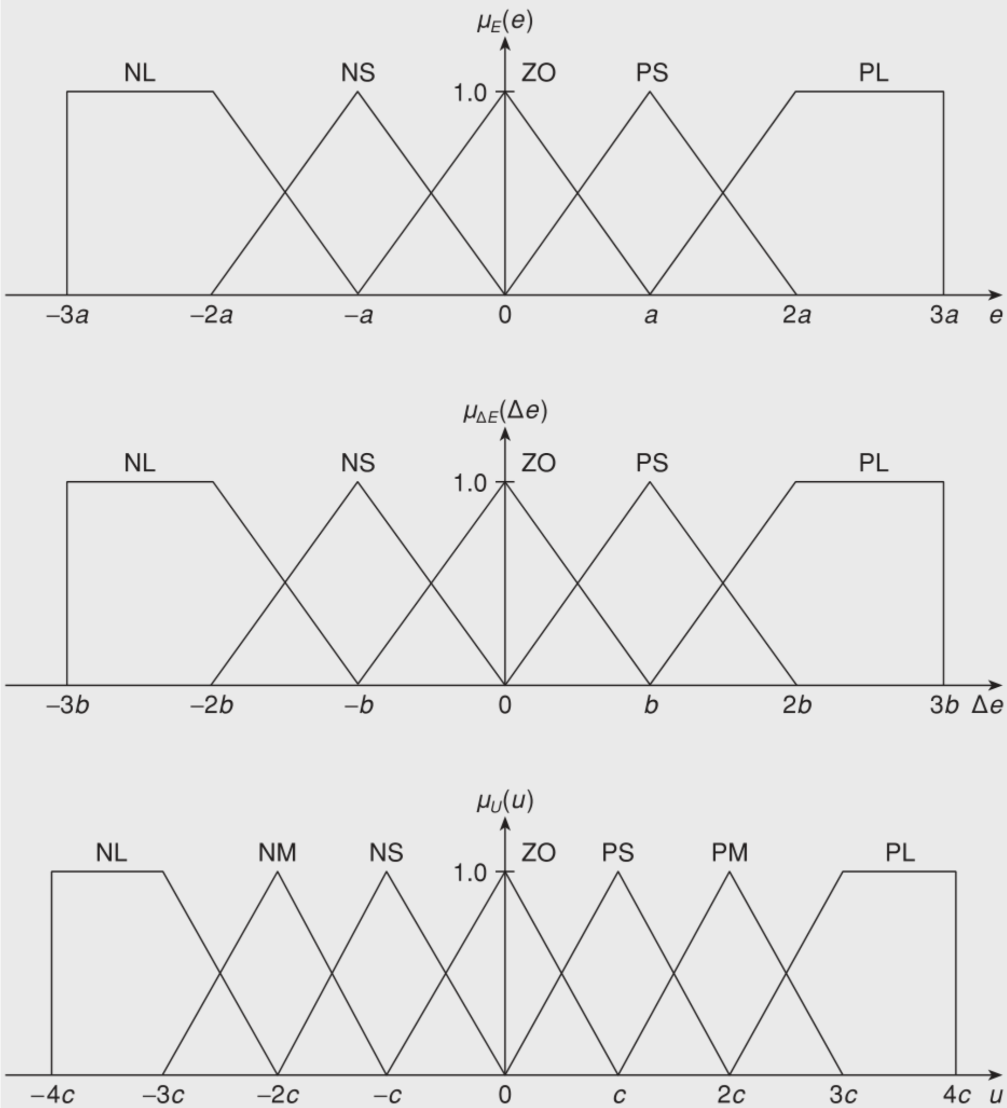


Figure 3.8 (c): The membership functions of error, change in error, and control action

Chapter 4: System Training

The difference between a fuzzy system and a neuro fuzzy system is that we can implement the fuzzy system like a neural network, then we can train system parameters.

We can use machine learning or training algorithms to optimize membership function parameters. This includes the TSK model (the consequent part parameters) and system reasoning structures. Parameters can be linear or nonlinear.

Linear: $z = 3x + 5y + 2$
Non-linear: $z^* = 2x^2 + 3y^3 + x + 2$

4.1 Least Squares Estimator (LSE)

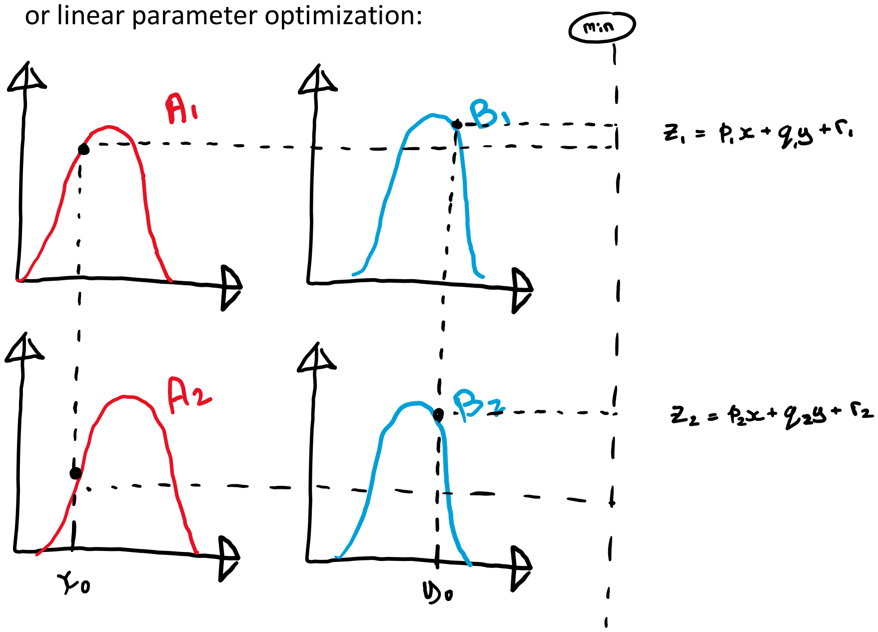
For linear parameter optimization:

$$y = \theta_1 f(\vec{u}_1) + \theta_2 f(\vec{u}_2) \dots \theta_n f(\vec{u}_n)$$

$\text{Parameters} = \{\theta_1 \quad \theta_2 \quad \dots \quad \theta_n\}$
$\text{Output} = y$
$\text{Input vectors} = \vec{\mathrm{u}}_1, \vec{\mathrm{u}}_2, \dots, \vec{\mathrm{u}}_\mathrm{n}$
(Because $\vec{\mathrm{u}} = \{\vec{\mathrm{u}}_1 \quad \vec{\mathrm{u}}_2 \quad \dots \quad \vec{\mathrm{u}}_\mathrm{n}\}$)

4.1 Least Squares Estimator

or linear parameter optimization:



$$z^* = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_2}$$

$$z^* = \frac{w_1(p_1 x + q_1 y + r_1) + w_2(p_2 x + q_2 y + r_2)}{w_1 + w_2}$$

Linear parameters: $p_1, q_1, r_1, p_2, q_2, r_2$

$$\mu_{A_2} = e^{-\frac{(x-a)^2}{b^2}} \quad ; \quad w_2 = e^{-\left(\frac{x_0 - a}{b}\right)^2}$$

Nonlinear: MF (membership function) parameters

$$y = \theta_1 f_1(\vec{u}) + \theta_2 f_2(\vec{u}) + \cdots + \theta_n f_n(\vec{u})$$

$$\vec{u} = \{x_1, x_2, \dots, x_n\}^T \quad \text{inputs}$$

$$\vec{\theta} = \{\theta_1, \theta_2, \dots, \theta_n\}^T \quad \text{unknown}$$

Linear parameters:

$$\{\vec{u}_1, y_1\}, \{\vec{u}_2, y_2\}, \dots, \{\vec{u}_m, y_m\}$$

General representation:

$$\{\vec{u}_i, y_i\} \quad ; \quad i = 1, 2, \dots, m$$

$$\begin{gathered} f_1(\vec{u}_1)\theta_1 + f_2(\vec{u}_1)\theta_2 + \cdots + f_n(\vec{u}_1)\theta_n = y_1 \\ f_1(\vec{u}_2)\theta_1 + f_2(\vec{u}_2)\theta_2 + \cdots + f_n(\vec{u}_2)\theta_n = y_2 \\ \vdots \\ f_1(\vec{u}_m)\theta_1 + f_2(\vec{u}_m)\theta_2 + \cdots + f_n(\vec{u}_m)\theta_n = y_m \end{gathered}$$

Matrix representation:

$$\begin{matrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_i^T \\ \vdots \\ \vec{a}_n^T \end{matrix} \underbrace{\begin{bmatrix} f_1(\vec{u}_1) & f_2(\vec{u}_1) & \cdots & f_n(\vec{u}_1) \\ f_1(\vec{u}_2) & f_2(\vec{u}_2) & \cdots & f_n(\vec{u}_2) \\ \vdots & \vdots & \cdots & \vdots \\ f_n(\vec{u}_i) & f_n(\vec{u}_i) & \cdots & f_n(\vec{u}_i) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\vec{u}_m) & f_2(\vec{u}_m) & \cdots & f_n(\vec{u}_m) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}}_{\vec{\theta}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}}_{\vec{y}}$$

$\vec{\theta}^T = \{\theta_1, \theta_2, \dots, \theta_n\}$

Summary:

- Vectors "→" (column representation, typically)
- Matrix $\underline{A}$
- Scalar

$$\underbrace{\vec{a}_i^T = \{f_1(\vec{u}_1), f_2(\vec{u}_2), \dots, f_n(\vec{u}_i)\}}_{\{\vec{u}_i;\, y_i\}}$$

$\underline{A}\,\vec{\theta} = \vec{y}$

If $\underline{A}$ is non-singular ($\det \neq 0$)

Identity Matrix → $\underline{A}^{-1}\underline{A}\,\vec{\theta} = \underline{A}^{-1}\,\vec{y}$

$\vec{\theta} = \underline{A}^{-1}\vec{y}$

$m \to n$

$m =$ # of training data points

$n =$ # of linear paramerers to be optimized

"In general, the training data points should be 5-times the number of linear data points to be optimized"

- Noise in experiments

Unavoidable (always present)

$$\underbrace{\underline{A}\,\vec{\theta}}_{\text{theoretical}} + \overset{\text{error}}{\vec{e}} = \underbrace{\vec{y}}_{\text{measured}}$$

Error vector:

$$\vec{e} = \vec{y} - \underline{A}\,\vec{\theta}$$

Objective function:

$$E(\vec{\theta}) = \left(y_1 - \vec{a}_1^T\vec{\theta}\right)^2 + \left(y_2 - \vec{a}_2^T\vec{\theta}\right)^2 + \cdots + \left(y_i - \vec{a}_i^T\vec{\theta}\right)^2 + \cdots + \left(y_m - \vec{a}_m^T\vec{\theta}\right)^2$$

$$E(\vec{\theta}) = \sum_{i=1}\left(y_i - \vec{a}_i^T\vec{\theta}\right)^2$$

$$\vec{e}_i = y_i - \vec{a}_i^T\vec{\theta} \quad ; \quad \text{Where } i = 1, 2, 3, \dots, m$$

$$E(\vec{\theta}) = \vec{e}_1^T\vec{e}_1 + \vec{e}_2^T\vec{e}_2 + \cdots + \vec{e}_i^T\vec{e}_i + \cdots + \vec{e}_m^T\vec{e}_m$$

$$E(\vec{\theta}) = \sum_{i=1}^{m} \vec{e}_i^T\vec{e}_i$$

Consider:

$\vec{e}^T\vec{e} = (\vec{y} - \underline{A}\vec{\theta})^T(\vec{y} - \underline{A}\vec{\theta})$

$= [\vec{y}^T - (\underline{A}\vec{\theta})^T](\vec{y} - \underline{A}\vec{\theta})$

$= [\vec{y}^T - \vec{\theta}^T\underline{A}^T](\vec{y} - \underline{A}\vec{\theta})$

$= \vec{y}^T\vec{y} - \vec{y}^T\underline{A}\vec{\theta} - \vec{\theta}^T\underline{A}^T\vec{y} + \vec{\theta}^T\underline{A}^T\underline{A}\vec{\theta}$

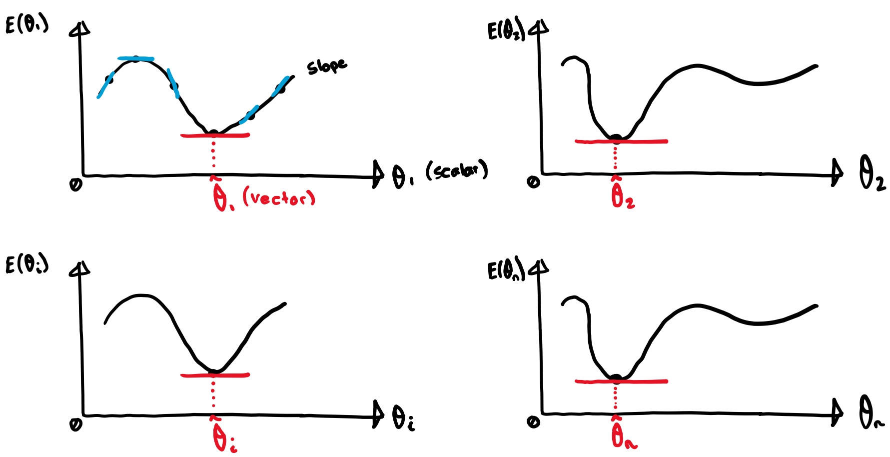
$\vec{y} = \underline{A}\vec{\theta}$

$\vec{y}^T = (\underline{A}\vec{\theta})^T$

$\vec{y}^T = \vec{\theta}^T\underline{A}^T$

$= \vec{y}^T\vec{y} - \vec{y}^T\underline{A}\vec{\theta} - \vec{y}^T\underline{A}\vec{\theta} + \vec{\theta}^T\underline{A}^T\underline{A}\vec{\theta}$

$= \vec{y}^T\vec{y} - 2\vec{y}^T\underline{A}\vec{\theta} + \vec{\theta}^T\underline{A}^T\underline{A}\vec{\theta}$



$$\vec{\theta} = \{\theta_1, \theta_2, \dots, \theta_n\}^T$$

$$\frac{\partial E(\vec{\theta})}{\partial\vec{\theta}} = \frac{\partial(\vec{y}^T\vec{y})}{\partial\vec{\theta}}^{=0} - 2(\vec{y}^T\underline{A})^T + \left[(\underline{A}^T\underline{A}) + (\underline{A}^T\underline{A})^T\right]\vec{\theta}$$

Let:

$$\frac{\partial E(\vec{\theta})}{\partial\vec{\theta}} = 0 \quad ; \quad \vec{\theta} = \hat{\vec{\theta}}$$

Consider:

$$\frac{\partial(\vec{y}^T\underline{A}\vec{x})}{\partial\vec{x}} = A^T\vec{y}$$

$$= 0 - 2\underline{A}^T\vec{y} + \left[\underline{A}^T A + \underline{A}^T\left(\underline{A}^T\right)^T\right]\hat{\vec{\theta}}$$

$$-2\underline{A}^T\vec{y} + 2\underline{A}^T\underline{A}\hat{\vec{\theta}} = 0$$

$$\underline{A}^T\underline{A}\hat{\vec{\theta}} = \underline{A}^T\vec{y}$$

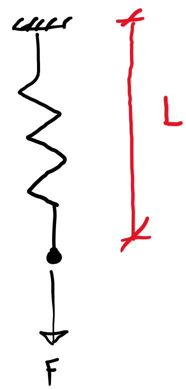
$$\left(\underline{A}^T\underline{A}\right)^{-1}\left(\underline{A}^T\underline{A}\right)\hat{\vec{\theta}} = \left(\underline{A}^T\underline{A}\right)^{-1}\underline{A}^T\vec{y}$$

$$\hat{\vec{\theta}} = \left(\underline{A}^T\underline{A}\right)^{-1}\underline{A}^T\vec{y}$$

$$\boxed{\hat{\vec{\theta}} = \frac{\underline{A}^T\vec{y}}{\underline{A}^T\underline{A}}}$$

Example 3.1 (Jang's Book)

$m = 7$



Experiment	Force (Newtons)	Length of Spring (inches)
1	1.1	1.5
2	1.9	2.1
3	3.2	2.5
4	4.4	3.3
5	5.9	4.1
6	7.4	4.6
7	9.2	5.0

$$L = k_0 + k_1 F$$

$$\begin{cases} k_0 + 1.1k_1 = 1.5 \\ k_0 + 1.9k_1 = 2.1 \\ \cdots \\ k_0 + 9.2k_1 = 5.0 \end{cases}$$

$$\underline{A}\hat{\vec{\theta}} = \vec{y} - \vec{e}$$

Since $\vec{e} = \vec{y} - \underline{A}\vec{\theta}$

$$\begin{bmatrix} 1 & 1.5 \\ 1 & 2.1 \\ \vdots & \vdots \\ 1 & 5.0 \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_7 \end{bmatrix} - \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_7 \end{bmatrix}$$

$$\hat{\vec{\theta}} = \begin{bmatrix} k_0 \\ k_1 \end{bmatrix} = \frac{\underline{A}^T \vec{y}}{\underline{A}^T \underline{A}}$$ (2×2 matrix)

Use MATLAB (*inv* and $.*$ operators)

Or, manually (since it is a 2x2 matrix) via:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$\hat{\vec{\theta}} = \begin{bmatrix} k_0 \\ k_1 \end{bmatrix} = \begin{bmatrix} 1.20 \\ 0.44 \end{bmatrix}$$

$$L = 1.20 + 0.40F$$

Mathworks:
MATLAB toolboxes > Fuzzy Logic
(do the tutorials)

LSE (least squares estimator) → used to optimize the linear parameters of a system

$$\vec{\theta} = \{\theta_1, \theta_2, \theta_3, \dots, \theta_n\}^T$$
$$\vec{u}_i = \{x_1, x_2, x_3, \dots, x_p\}^T$$

$$m - \text{training data pairs}$$

$$\{\vec{u}_1, y_1\}, \{\vec{u}_2, y_2\}, \dots, \{\vec{u}_m, y_m\}$$
$$i = 1, 2, 3, \dots, m$$

$$\underline{A}\vec{\theta} = \vec{y}$$
$$\vec{\theta} = \left(\underline{A}^T \underline{A}\right)^{-1} \underline{A}^T \vec{y}$$

This is **offline training** (speed of operation is not a primary concern)
- you use all the training data pairs at once.

Recursive, or **online training**, is when training data pairs are used one after the other, or one at a time.

4.2 Recursive Lease Squares Estimator (LSE)

Suppose m –training data pairs.
k^{th} training data pair
k^{th} training operation

$$0 \le k \le m - 1$$
$$(\text{In MATLAB: } 1 \le k \le m)$$

Corresponding to the k^{th} training data pair: $1, 2, \dots, k$

$$\begin{bmatrix} f_1(\vec{u}_1) & f_2(\vec{u}_1) & \cdots & f_n(\vec{u}_1) \\ f_1(\vec{u}_2) & f_2(\vec{u}_2) & \cdots & f_n(\vec{u}_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\vec{u}_k) & f_2(\vec{u}_k) & \cdots & f_n(\vec{u}_k) \\ f_1(\vec{u}_{k+1}) & f_2(\vec{u}_{k+1}) & \cdots & f_n(\vec{u}_{k+1}) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \\ y_{k+1} \end{bmatrix}$$

$$\underline{A}\vec{\theta}_k = \vec{y}$$
$$\vec{\theta}_k = \left(\underline{A}^T \underline{A}\right)^{-1} \underline{A}^T \vec{y}$$

If $(k+1)^{th}$ training data pair is available:

$$\{\vec{u}_{k+1}, y_{k+1}\}$$

Will do $(k+1)^{th}$ update operation:

$$\begin{bmatrix} \underline{A} \\ \vec{a}_{k+1}^T \end{bmatrix} \vec{\theta}_{k+1} = \begin{bmatrix} \vec{y} \\ y_{k+1} \end{bmatrix}$$

$$\vec{\theta}_{k+1} = \underbrace{\left[\begin{bmatrix} \underline{A} \\ \vec{a}_{k+1}^T \end{bmatrix}^T \begin{bmatrix} \underline{A} \\ \vec{a}_{k+1}^T \end{bmatrix}\right]^{-1}}_{P_{k+1}} \begin{bmatrix} \underline{A} \\ \vec{a}_{k+1}^T \end{bmatrix}^T \begin{bmatrix} \vec{y} \\ y_{k+1} \end{bmatrix}$$

$$\vec{\theta}_{k+1} \sim \vec{\theta}_k + \text{update (modification)}$$

Introduce:

$$\underline{P}_k = \left(\underline{A}^T \underline{A}\right)^{-1}$$

$$\underline{P}_k^{-1} = \underline{A}^T \underline{A}$$

$$\underline{P}_{k+1} = \left[\begin{bmatrix} \underline{A} \\ \vec{a}_{k+1}^T \end{bmatrix}^T \begin{bmatrix} \underline{A} \\ \vec{a}_{k+1}^T \end{bmatrix}\right]^{-1} \quad \leftarrow (1)$$

$$= \left[\begin{bmatrix} \underline{A}^T & \vec{a}_{k+1} \end{bmatrix} \begin{bmatrix} \underline{A} \\ \vec{a}_{k+1}^T \end{bmatrix}\right]^{-1}$$

$$\underline{P}_{k+1} = \left[\underline{A}^T \underline{A} + \vec{a}_{k+1}^T \vec{a}_{k+1}\right]^{-1} \quad \leftarrow (2)$$

$$\underline{P}_{k+1}^{-1} = \underbrace{\underline{A}^T \underline{A}}_{P_k^{-1}} + \vec{a}_{k+1} \vec{a}_{k+1}^T$$

$$\underline{P}_{k+1}^{-1} = \underline{P}_k^{-1} + \vec{a}_{k+1} \vec{a}_{k+1}^T \quad \leftarrow (3)$$

$$\vec{\theta}_k = \left(\underline{A}^T \underline{A}\right)^{-1} \underline{A}^T \vec{y}$$

$$\vec{\theta}_k = \underline{P}_k \underline{A}^T \vec{y} \quad \leftarrow (4)$$

$$\vec{\theta}_{k+1} = \underline{P}_{k+1} \begin{bmatrix} \underline{A}^T & \vec{a}_{k+1} \end{bmatrix} \begin{bmatrix} \vec{y} \\ y_{k+1} \end{bmatrix}$$

$$\vec{\theta}_{k+1} = \begin{bmatrix} \underline{A}^T \vec{y} & \vec{a}_{k+1} y_{k+1} \end{bmatrix} \quad \leftarrow (5)$$

From Eq. (4):

$$\underline{P}_k^{-1} \vec{\theta}_k = \underbrace{\underline{P}_k^{-1} \underline{P}_k}_{I} \underline{A}^T \vec{y}$$

$$\underline{A}^T \vec{y} = \underline{P}_k^{-1} \vec{\theta}_k$$

Eq. (5) becomes:

$$\vec{\theta}_{k+1} = \underline{P}_{k+1} \left[\underline{P}_k^{-1} \vec{\theta}_k \quad \vec{a}_{k+1}^T y_{k+1}\right]$$

From Eq. (3):

$$\underline{P}_k^{-1} = \underline{P}_{k+1}^{-1} - \vec{a}_{k+1}^T \vec{a}_{k+1}$$

Eq. (5) becomes:

$$\vec{\theta}_{k+1} = \underline{P}_{k+1} \left[\left(\underline{P}_{k+1}^{-1} - \vec{a}_{k+1} \vec{a}_{k+1}^T\right) \vec{\theta}_k \quad \vec{a}_{k+1} y_{k+1}\right]$$

$$= \left[\mathrm{I} - \underline{P}_{k+1}\vec{a}_{k+1}\vec{a}_{k+1}^{T}\right]\vec{\theta}_k + \underline{P}_{k+1}\vec{a}_{k+1}y_{k+1}$$

$$= \vec{\theta}_k - \underline{P}_{k+1}\vec{a}_{k+1}\vec{a}_{k+1}^{T}\vec{\theta}_k + \underline{P}_{k+1}\vec{a}_{k+1}y_{k+1}$$

$$\vec{\theta}_{k+1} = \vec{\theta}_k + \underline{P}_{k+1}\vec{a}_{k+1}\left(y_{k+1} - \vec{a}_{k+1}^{T}\vec{\theta}_k\right) \tag{6}$$

From Eq. (3):

$$\underline{P}_{k+1}^{-1} = \underline{P}_k^{-1} + \vec{a}_{k+1}\vec{a}_{k+1}^{T}$$

$$\underline{P}_{k+1} = \left[\underline{P}_k^{-1} + \vec{a}_{k+1}\vec{a}_{k+1}^{T}\right]^{-1}$$

A B C

Formula:

$$(A + BC)^{-1}$$
$$= A^{-1} - A^{-1}B(I + CA^{-1}B)^{-1}CA^{-1}$$

$$A = \underline{P}_k^{-1}$$
$$B = \vec{a}_{k+1}$$
$$C = \vec{a}_{k+1}^{T}$$

$$\underline{P}_{k+1} = \underline{P}_k - \underline{P}_k\,\vec{a}_{k+1}\left(I + \vec{a}_{k+1}^{T}\underline{P}_k\,\vec{a}_{k+1}\right)^{-1}\vec{a}_{k+1}^{T}\underline{P}_k$$

$$\underline{P}_{k+1} = \underline{P}_k - \frac{\underline{P}_k\,\vec{a}_{k+1}\vec{a}_{k+1}^{T}\underline{P}_k}{I + \vec{a}_{k+1}^{T}\underline{P}_k\,\vec{a}_{k+1}} \tag{7}$$

Use Eq. (6) and Eq. (7) to do recursive LSE and update $\vec{\theta}_{k+1}$

Initialization:

$$\underline{P}_0 = \alpha I$$

Where α is a larger number (1000, 10000, etc.).

From this, you can generate:

→ To be used in project

$$\left(\vec{\theta}_0 \ldots \vec{\theta}_1 \ldots \vec{\theta}_2 \ldots\right)$$

4.3 Gradient Algorithms

Non-linear parameter optimization method.

For linear parameters, LSE is the general method – not many methods are required.

Compared to linear parameter optimization, there are many optimization methods for non-linear systems, but this one is the basic one (most general).

$$\vec{\theta} = \left[\vec{\theta}_1, \vec{\theta}_2, \dots, \vec{\theta}_n\right]^T$$

Objective function (error function):

$$E(\vec{\theta})$$

We want to minimize this function.

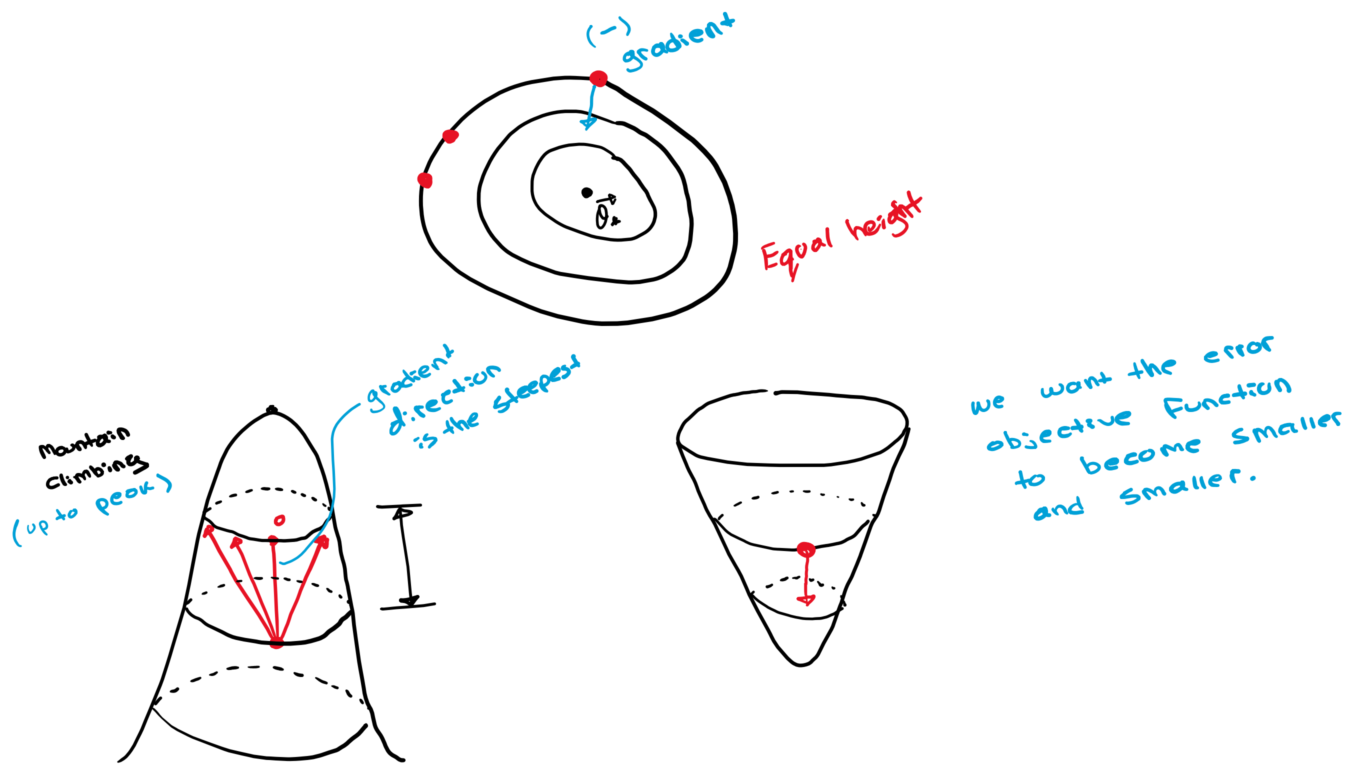
But, θ can have many values, and it's possible that different numbers produce the same value:

Consider:

$2x_1^2 + x_2$p

When $x_1 = 1, x_2 = 1, E(\theta) = 3$
When $x_1 = 2, x_2 = -5, E(\theta) = 3$

These points would be on the same 'error height'



3.3 Gradient Methods

Suppose this parameter has a non-linear relationship with the output:

$$\vec{\theta}$$

For example (gaussian function),

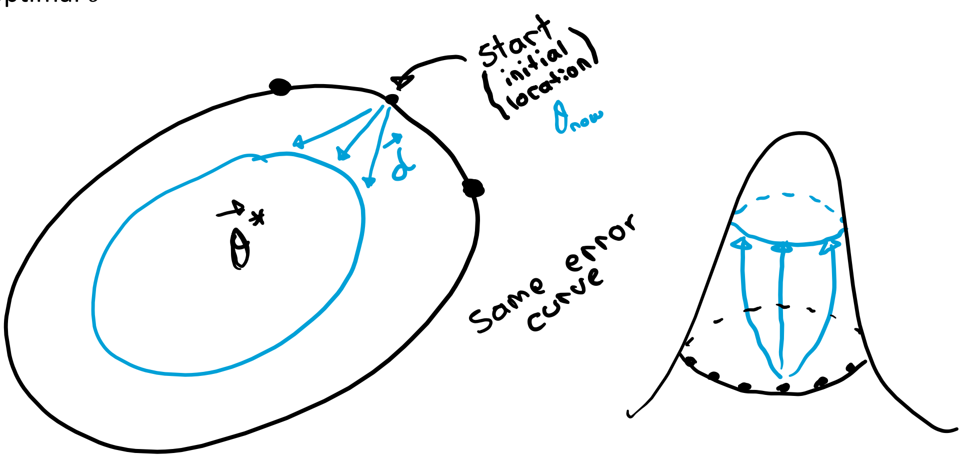
$$\mu_A(x) = e^{\frac{-(x-\mu)^2}{\sigma^2}}$$

For this function, the relationship between μ and σ and the MF μ_A is non-linear, and by association, the error function E.

Objective Function $\quad \vec{\theta} = [\theta_1, \theta_2, \dots, \theta_n]^T$

Error Function $\quad E(\vec{\theta})$

Looking for optimal $\vec{\theta}^*$



$$\vec{\theta}_{next} = \vec{\theta}_{now} + \eta\vec{d}$$

$\eta =$ step
$\vec{d} =$ direction vector

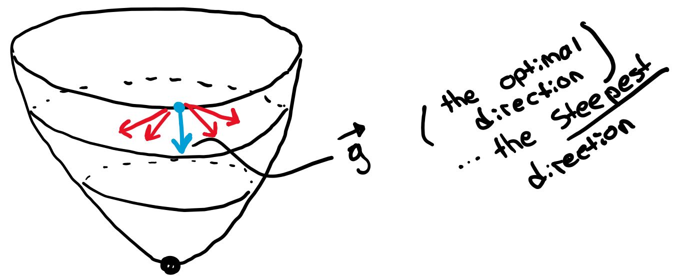
k^{th} step:

$$\vec{\theta}_k$$

$(k+1)^{th}$ step:

$$\vec{\theta}_k + \eta_k\vec{d}_k$$

Generally, $E(\vec{\theta}_{k+1}) \leq E(\vec{\theta}_k)$

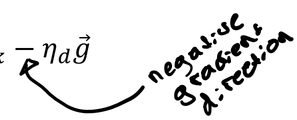


Steepest-gradient descent method:

$$\vec{g}(\vec{\theta}) = \left[\frac{\partial E}{\partial \theta_1} \quad \frac{\partial E}{\partial \theta_2} \quad \cdots \quad \frac{\partial E}{\partial \theta_n}\right]^T$$

Let $\vec{g}(\vec{\theta}) = \left.\frac{\partial E(\vec{\theta})}{\partial \hat{\vec{\theta}}}\right|_{\hat{\vec{\theta}}} = 0$

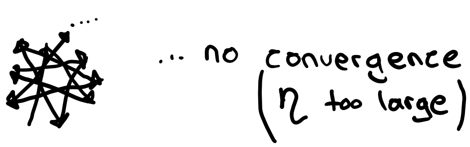
$$\vec{\theta}_{k+1} = \vec{\theta}_k - \eta_d \vec{g}$$



In general, this is a recursive (or repetitive) algorithm.

- Stopping criteria:

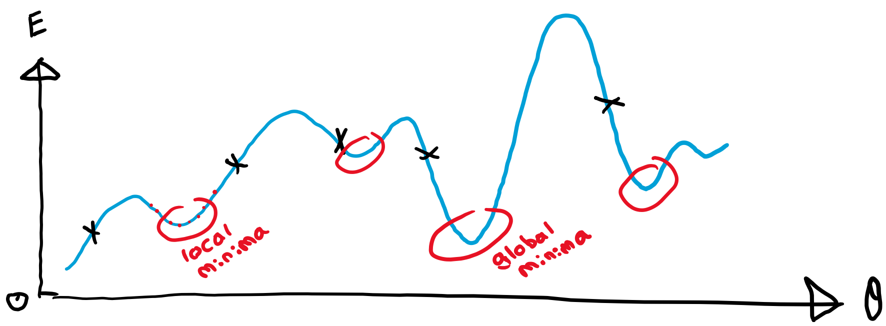
1) $E \leq$ threshold: 10^{-5}
2) # of iterations: ≤ 200



3.4 Genetic Algorithms (GA)

Easy to read, does not involve gradient related operations, it is a derivative free method, but it takes a very long time to execute.

It can reach the global minimum, but all other derivative-based methods reach a local minimum (depending on initial conditions):

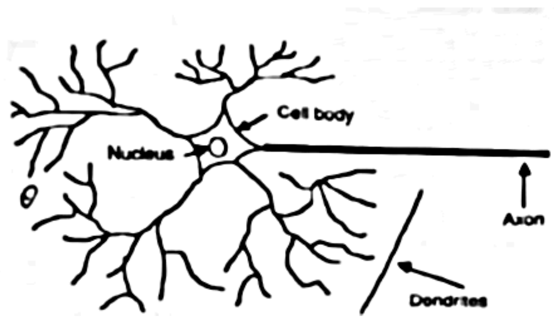


- Population-based search, so it returns the *best* results
- But it's <u>very</u> time consuming (like, 8 hours)
- Compare with gradient-based method (15 seconds)
- Evolution operation
    - Reproduction, cross over, mutation

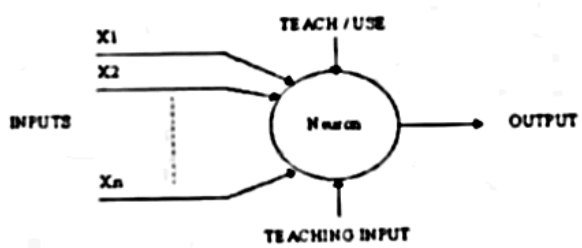
Chapter 4: Artificial Neural Networks

4.1 Introduction

Neuron networks: parallel and distributed neurons.



Artificial neural networks:

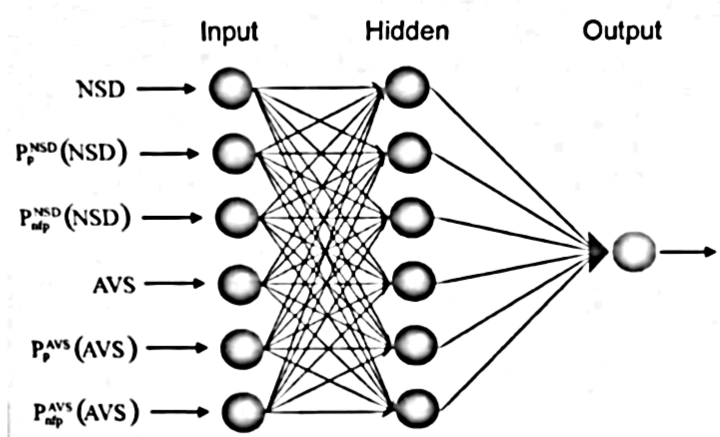


4.2 Features of Neural Networks

Layered neurons

Weighted links

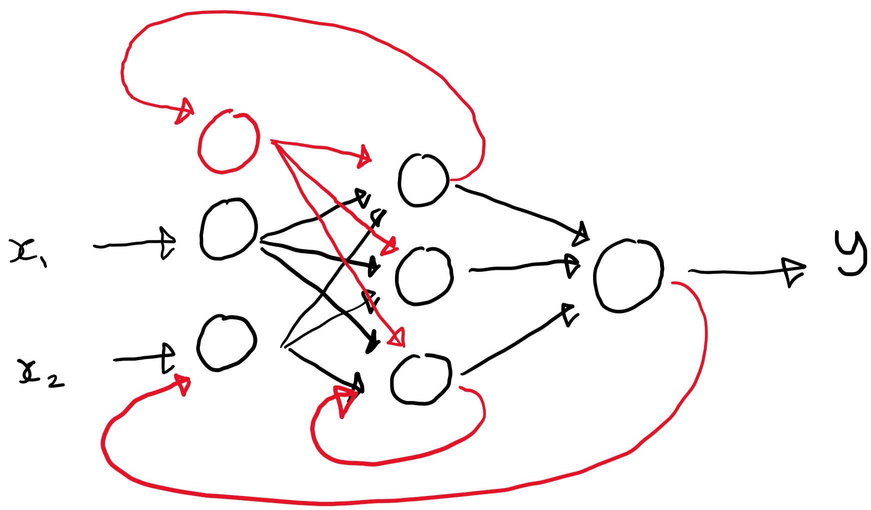
(link weights 0~1.0)



1) Neural network topologies

(a) Feed forward topology (static neural network)

unidirectional links (just move in one direction, in this case from input to hidden nodes)



(b) Recurrent topology (dynamic neural network)

Outputs can move to back into itself, can mode into a different node… they can move anywhere depending on requirements.

Much more complicated, but has some distinct advantages, specifically in terms of access to historical data:

Consider:

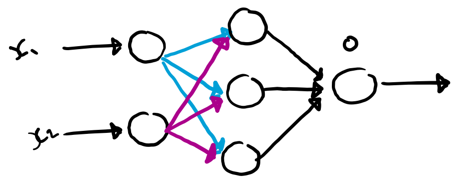
- k^{th} step: $x_1(k),\ x_2(k)$
  Output: $y(k)$
- $(k+1)^{th}$ step: $x_1(k+1),\ x_2(k+1)$
  Historical information $\rightarrow y(k)$
  Output: $y(k+1)$

4.2 Features of Neural Networks (Cont'd)

1) NN Topologies

- FF NNs

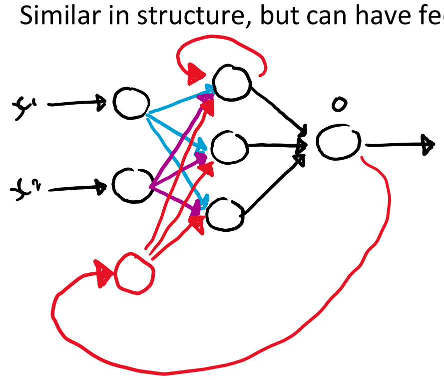
Simple in structure, consider two inputs and one output below:



Connections between intermediate neurons and output neurons are unidirectional.
Static modeling method (given a set of inputs, an output is generated).

- Recurrent NNs

Similar in structure, but can have feedback links.

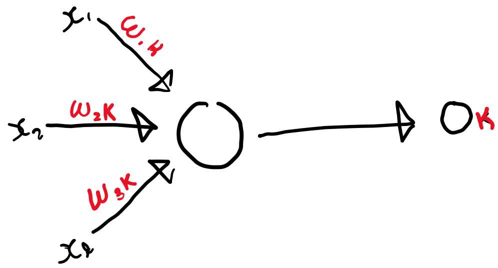


Gives access to historical information, making the network a 'dynamic' network. However, training complexity increases.

$x(k) = O(k)$
Has access to inputs x_1 and x_2

$x(k+1) = O(k+1)$
Has access to inputs x_1, x_2, and historical data $x(k)$

2) Activation functions



Input: $x_1 w_{1k} + x_2 w_{2k} + \cdots + x_l w_{lk}$

$$\sum_{i=1}^{l} x_i w_{ik}$$

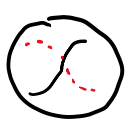
$$O_k = f\left(\sum_{i=1}^{l} x_i w_{ik} - \theta_k\right)$$

Where:

f = activation function

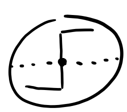
θ_k = threshold of the k^{th} neuron (bias)

Sigmoid function:



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

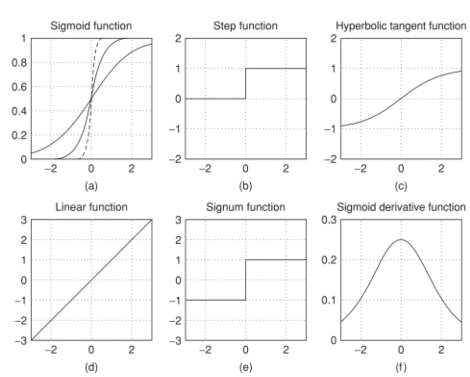
Signum function:



$$sgm(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x = 0 \\ -1 & ; \quad x < 0 \end{cases}$$

Step function:

$$step(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x \leq 0 \end{cases}$$



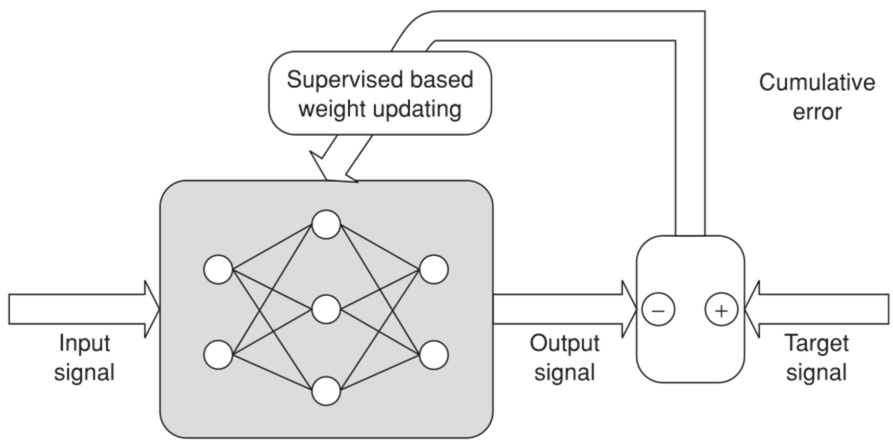
3) Neural Network Learning

- Supervised learning

We have a desired output, which can be considered a teacher.

Teacher $(\vec{x}, t)$

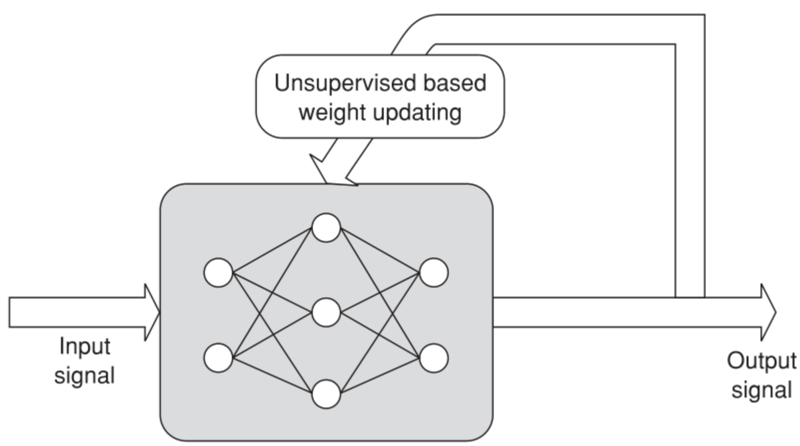
We compare the desired output and the calculated output and feed the error information back into the system to train it. This is the general approach for engineering applications.



- Unsupervised learning

In applications where we can't get a target, or we can't find the desired output, we have no teacher.

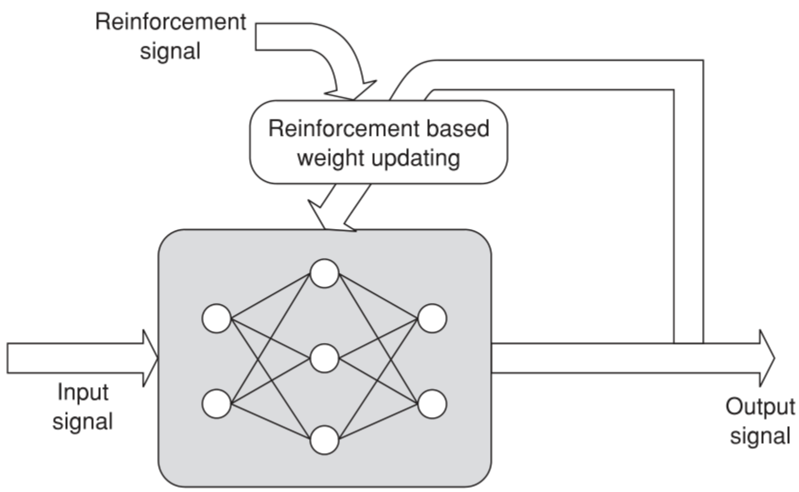
Thus, we cannot do supervised learning – this is usually the case for 'big data'.



- Reinforcement learning

Feedback information provides a guide for training, but not a target.

Can be used for special circumstances like scenarios in video games (i.e. beating a bad guy in fewer moves to achieve a higher bonus)



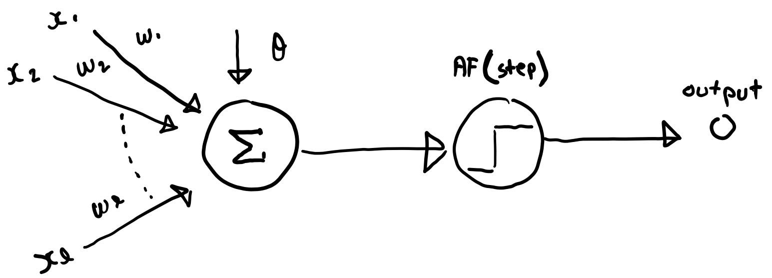
Additional info:

ANFIS – adaptive neuro-fuzzy inference system

Our course will focus on the following engineering applications:
- Control
- Classification (diagnosis)
- Modeling (forecasting)

4.4 Connectionist Modeling

1) McCulloch-Pitts (MP) Modeling



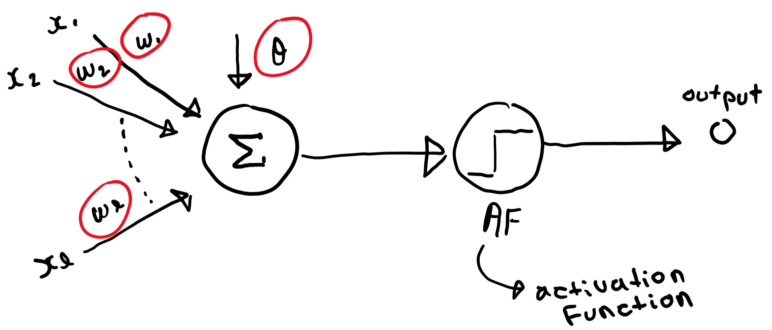
$w_1, w_2, \dots, w_l$ are fixed

$$O = f(x_1w_1 + x_2w_2 + \cdots + x_lw_l - \theta)$$
$$= f\left(\sum_{i=1}^{l} x_iw_i - \theta\right)$$

threshold

Step: $\sum x_iw_i = 0.001 \quad ; \quad 0 \rightarrow 1$

2) Perceptron Modeling

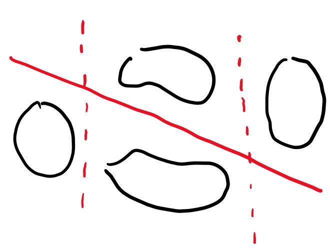


Train link weights ($w_1, w_2, \dots, w_l$) and the bias (θ), but we don't train the activation function parameters, it is fixed – we simply choose one.

$$O = f\left(\sum_{i=1}^{l} x_iw_i - \theta\right)$$

If the training data pairs are linearly separable (or separable by hyper planes) then the training process can converge.

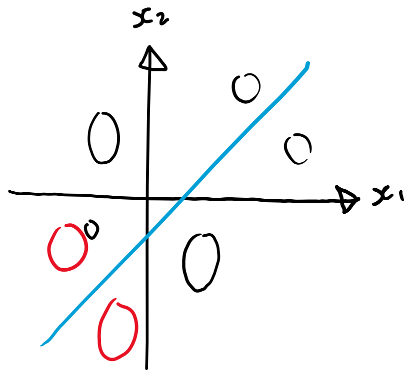
Meaning we can get optimal parameters by a finite number of training operations.



Consider 2-D data sets:

x_1, x_2

$w_1 x_1 + w_2 x_2 - \theta = 0$



Summary of Perceptron training algorithm:

1. Initialize weights and thresholds to small random values
2. Choose an input-output pattern $(x^{(k)}, t^{(k)})$ from the training data.
3. Compute the network's actual output $o^{(k)} = f\left(\sum_{i=1}^{j} w_i x_i^{(k)} - \theta\right)$.
4. Adjust the weights and bias according to the Perceptron learning rule:
   $\Delta w_i = \eta\left[t^{(k)} - o^{(k)}\right]x_j^{(k)}$, and $\Delta\theta = -\eta\left[t^{(k)} - o^{(k)}\right]$, where $\eta \in [0, 1]$ is the Perceptron's learning rate.
   If f is the signum function, this becomes equivalent to:

$$\Delta w_i = \begin{cases} 2\eta t^{(k)} x_j^{(k)} & ; \quad \text{if } \mathrm{t}^{(k)} \neq o^{(k)} \\ 0 & ; \quad \text{otherwise} \end{cases}$$

$$\Delta\theta = \begin{cases} -2\eta t^{(k)} & ; \quad \text{if } \mathrm{t}^{(k)} \neq o^{(k)} \\ 0 & ; \quad \text{otherwise} \end{cases}$$

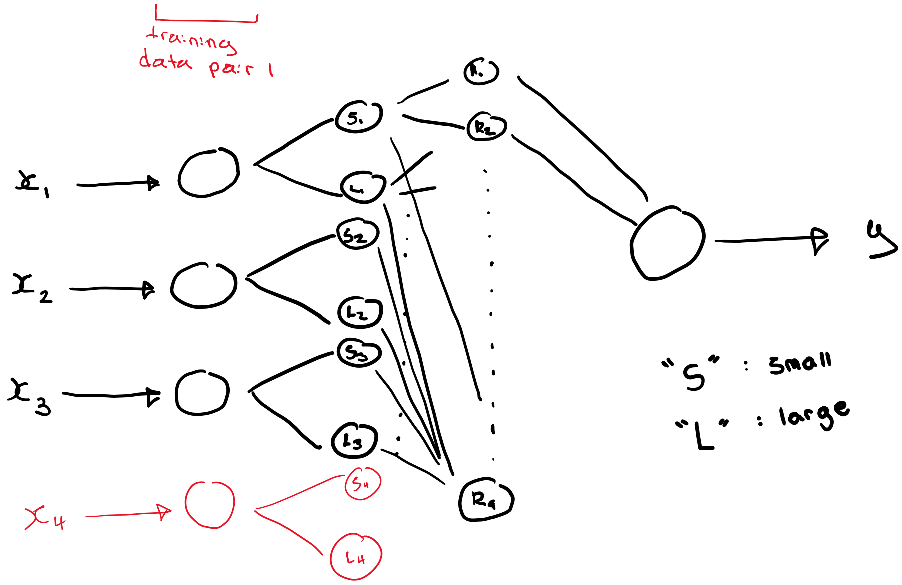
5. If a whole epoch is complete, then pass to the following stepl otherwise go to Step 2
6. If the weights (and bias) reached steady state ($\Delta w_i \approx 0$) through the whole epoch, then stop the learning; otherwise go through one more epoch starting from Step 2.s

Tensorflow for NNs:
http://playground.tensorflow.org/

Mackey-Glass data:
https://www.mathworks.com/matlabcentral/fileexchange/24390-mackey-glass-time-series-generator

For example:

If we have x_1 , x_2 , x_3 , x_4 , x_5 , x_6 , x_7 , x_8 , …



One -step-ahead prediction:

$$\{(x_1, x_2, x_3); x_4\}$$
$$\{(x_2, x_3, x_4); x_5\}$$
$$\{(x_3, x_4, x_5); x_6\}$$
$$\vdots$$

TSK-1:

If $(x_1 \text{ is } s_1)$ and $(x_2 \text{ is } s_2)$ and $(x_3 \text{ is } L_3)$ then
$y_1 = a_1x_1 + b_1x_2 + c_1x_3 + d_1$

Then each rule has 4 linear parameters to be updated, and we have 9 rules.

Then in total, we have $4 \text{ x } 9 = 36$ linear parameters to be updated.

Non-linear parameters are related to the 'small' and 'large' membership function parameters (assume each is a sigmoid function, and has two parameters):
$2 \text{ x } 6 = 12$

In total, there are 48 training data pairs.

Training data pairs is at least 5 times the number of non-linear parameters:

$$5 \text{ x } 48 = 240$$

If you have 16 rules with 4 inputs, each having 2 membership functions:
16 rules: linear

R_1: if (x_1 is s_1) and (x_2 is s_2) and (x_3 is L_3) and (x_4 is s_4)

Then: $y_1 = a_1x_1 + b_1x_2 + c_1x_3 + d_1x_4 + g_1$

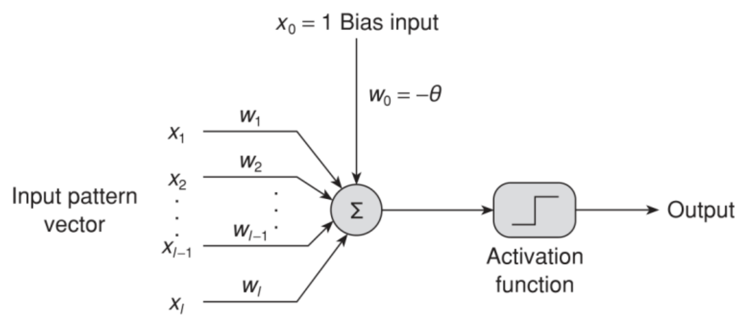
How many linear parameters for each rule?
$5 \text{ x } 16 = 80$

How many non-linear function parameters?
$8 \text{ x } 2 = 16$

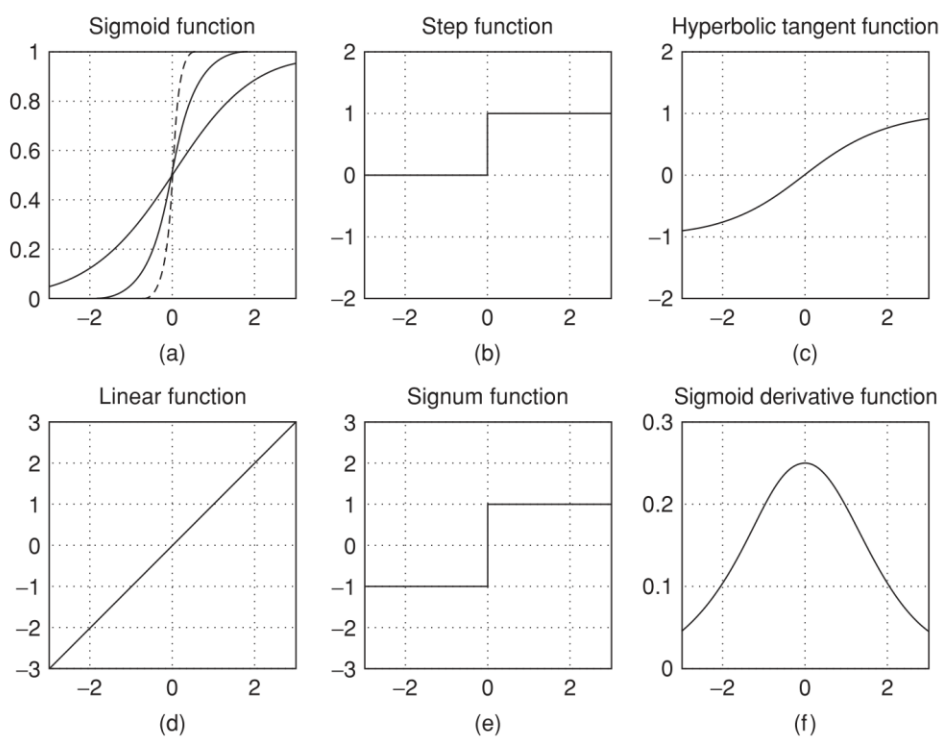
4.4 Connectionist Modeling

- MP Modeling

$$O = f(x_1 w_1 + x_2 w_2 + \cdots + x_l w_l - \theta) = f\left(\sum_{i=1}^{l} x_i w_i - \theta\right)$$



Consider the general activation functions:



- Perceptron

Training algorithm (DIRECTLY FROM TEXTBOOK):

1. Initialize weights and thresholds to small random values.
2. Choose an input-output pattern $\left(x^{(k)}, t^{(k)}\right)$ from the training data.
3. Compute the network's actual output $o^{(k)} = f\left(\sum_{i=1}^{l} w_i x_i^{(k)} - \theta\right)$.
4. Adjust the weights and bias according to the Perceptron learning rule:
$$\Delta w_i = \eta\left[t^{(k)} - o^{(k)}\right]x_i^{(k)}$$
And:
$$\Delta\theta = -\eta\left[t^{(k)} - o^{(k)}\right]$$
Where $\eta \in [0, 1]$ is the Perceptron's learning rate.
If f is the signum function, this becomes equivalent to:
$$\Delta w_i = \begin{cases} 2\eta t^{(k)} x_i^{(k)} & ;\ if\ t^{(k)} \neq o^{(k)} \\ 0 & ;\ \text{otherwise} \end{cases}$$
And:
$$\Delta\theta = \begin{cases} -2\eta t^{(k)} & ;\ if\ t^{(k)} \neq o^{(k)} \\ 0 & ;\ \text{otherwise} \end{cases}$$
5. If a whole epoch is complete, then pass to the following step; otherwise go to Step 2.
6. If the weights (and bias) reached steady state ($\Delta w_i \approx 0$) through the whole epoch, then stop the learning; otherwise go through one more epoch starting from Step 2.

Training algorithm (CLASS NOTES):

$$\vec{x}^{(k)} = \left\{x_1^{(k)}, \quad x_2^{(k)}, \dots, \quad x_l^{(k)}\right\}^T$$

Where:
$t^{(k)} = \text{target, desired output}$

$$O^{(k)} = f\left(\sum_{i=1}^{l} w_i^{(k)} x_i^{(k)} - \theta\right)$$
$$\vec{w}^{(k+1)} = \vec{w}^{(k)} \underbrace{+ \Delta\vec{w}}_{\text{update}}$$

$$\Delta\vec{w} = \eta\left(t^{(k)} - O^{(k)}\right)\vec{x}^{(k)}$$

If $f(\cdot) \sim signum\ fxn \begin{cases} 1 & ; \quad \text{input} > 0 \\ 0 & ; \quad \text{otherwise} \\ -1 & ; \quad \text{input} < 0 \end{cases}$

If $t^{(k)} = O^{(k)}$ (then we don't need to update anything)

$$\Delta\vec{w} = \begin{cases} 0 & ; \quad t^{(k)} = O^{(k)} \\ 2\eta t^{(k)}\vec{x}^{(k)} & ; \quad \text{otherwise} \end{cases}$$

Otherwise, if $t^{(k)} \neq O^{(k)}$

If $t^{(k)} = +1$, then $O^{(k)} = -1 = -t^{(k)}$
If $t^{(k)} = -1$, then $O^{(k)} = +1 = -t^{(k)}$

Similarly:

$$\theta^{(k+1)} = \theta^{(k)} + \Delta\theta$$

$$\Delta\theta = -\eta(t^{(k)} - O^{(k)})$$

If AF is $signum\ fxn$
If $t^{(k)} = O^{(k)}$
Otherwise $t^{(k)} \neq O^{(k)}$, $O^{(k)} = -t^{(k)}$

$$\Delta\theta = \begin{cases} 0 & ; \quad t^{(k)} = O^{(k)} \\ 2\eta t^{(k)} & ; \quad \text{otherwise} \end{cases}$$

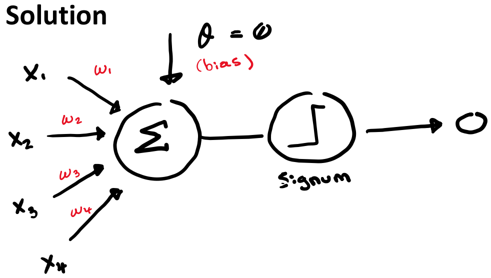
Example 4.1 (Book 1)

Train a network using the following set of input and desired output training vectors:

$$x^{(1)} = [1, -2, 0, 1]^T; \;\; t^{(1)} = -1$$
$$x^{(2)} = [0, 1.5, -0.5, -1]^T; \;\; t^{(2)} = -1$$
$$x^{(3)} = [-1, 1, 0.5, -1]^T; \;\; t^{(3)} = +1$$

With initial weight vector $w^{(1)} = [1, -1, 0, 0.5]^T$, learn $\eta = 0.1$

Solution



$\eta = 0.1$
$w^{(1)} = [1, -1, 0, 0.5]^T$

$O = f(x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 - \theta) = f(\vec{w}^T \vec{x} - \theta)$
(but θ is 0 here)

Epoch 1:

$$\vec{x}^{(1)} = [1, -2, 0, -1]^T, \qquad t^{(1)} = -1$$

$$O^{(1)} = sgn\left(\vec{w}^{(1)^T} \vec{x}\right)$$
$$= sgn\left(\begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \right)$$
$$= sgn(1 + 2 + 0 - 0.5) = sgn(2.5)$$

$$\vec{w}^{(2)} = \vec{w}^{(1)} + \Delta\vec{w}$$
$$\vec{w}^{(2)} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 2(0.1)(-1) \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$
$$\vec{w}^{(2)} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ -0.7 \end{bmatrix}$$

Input the 2nd training data pair $\vec{x}^{(2)}$:

$$O^{(2)} = f\left({\vec{w}^{(2)}}^T \vec{x}^{(2)}\right)$$

$$= sgn\left(\begin{bmatrix} 0.8 & -0.6 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \right)$$

$$= sgn(0 - 0.9 + 0 - 0.7)$$
$$= sgn(-1.6)$$
$$= -1 = t^{(2)} = -1$$

$$\vec{w}^{(3)} = \vec{w}^{(2)} + \Delta\vec{w} = \vec{w}^{(2)} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Input the 3rd training data pair $\vec{x}^{(3)}$:

$$\vec{x}^{(3)} = [-1, 1, 0.5, -1]^T, \qquad t^{(3)} = 1$$

$$O^{(3)} = f\left({\vec{w}^{(3)}}^T \vec{x}^{(3)}\right)$$

$$= sgn\left(\begin{bmatrix} 0.8 & -0.6 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} \right)$$

$$= sgn(0.8 - 0.6 + 0 - 0.7)$$
$$= sgn(-2.1)$$
$$= -1 \neq t^{(3)} = 1$$

$$\vec{w}^{(4)} = \vec{w}^{(3)} + \Delta\vec{w}$$

$$\vec{w}^{(4)} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ -0.7 \end{bmatrix} + \underbrace{2(0.1)(1) \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}}_{\Delta w}$$

$$\vec{w}^{(4)} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

<u>End of epoch 1</u>

Since $\Delta w \neq 0$, proceed to epoch 2…

$$\begin{matrix} \left(\vec{x}^{(1)},\ t^{(1)}\right) & \left(\vec{x}^{(2)},\ t^{(2)}\right) & \left(\vec{x}^{(3)},\ t^{(3)}\right) \\ \downarrow & \downarrow & \downarrow \\ \left(\vec{x}^{(4)},\ t^{(4)}\right) & \left(\vec{x}^{(5)},\ t^{(5)}\right) & \left(\vec{x}^{(6)},\ t^{(6)}\right) \end{matrix}$$

$\vec{w}^{(4)} = [0.6, -0.4, 0.1, 0.5]^T$

$O^{(4)} = f\left({\vec{w}^{(4)}}^T \vec{x}^{(4)}\right)$

$$= sgn\left([0.6 \quad -0.4 \quad 0.1 \quad 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \right)$$

$= sgn(0.6 + 0.8 + 0 - 0.5) =$

$= sgn(0.9) = +1 \neq t^{(4)} = -1$

$$\vec{w}^{(5)} = \vec{w}^{(4)} + 2\eta t^{(4)} \vec{x}^{(4)}$$

$$= \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix} + 2(0.1)(-1) \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4 \\ 0 \\ 0.1 \\ 0.7 \end{bmatrix}$$

$\vec{x}^{(5)} = \vec{x}^{(2)} = [0, 1.5, -0.5, -1]^T$

$$O^{(5)} = f({\vec{w}^{(5)}}^T \vec{x}^{(5)})$$

$$\vdots$$

Epoch 3 (still doesn't meet requirements)

$\vec{w}^{(10)} = [0 \quad 0.4 \quad 0.3 \quad 0.3]^T$

Epoch 4 (still doesn't meet requirements)

$\vec{w}^{(12)} = [-2 \quad 0.3 \quad 0.5 \quad 0.3]^T$

After Epoch 5, we can meet the requirements.

Example 4.2 (Example 2 in Book 1)

Assume $\eta = 0.5$, and there exists two sets of patterns to be classified:

Class 1: Target value -1:

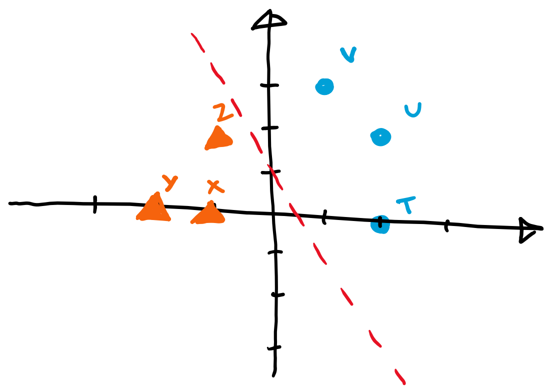
$$T = [2, 0]^T; \quad U = [2, 2]^T; \quad V = [1, 3]^T$$

Class 2: Target value 1:

$$X = [-1, 0]^T; \quad Y = [-2, 0]^T; \quad Z = [-1, 2]^T$$

Solution:

$t^{(1)} = -1$



$$w_1 x_1 + w_2 x_2 - \theta = 0$$

$t^{(2)} = +1$

Assume initial values:

$$w_1 = -1, w_2 = 1, \theta = -1$$
$$(-1)x_1 + (1)x_2 + 1 = 0$$

- Pattern $T = [2, 0]^T$, signum AF

$$0 = sgn(\vec{w}^T \vec{x}) + 1 = sgn\left([-1 \quad 1]\begin{bmatrix}2\\0\end{bmatrix} + 1\right)$$
$$= sgn(-2 + 0) + 1 = -1 = t^{(1)}$$

- Input $\vec{x} = \vec{u} = \begin{bmatrix}2\\2\end{bmatrix}$

$$O = sgn(\vec{w}^T \vec{x} - \theta)$$
$$= sgn\left([-1 \quad 1]\begin{bmatrix}2\\2\end{bmatrix} + 1\right)$$
$$= sgn(-2 + 2 + 1) = +1 \neq t^{(1)}$$
$$t^{(1)} = -1$$

$$\vec{w}^{(2)} = \vec{w}^{(1)} + 2\eta t^{(1)} \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$
$$= \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 2(0.5)(-1) \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$$

$$\theta^{(2)} = \theta^{(1)} + \Delta\theta$$
$$= -1 + [-2(0.5)(-1)] = 0$$

Boundary function:

$$w_1 x_1 + w_2 x_2 - \theta = 0$$
$$-3x_1 - x_2 = 0$$
$$x_2 = -3x_1$$

- Input $\vec{x} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$

$$O = sgn(\vec{w}^T\ \vec{x} - \theta)$$
$$= sgn\left([-3 \quad -1] \begin{bmatrix} 1 \\ 3 \end{bmatrix} - 0\right)$$
$$= sgn(-3 + 3) = -1 = t^{(1)}$$

$$\vec{w}^{(3)} = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$$

$$\theta^{(3)} = 0$$

- Input $\vec{x} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$

$$O = sgn(\vec{w}^T\ \vec{x} - \theta)$$
$$= sgn\left([-3 \quad -1] \begin{bmatrix} -1 \\ 0 \end{bmatrix} - 0\right)$$
$$= sgn(3 + 0) = +1 = t^{(2)}$$

$$\vec{w}^{(4)} = \vec{w}^{(3)} = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$$

$$\theta^{(4)} = 0$$

- Input $\vec{x} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$

$$O = sgn(\vec{w}^T\ \vec{x} - \theta)$$
$$= sgn\left([-3 \quad -1] \begin{bmatrix} -2 \\ 0 \end{bmatrix} - 0\right)$$
$$= sgn(6 + 0 - 0) = +1 = t^{(2)}$$

$$\vec{w}^{(5)} = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$$

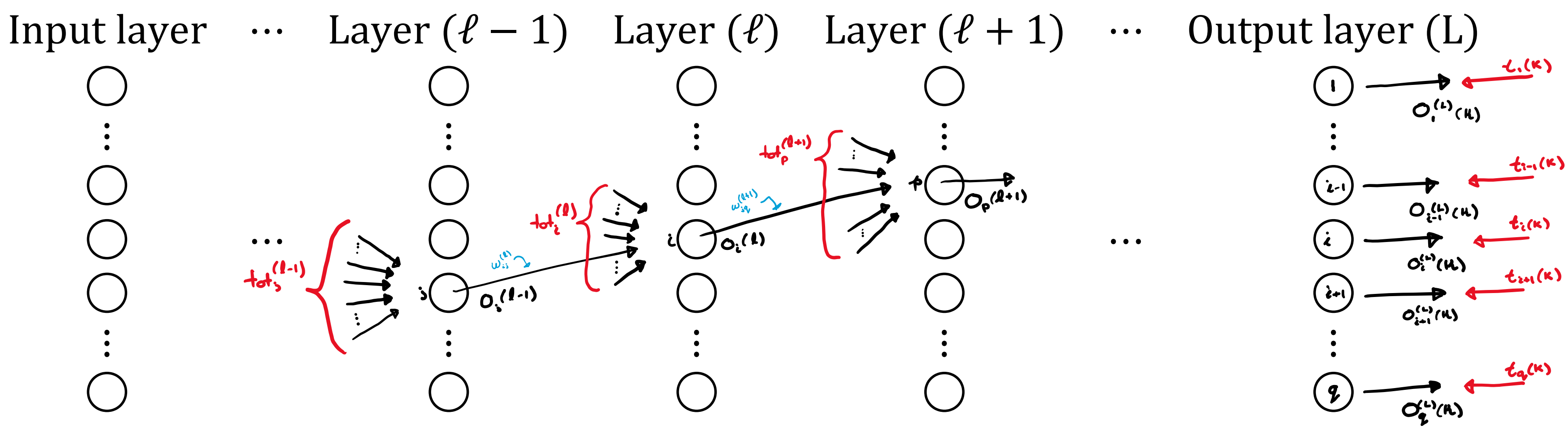
$$\theta^{(5)} = 0$$

- Input $\vec{x} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$O = sgn(\vec{w}^T \, \vec{x} - \theta)$$
$$= sgn\left(\begin{bmatrix} -3 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} - 0\right)$$
$$= sgn(3 - 2 - 0) = 1 = t^{(2)}$$

$$\vec{w}^{(6)} = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$$

$$\theta^{(6)} = 0$$



Input layer
⋯
Layer (ℓ − 1)
Layer (ℓ)
Layer (ℓ + 1)
⋯
Output layer (L)

If $t(k) = k^{th}$ largest target output of the NN

k^{th} training data pair

$k = 1, 2, \dots, n$
$n =$ total number of training data pairs

Error function:

$$E(k) \sim \left[t_1(k) - O_1^{(L)}(k)\right], \dots, \left[t_i(k) - O_i^{(L)}(k)\right], \dots, \left[t_q(k) - O_q^{(L)}(k)\right]$$

$$E(k) = \frac{1}{2}\left[\left(t_1(k) - O_1^{(L)}(k)\right)^2 + \cdots + \left(t_q(k) - O_q^{(L)}(k)\right)^2\right]$$
$$= \frac{1}{2}\sum_{i=1}^{q}[t_i(k) - O_i(k)]^2$$

online training

(For simplicity, drop the "(L)" from notation)

Overall error function:

$$E_c = \sum_{k=1}^{n} E(k)$$
$$= E(1) + E(2) + \cdots + E(k) + \cdots + E(n)$$

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

offline training

$E(k) \sim$ online training
$E_c \sim$ offline training

For online training:

$$\min E(k)$$

$$\vec{w}^{(l)}(k+1) = \vec{w}^{(l)}(k) + \Delta\vec{w}(k)$$
gradient descent method

$$\Delta\vec{w}^{(l)} = \Delta w_{ij}^{(l)}$$

Chain rule:

$$\frac{\partial E(k)}{\partial w_{ij}}$$

$$\begin{aligned}\Delta\vec{w}^{(\ell)} &= \Delta\vec{w}_{ij}^{(\ell)} \\ &= -\eta\frac{\partial E(k)}{\partial O_i^{(\ell)}}\cdot\frac{\partial O_i^{(\ell)}}{\partial t_o t_i^{(\ell)}}\cdot\frac{\partial tot_i{}^{(\ell)}}{\partial w_{ij}^{(\ell)}}\end{aligned}$$

If layer ℓ is the output layer L:

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

Omit "k"

$$\frac{\partial E}{\partial W_{ij}^{(L)}} = \frac{\partial E}{\partial O_i^{(L)}}\cdot\frac{\partial O_i^{(L)}}{\partial t_o t_i^{(L)}}\cdot\frac{\partial tot_i^{(L)}}{\partial w_{ij}^{(L)}}$$

$$tot_i^{(L)} = \rightarrow O_1^{(L-1)}w_{i1}^{(L)} + O_2^{(L-1)}w_{i2}^{(L)} + \cdots + O_j^{(L-1)}w_{ij}^{(L)} + \cdots$$

$$O_i^{(L)} = f\left(tot_i^{(L)}\right)$$

$$\frac{\partial E}{\partial w_{ij}^{(L)}} = -(t_i - O_i)f'\left(tot_i^{(L)}\right)O_j^{(L-1)}$$

$$\begin{aligned}\Delta w_{ij}^{(L)} &= -\eta\frac{\partial E}{\partial w_{ij}^{(L)}} \\ &= \eta\left(t_i - O_i^{(L)}\right)f'\left(tot_i^{(L)}\right)O_j^{(L-1)}\end{aligned}$$

$\delta_i^{(L)}$ = error signal

$\Delta w_{ij}^{(L)} = \eta\delta_i O_j^{(L-1)}$

Given k^{th} training data pair:
$\{ (x_1(k_1), x_2(k_2), \dots)^T, t(k) \}$

$t_i(k) - o_i^{(L)} \sim \text{error}$

<u>Objective function of online training:</u>

$$E(k) = \frac{1}{2}\sum_{i=1}^{q}\left(t_i(k) - o_i^{(L)}(k)\right)^2 \quad ; \quad k = 1, 2, \dots, n$$

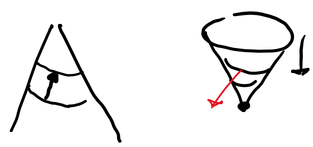
$n = $ the total number of training data pairs

<u>Objective function of offline training:</u>

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

$\vec{W}^{(L)}(k+1) = \vec{W}^{(L)}(k) + \Delta\vec{W}^{(L)}(k)$

$\Delta\vec{W}^{(L)}(k) = -\eta\frac{\partial E(k)}{\partial\vec{W}^{(L)}(k)}$



$$\Delta\vec{W}_{ij}^{(\ell)} = -\eta\frac{\partial E}{\partial o_i^{(\ell)}}\cdot\frac{\partial o_i^{(\ell)}}{\partial tot_i^{(\ell)}}$$

$$tot_i^{(\ell)} = w_{i1}^{(\ell)}o_1^{(\ell-1)} + \dots + w_{ij}^{(\ell)}o_j^{(\ell-1)} + \dots + w_{i(m_1)}^{(\ell)}o_{m_1}^{(\ell-1)}$$

(See previous neural network node map)

<u>Output layer L:</u>

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

$$E(k) = \left(t_1^{(L)} - o_1^{(L)}\right) + \dots + \left(t_i^{(L)} - o_i^{(L)}\right) + \dots + (t_q^{(L)} - o_q^{(L)})$$

$$tot_i^{(L)} = \dots + w_{ij}^{(L)}o_j^{(L-1)} + \dots$$

$$\Delta W_{ij}^{(L)} = -\eta\frac{\partial E}{\partial o_i^{(L)}}\cdot\frac{\partial o_i^{(L)}}{\partial tot_i^{(L)}}\cdot\frac{\partial tot_i^{(L)}}{\partial W_{ij}^{(L)}}$$

$$\Delta W_{ij}^{(L)} = -\eta \frac{1}{2} * (2)\left(t_1^{(L)} - o_1^{(L)}\right)(-1) * f'\left(tot_i^{(L)}\right) * o_j^{(L-1)}$$

$$o_i^{(L-1)} = f\left(tot_i^{(L)}\right)$$

- If a sigmoid AF (activation function) is used:

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

$$x = tot_i^{(L)}$$
$$f(x) = o_i^{(L)}$$

$$f = \frac{1}{1 + e^{-x}}$$
$$f' = [(1 + e^{-x})^{-1}]' = -1(1 + e^{-x})^{-2} e^{-x}(-1)$$
$$= \frac{e^{-x}}{(1 + e^{-x})^2}$$
$$= \frac{1}{1 + e^{-x}} * \left(1 - \frac{1}{1 + e^{-x}}\right)$$

Then,

$$f' = f(1 - f)$$

- If $o_i^{(L)}$ is a sigmoid function:

$$\Delta W_{ij}^{(L)} = \eta\left(t_i^{(L)} - o_i^{(L)}\right) f(1 - f) o_j^{(L-1)}$$
$$= \eta\left(t_i^{(L)} - o_i^{(L)}\right) o_i^{(L)}\left(1 - o_i^{(L)}\right) o_j^{(L-1)}$$

$$\delta_i^{(L)} = \left(t_i^{(L)} - o_i^{(L)}\right) f'\left(tot_i^{(L)}\right)$$
$$\delta_i^{(L)} = \left(t_i^{(L)} - o_i^{(L)}\right) o_i^{(L)}\left(1 - o_i^{(L)}\right)$$

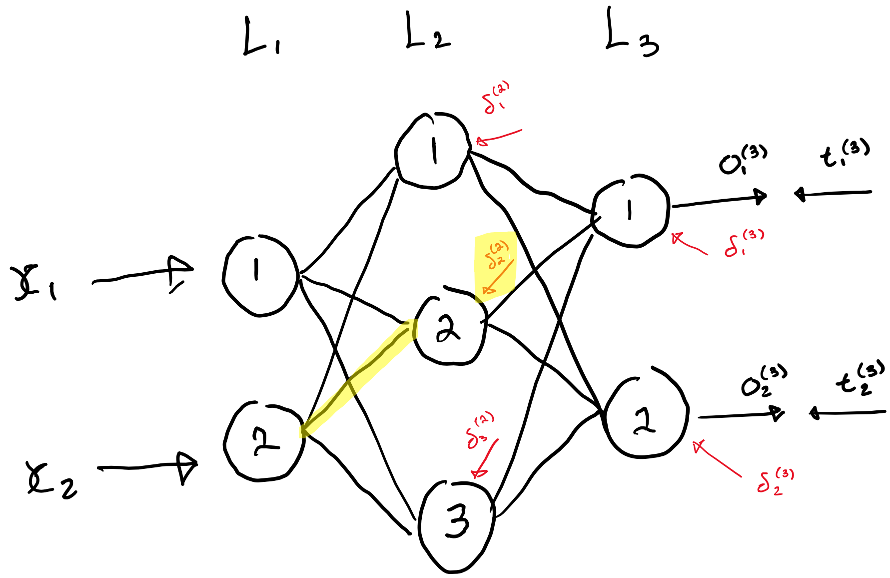
$$W_{ij}^{(L)} = \eta \delta_i^{(L)} * o_j^{(L-1)}$$

For $W_{ij}^{(\ell)}$:

$$\Delta W_{ij}^{(\ell)} = \eta \delta_i^{(\ell)} o_j^{(\ell-1)}$$

$$\left(t_i^{(\ell)} - o_i^{(\ell)}\right) f'\left(tot_i^{(\ell)}\right)$$

General structure:



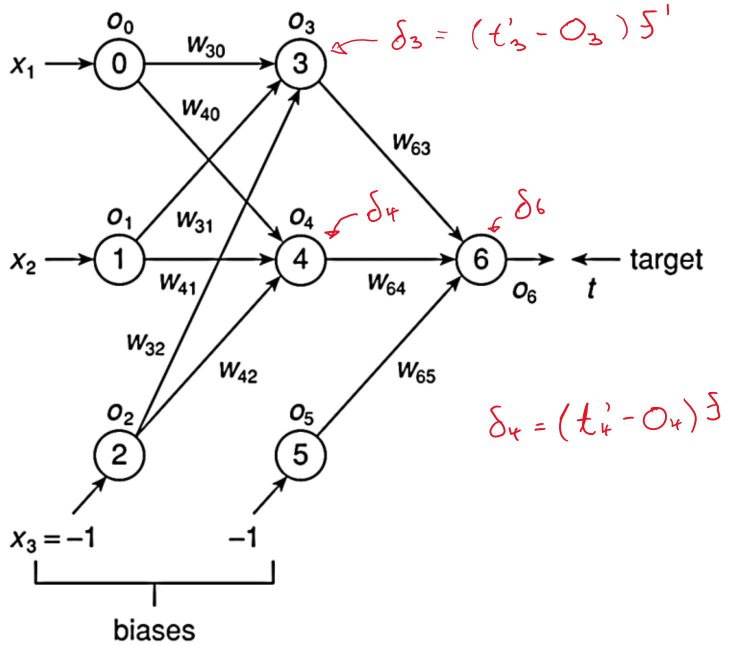
Forward pass → o (calculate output)

Backward pass → update $W_{ij}^{(L)}$

Example 5.1

To illustrate this powerful algorithm, we apply it for the training of the following network, shown in Figure 5.4. The following htree training pattern pairs are used, with $\boldsymbol{x}$ and $\boldsymbol{t}$ being the input and the output data respectively:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \quad \mathbf{t}^{(1)} = (0.88)$$
$$\mathbf{x}^{(2)} = (0.1, 0.6), \quad \mathbf{t}^{(2)} = (0.82)$$
$$\mathbf{x}^{(3)} = (0.9, 0.4), \quad \mathbf{t}^{(3)} = (0.57)$$



Biases are treated here as connection weights that are always multiplied by (-1) through a neuron to avoid special case calculation for biases. Each neuron uses a unipolar sigmoid activation function given by:

$$o = f(tot) = \frac{1}{1 + e^{-\lambda tot}}, \text{using } \lambda = 1, \text{then } f'(tot) = o(1 - o)$$

Solution

Example 5.1

To illustrate this powerful algorithm, we apply it for the training of the following network shown in Figure 5.4. The following three training pattern pairs are used, with **x** and **t** being the input and the output data respectively:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \quad \mathbf{t}^{(1)} = (0.88),$$

$$\mathbf{x}^{(2)} = (0.1, 0.6), \quad \mathbf{t}^{(2)} = (0.82),$$

$$\mathbf{x}^{(3)} = (0.9, 0.4), \quad \mathbf{t}^{(3)} = (0.57),$$

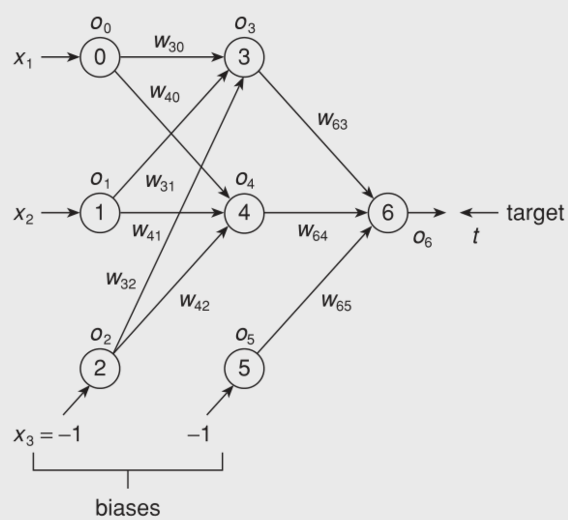


Figure 5.4: Structure of the neural network of Example 5.1

Biases are treated here as connection weights that are always multiplied by −1 through a neuron to avoid special case calculation for biases. Each neuron uses a unipolar sigmoid activation function given by:

$$o = f(\text{tot}) = \frac{1}{1 + e^{-\lambda \text{tot}}}, \text{ using } \lambda = 1, \text{ then } f'(\text{tot}) = o(1 - o)$$

Step (1) – Initialization

- Initialize the weights to small random values. We assume all weights are initialized to 0.2; set learning rate to $\eta = 0.2$; set maximum tolerable error to $E_{max} = 0.01$ (i.e., 1% error); set current error value to $E = 0$; set current training pattern to $k = 1$.

Training Loop – Loop (1)

Step (2) – Apply input pattern

- Apply the 1st input pattern to the input layer:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \mathbf{t}^{(1)} = (0.88), \text{ then, } o_0 = \mathbf{x}_1 = 0.3; o_1 = \mathbf{x}_2 = 0.4; o_2 = \mathbf{x}_3 = -1$$

Step (3) – Forward propagation

- Propagate the signal forward through the network:

$$o_3 = f(w_{30}o_0 + w_{31}o_1 + w_{32}o_2) = 0.4850$$
$$o_4 = f(w_{40}o_0 + w_{41}o_1 + w_{42}o_2) = 0.4850$$
$$o_5 = -1$$
$$o_6 = f(w_{63}o_3 + w_{64}o_4 + w_{65}o_5) = 0.4985$$

Step (4) – Output error measure

- Compute the error value E and the error signal δ_6 of the output layer:

$$E = \tfrac{1}{2}(t - o_6)^2 + E = 0.0728$$
$$\delta_6 = f'(\text{tot}_6)(t - o_6)$$
$$= o_6(1 - o_6)(t - o_6)$$
$$= 0.0954$$

Step (5) – Error backpropagation

- Propagate the errors backward to update the weights and compute the error signals of the preceding layers.

Third layer weight updates:

$$\Delta w_{63} = \eta \delta_6 o_3 = 0.0093 \qquad w_{63}^{\text{new}} = w_{63}^{\text{old}} + \Delta w_{63} = 0.2093$$

$$\Delta w_{64} = \eta \delta_6 o_4 = 0.0093 \qquad w_{64}^{\text{new}} = w_{64}^{\text{old}} + \Delta w_{64} = 0.2093$$

$$\Delta w_{65} = \eta \delta_6 o_5 = -0.0191 \qquad w_{65}^{\text{new}} = w_{65}^{\text{old}} + \Delta w_{65} = 0.1809$$

Second layer error signals:

$$\delta_3 = f_3'(\text{tot}_3) \sum_{i=6}^{6} w_{i3} \delta_i = o_3(1 - o_3) w_{63} \delta_6 = 0.0048$$

$$\delta_4 = f_4'(\text{tot}_4) \sum_{i=6}^{6} w_{i4} \delta_i = o_4(1 - o_4) w_{64} \delta_6 = 0.0048$$

Second layer weight updates:

$$\Delta w_{30} = \eta \delta_3 o_0 = 0.00028586 \qquad w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2003$$

$$\Delta w_{31} = \eta \delta_3 o_1 = 0.00038115 \qquad w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2004$$

$$\Delta w_{32} = \eta \delta_3 o_2 = -0.00095288 \qquad w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.1990$$

$$\Delta w_{40} = \eta \delta_4 o_0 = 0.00028586 \qquad w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2003$$

$$\Delta w_{41} = \eta \delta_4 o_1 = 0.00038115 \qquad w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2004$$

$$\Delta w_{42} = \eta \delta_4 o_2 = -0.00095288 \qquad w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.1990$$

Training Loop – Loop (2)

Step (2) – Apply the 2nd input pattern

Apply the 2nd input pattern to the input layer:

$\mathbf{x}^{(2)} = (0.1, 0.6)$, $\mathbf{t}^{(2)} = (0.82)$, then, $o_0 = 0.1$, $o_1 = 0.6$, $o_2 = -1$

Step (3) – Forward propagation

$$o_3 = f(w_{30} o_0 + w_{31} o_1 + w_{32} o_2) = 0.4853$$

$$o_4 = f(w_{40} o_0 + w_{41} o_1 + w_{42} o_2) = 0.4853$$

$$o_5 = -1$$

$$o_6 = f(w_{63} o_3 + w_{64} o_4 + w_{65} o_5) = 0.5055$$

Step (4) – Output error measure

$$E = \tfrac{1}{2}(t - o_6)^2 + E = 0.1222$$

$$\delta_6 = o_6(1 - o_6)(t - o_6) = 0.0786$$

Step (5) – Error backpropagation

Third layer weight updates:

$$\Delta w_{63} = \eta\delta_6 o_3 = 0.0076 \qquad w_{63}^{\text{new}} = w_{63}^{\text{old}} + \Delta w_{63} = 0.2169$$
$$\Delta w_{64} = \eta\delta_6 o_4 = 0.0076 \qquad w_{64}^{\text{new}} = w_{64}^{\text{old}} + \Delta w_{64} = 0.2169$$
$$\Delta w_{65} = \eta\delta_6 o_5 = -0.0157 \qquad w_{65}^{\text{new}} = w_{65}^{\text{old}} + \Delta w_{65} = 0.1652$$

Second layer error signals:

$$\delta_3 = f_3'(\text{tot}_3)\sum_{i=6}^{6} w_{i3}\delta_i = o_3(1-o_3)w_{63}\delta_6 = 0.0041$$
$$\delta_4 = f_4'(\text{tot}_4)\sum_{i=6}^{6} w_{i4}\delta_i = o_4(1-o_4)w_{64}\delta_6 = 0.0041$$

~~Second~~ First layer weight updates:

$$\Delta w_{30} = \eta\delta_3 o_0 = 0.000082169 \qquad w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2004$$
$$\Delta w_{31} = \eta\delta_3 o_1 = 0.00049302 \qquad w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2009$$
$$\Delta w_{32} = \eta\delta_3 o_2 = -0.00082169 \qquad w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.1982$$

$$\Delta w_{40} = \eta\delta_4 o_0 = 0.000082169 \qquad w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2004$$
$$\Delta w_{41} = \eta\delta_4 o_1 = 0.00049302 \qquad w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2009$$
$$\Delta w_{42} = \eta\delta_4 o_2 = -0.00082169 \qquad w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.1982$$

Training Loop – Loop (3)

Step (2) – Apply the 3rd input pattern to the input layer

$\mathbf{x}^{(2)} = (0.9, 0.4)$, $\mathbf{t}^{(2)} = (0.57)$, then, $o_0 = 0.9$, $o_1 = 0.4$, $o_2 = -1$

Step (3) – Forward propagation

$$o_3 = f(w_{30}o_0 + w_{31}o_1 + w_{32}o_2) = 0.5156$$
$$o_4 = f(w_{40}o_0 + w_{41}o_1 + w_{42}o_2) = 0.5156$$
$$o_5 = -1$$
$$o_6 = f(w_{63}o_3 + w_{64}o_4 + w_{65}o_5) = 0.5146$$

Step (4) – Output error measure

$$E = \tfrac{1}{2}(t - o_6)^2 + E = 0.1237$$
$$\delta_6 = o_6(1-o_6)(t-o_6) = 0.0138$$

Required Steps for Backpropagation Learning Algorithm

- **Step 1:** Initialize weights and thresholds to small random values.
- **Step 2:** Choose an input-output pattern form the training input-output data set:

$$(x(k), t(k))$$

- **Step 3:** Propagate the k^{th} signal forward through the network and compute the output values or all i neurons at every layer (ℓ) using:

$$o_i^{\ell}(k) = f\left(\sum_{p=0}^{n_{\ell-1}} w_{ip}^{(\ell)} o_p^{(\ell-1)}\right)$$

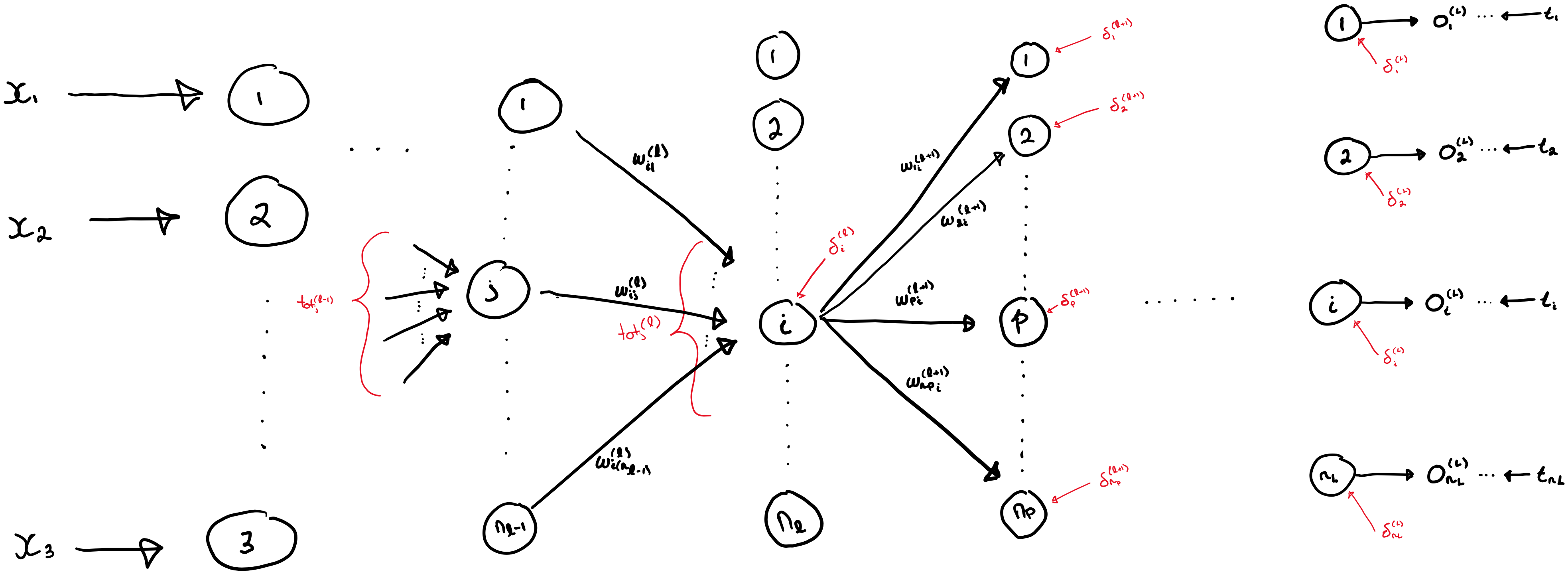
- **Step 4:** Compute the total error value $E = E(k) + E$ and the error signal $\delta_i^{(L)}$ using formulae:

$$\delta_i^{(L)} = \left[t_i - o_i^{(L)}\right]\left[(tot)_i^{(L)}\right]$$

- **Step 5:** Update the weights according to:

$$\Delta w_{ij}^{(\ell)} = -\eta \delta_i^{(\ell)} o_j^{(\ell-1)}, \quad \text{for } \ell = \text{L}, \dots, 1 \quad \text{using}$$
$$\delta_i^{(L)} = \left[t_i - o_i^{(L)}\right]\left[f'(tot)_i^{(L)}\right] \quad \text{and proceeding backward using}$$
$$\delta_i^{(\ell)} = o_i^{(\ell)}\left(1 - o_i^{(\ell)}\right)\sum_{p=1}^{n_\ell} \delta_p^{(\ell+1)} w_{pi}^{(\ell+1)} \quad \text{for } \ell < L$$

- **Step 6:** Repeat the process starting from step 2 using another exemplar. Once all exemplars have been used, we then reach what is known as one epoch training.
- **Step 7:** Check is the cumulative error E in the output layer has become less than a predetermined value. If so, we say the network has been trained. If not, repeat the whole process for one more epoch.



x1
x2
x3
1
2
3
1
j
j
1
2
i
1
2
p
1
2
i

$$\Delta w_{ij}^{(\ell)} = \eta \delta_i^{(\ell)} o_i^{(\ell-1)}$$

Error signal:

$$\delta_i^{(\ell)} = f'\left(tot_i^{(\ell)}\right)\left[\delta_1^{(\ell+1)} w_{1i}^{(\ell+1)} + \delta_2^{(\ell+1)} w_{2i}^{(\ell+1)} + \cdots + \delta_p^{(\ell+1)} w_{pi}^{(\ell+1)} + \cdots + \delta_{n_p}^{(\ell+1)} w_{n_p i}^{(\ell+1)}\right]$$

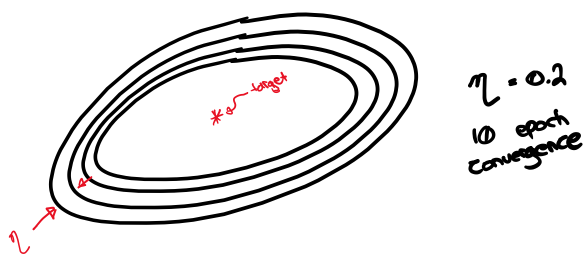
$$\delta_i^{(\ell)} = f'\left(tot_i^{(\ell)}\right) \sum_{p=1}^{n_p} \delta_p^{(\ell+1)} w_{p_i}^{(\ell+1)}$$

For a sigmoid AF, there is a special case:

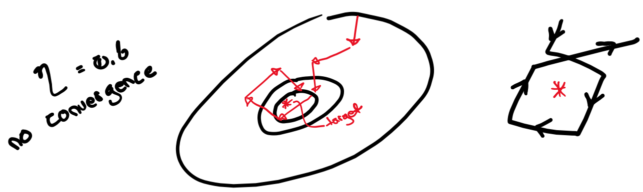
$$f' = f(1-f) \rightarrow o^{(\ell)}(1-o^{(\ell)})$$

4.6 Momentum

When η is small, the convergence towards the target is slow:



Conversely, when η is large, it can miss the target (convergence not met)



$E_{min} = 0.05$

$$\Delta \vec{w}^{(\ell)}(k+1) = -\eta \frac{\partial E(k)}{\partial \vec{w}^{(\ell)}} \nu \Delta \vec{w}_{(k)}^{(\ell)}$$

momentum

$\Delta \vec{w}^{(\ell)}$
$\nu \in [0, 1]$
$\nu = 0.8, 0.9$

Example 5.2

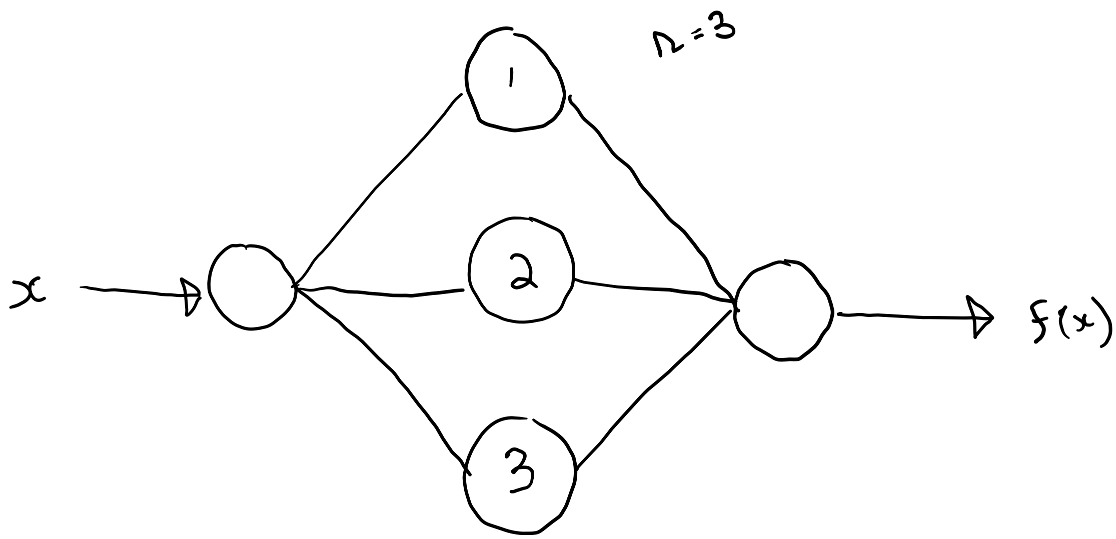
Effect of hidden nodes on function approximation

To illustrate the effects of the number of hidden neurons on the approximation capabilities of the MLP, we use here the simple function $f(x)$ given by:

$$f(x) = x \sin(x)$$

Six input/output samples were selected from the range [0:10] of the variable x. The first run was made for a network with three hidden nodes. The results are shown in Figure 5.6(a). Another run was made for a network with five (Figure 5.6(b)) and 20 (Figure 5.6(c)) nodes respectively. From the result of the simulation, one may conclude that a higher number of nodes is not always better as is seen in Figure 5.6(c). This is mostly due to the fact that a network with this structure has overinterpolated in between the samples and we say that the network was overtrained. This happens when the network starts to memorize the patterns instead of interpolating between them. In this series of simulations the best match with the original curve was obtained with a network having five hidden nodes. It seems here that this network (with five nodes) was able to interpolate quite well the nonlinear behavior of the curve. A smaller number of nodes didn't permit a faithful approximation of the function given that the nonlinearities induced by the network were not enough to allow for adequate interpolation between samples.

$$f(x) = x \sin(x)$$
$$x = 0 \sim 10$$



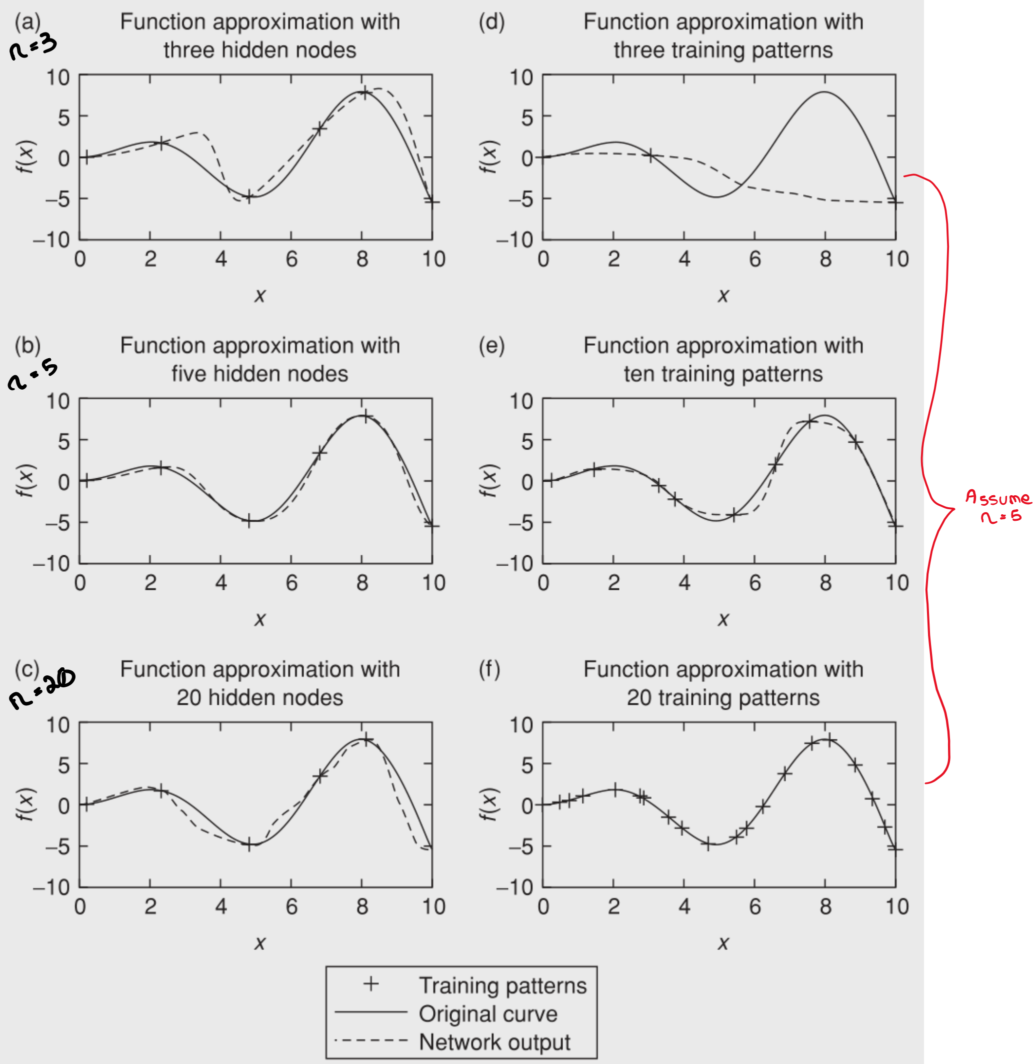


Figure 5.6 a, b, c: Effect of the number of hidden nodes on MLP approximation of $f(x)$

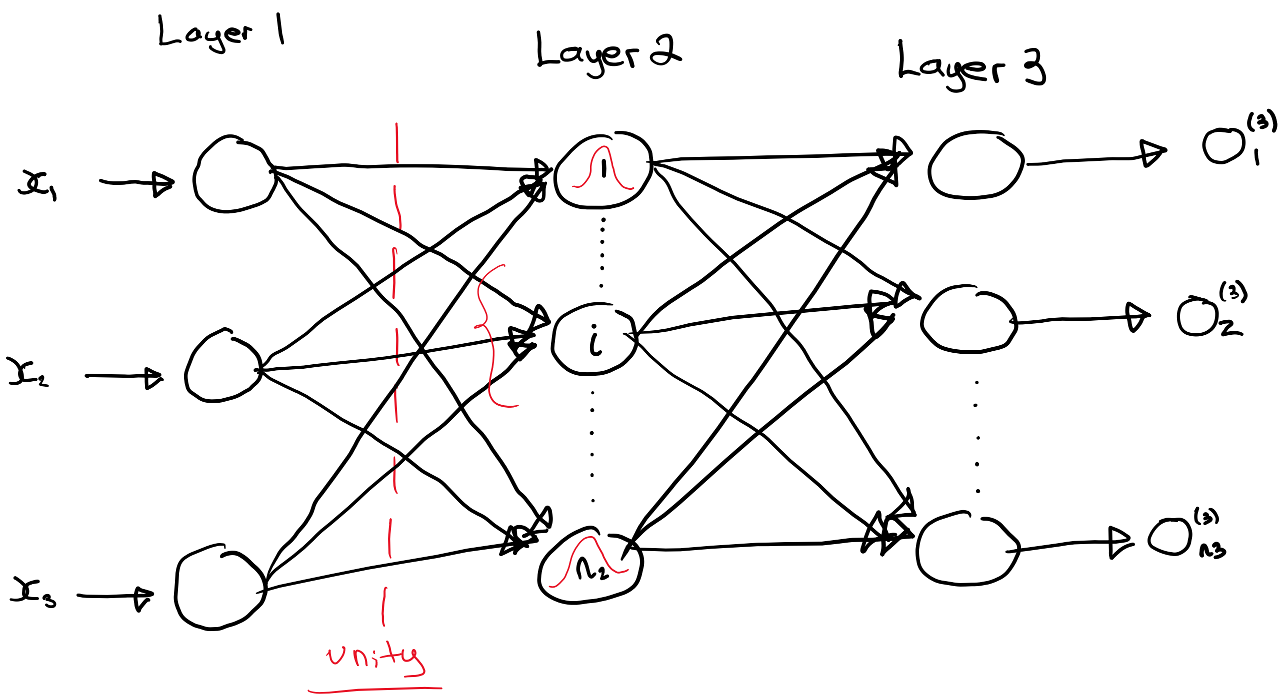
Figure 5.6 d, e, f: Effect of the number of training patterns on MLP approximation of $f(x)$

Using more neurons in the hidden layer doesn't necessarily improve the performance of the system, but using more training data pairs improves system performance.

4.7 Radial Basis Function Neural Network (RBF NN)

- Special case of a feedforward neural network

1. 3 Layer FF NN



2. Unity line weights between (neurons) layer 1 and layer 2 (they have the same value).

3. AFs in the neurons in hidden layer are kernel functions.

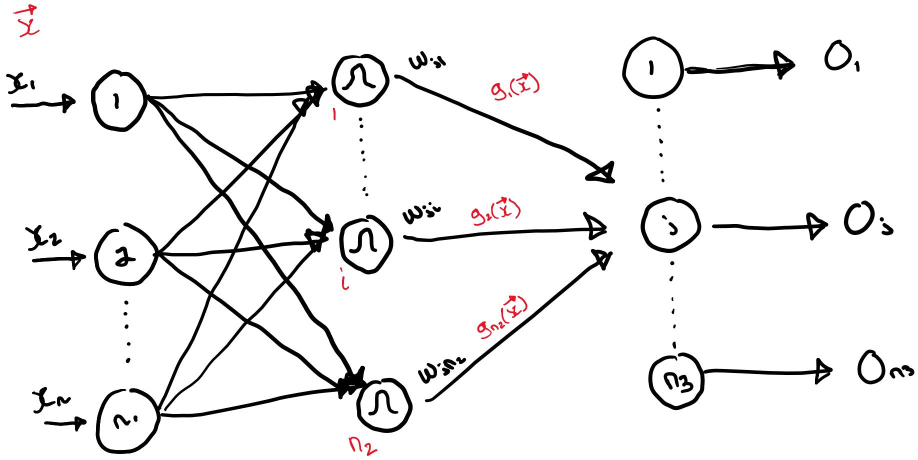
- Gaussian function:

$$g_i(\vec{x}) = e^{\frac{-||\vec{x}-\vec{v}_i||^2}{2\sigma_i^2}}$$

$\vec{x}$ = input vector
$\vec{v}_i$ = center vector
σ_i = spread parameter

- Logic function:

$$g_i(\vec{x}) = \frac{1}{1 + e^{\frac{-||\vec{x}-\vec{v}_i||^2}{2\sigma_i^2}}}$$



Output:

$$o_j(\vec{x}) = g_1(\vec{x})w_{j1} + \cdots + g_i(\vec{x})w_{ji} + \cdots + g_{n_2}(\vec{x})w_{jn_2} \quad ; \quad j = 1, 2, \ldots, n_3$$

$$o_j(\vec{x}) = \sum_{i=1}^{n_2} w_{ji} * g_i(\vec{x})$$

Training:

- Parameters in the hidden neuron AFs (centers and spreads)
- Link weights between the hidden layer & output layer

Note:

A Radial Basis Function (RBF) neural network is a neuro-fuzzy system

Chapter 5: Neuro-Fuzzy Systems

5.1 Introduction

	Fuzzy logic	**Neural networks**
Representation	Linguistic description of knowledge	Knowledge distributed within computational units
Adaption	Some adaptation	Adaptive
Knowledge Representation	Explicit and easy to interpret	Implicit and difficult to interpret
Learning	Non-existent	Excellent tools for imparting learning
Verification	Easy and efficient	Not straightforward (“black box” reasoning)

Integrated systems of fuzzy logic (FL) and neural networks (NN)

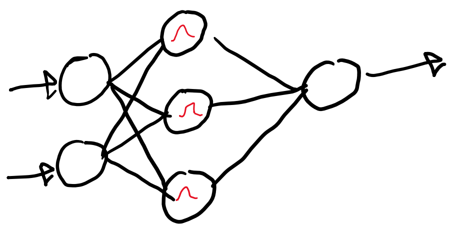
1. Neuro-fuzzy (NF) system
   FL parameters can be trained by using NN training methods (back propagation, etc.)
2. Fuzzy-neuro system (RBF)
   Neural network, but some neurons are fuzzified
3. Neural fuzzy systems
   Just a simple combination of FL and NN (separate systems utilized in series)

NN: universal approximators
- Desired accuracy

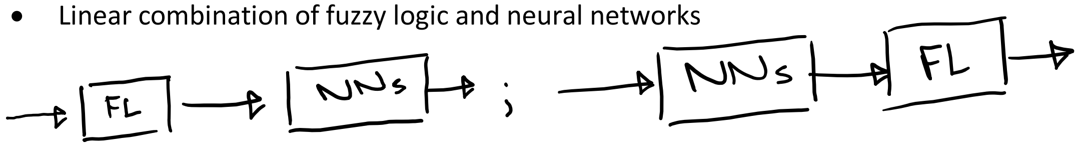
1) Neuro-fuzzy
- Fuzzy logic system with neural network training

2) Fuzzy neural
- Neural network, some neurons are fuzzified

e.g. RBF NN (Radial basis function neural network)



3) Neural fuzzy systems
- Linear combination of fuzzy logic and neural networks



5.2 Adaptive Neuro-Fuzzy Inference Systems (ANFIS)

Consider the following general model:

Sugeno fuzzy model (TSK-1):

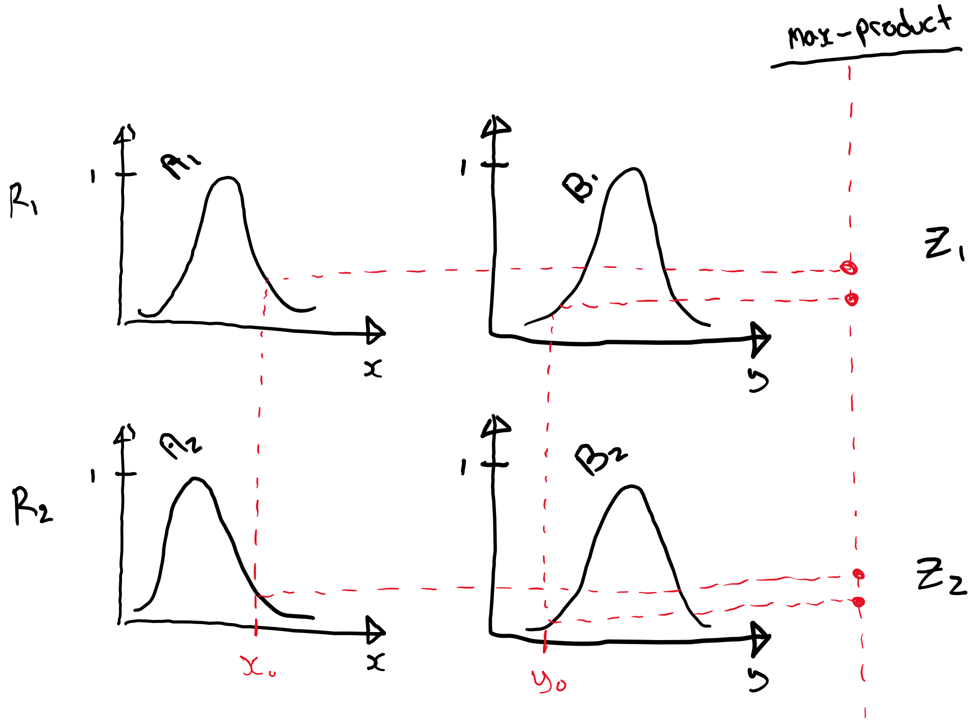
Consider a system with two inputs (x, y), each having two memberships functions, and one output z

$\mathcal{R}_1$: If (x is A_1) and (y is B_1) then ($z_1 = p_1 x + q_1 y + r_1$)

$\mathcal{R}_2$: If (x is A_2) and (y is B_2) then ($z_2 = p_2 x + q_2 y + r_2$)

$A_1, A_2, B_1, B_2 \sim$ fuzzy sets
$p_1, p_2, q_1, q_2, r_1, r_2 \sim$ parameters

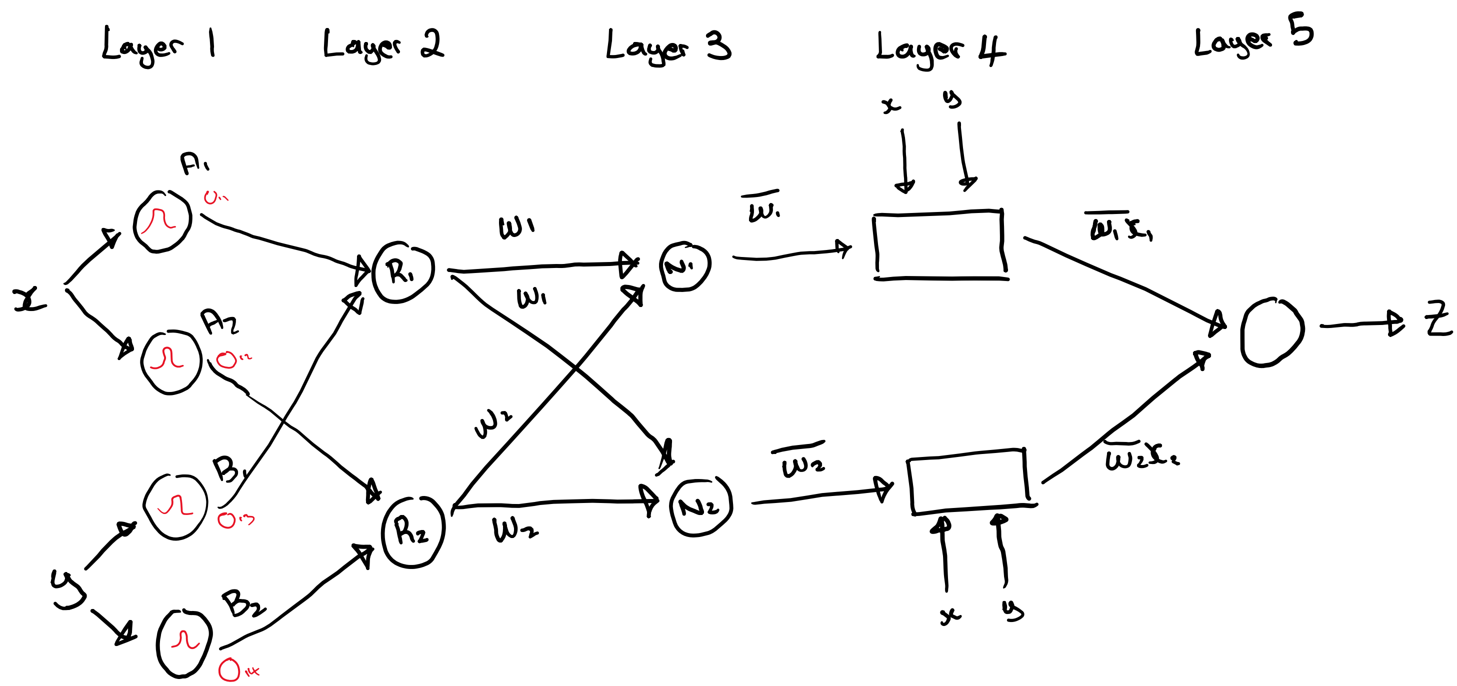
For our project, 3 inputs 2 MF's $\sim 2^3 = 8$ rules



$$z_1 = p_1 x + q_1 y + r_1$$

$$z_2 = p_2 x + q_2 y + r_2$$

$$z = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_2} = \left(\frac{w_1}{w_1 + w_2}\right) z_1 + \left(\frac{w_2}{w_1 + w_2}\right) z_2 = \overline{w}_1 z_1 + \overline{w}_2 z_2$$



- **Layer 1**: Input later, adaptive layer

$$o_{11} = \mu_{A1}(x)$$
$$o_{12} = \mu_{A2}(x)$$
$$o_{13} = \mu_{B1}(y)$$
$$o_{14} = \mu_{B2}(y)$$

MF grade

For example, generalized bell membership function MF:

$$\mu_{A1}(x) = \frac{1}{1 + \left|\frac{x - c_i}{a_i}\right|^{2bi}} \quad ; \quad i = 1, 2$$

A sigmoid, gaussian, etc. functions can be utilized instead.

- **Layer 2**: fixed nodes

Firing strength: (e.g., product)

$$w_1 = o_{11} * o_{13} = \mu_{A1}(x) * \mu_{B1}(y)$$
$$w_2 = o_{12} * o_{14} = \mu_{A2}(x) * \mu_{B2}(y)$$

$T - \text{norm can be product, minimum, etc.}$

- **Layer 3:** normalization layer, fixed neurons

$$\overline{w}_1 = \frac{w_1}{w_1 + w_2}$$
$$\overline{w}_2 = \frac{w_2}{w_1 + w_2}$$

- **Layer 4:** nodes are adaptive nodes

Output:

$$\overline{w}_1 z_1 = \frac{w_1}{w_1 + w_2}(p_1 x + q_1 y + r_1)$$
$$\overline{w}_2 z_2 = \frac{w_2}{w_1 + w_2}(p_2 x + q_2 y + r_2)$$

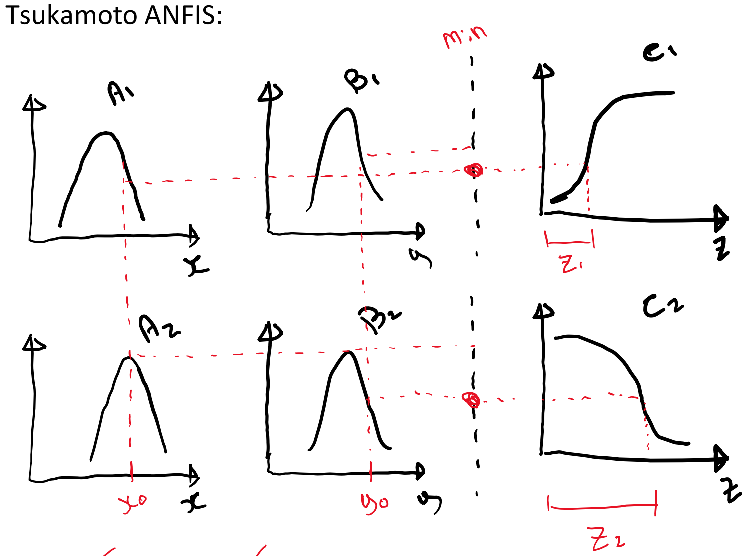
- **Layer 5:** nodes are fixed nodes

$$z = \overline{w}_1 z_1 + \overline{w}_2 z_2$$

Notes:

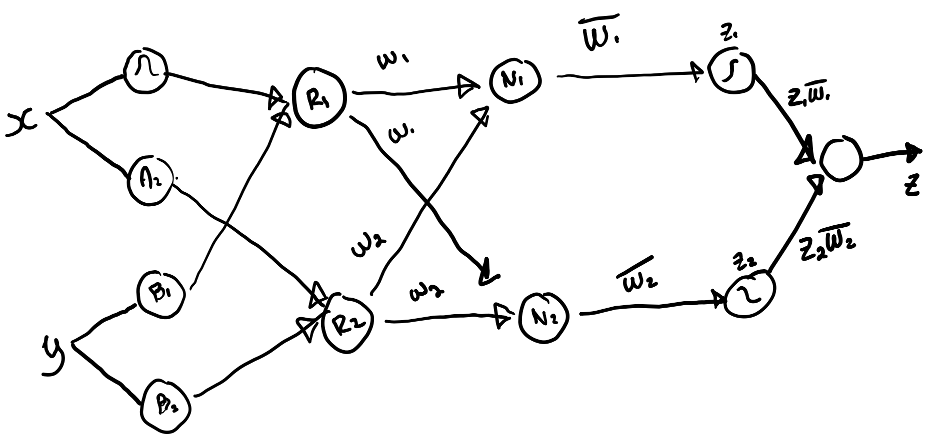
- The structure of the adaptive network is not unique.

Tsukamoto ANFIS:



$$z = \frac{w_1 z_1}{w_1 + w_2} + \frac{w_2 z_2}{w_1 + w_2}$$

$$z = \overline{w}_1 z_1 + \overline{w}_2 z_2$$



- TSK-1

  TSK-0

  Tsukamoto (monotonic function)

  Mamdani model (related to summation of area, difficult to utilize, so not common for making an ANFIS)

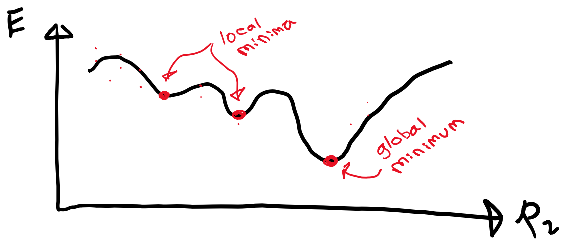
5.6 System Training

- Non-linear MF parameters
- Linear parameters

TSK-1:
- premise MF parameters
- consequent linear parameters

$$z = \overline{w}_1 z_1 + \overline{w}_2 z_2$$
$$= \overline{w}_1(p_1 x + q_1 y + r_1) + \overline{w}_2(p_2 x + q_2 y + r_2)$$
$$= (\overline{w}_1 x)p_1 + (\overline{w}_1 y)q_1 + w_1 r_1 + (\overline{w}_2 x)p_2 + (\overline{w}_2 y)q_2 + w_2 r_2$$

Hybrid training (LSE + GD):
training efficiency ↑
reduce some local minima:



GA (genetic algorithm):

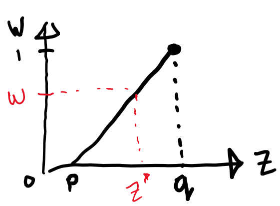
- Forward pass:

premise MF parameters → fixed
optimize linear parameters through LSE (least-squares estimator)

- Backward pass

Linear parameters → fixed
update → MF parameters (these are the non-linear parameters)

Tsukamoto:
Linearized consequent MF:



$$w = p + \frac{1}{q-p} z$$
$$z^* = (w-p)(q-p)$$

Example: (Book 2, Ch. 12, Sec. 6.5) – good example for a forecasting project

MG (Mackey-Glass):

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1x(t)$$

Initial values:

$x(0) = 1.2$
$\tau = 17 \sim 30, dt = 1$

Six-steps-ahead prediction

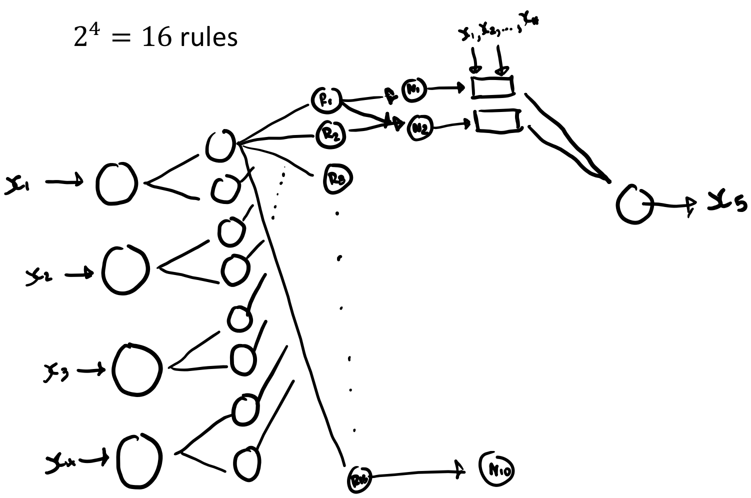
4-inputs
1-output

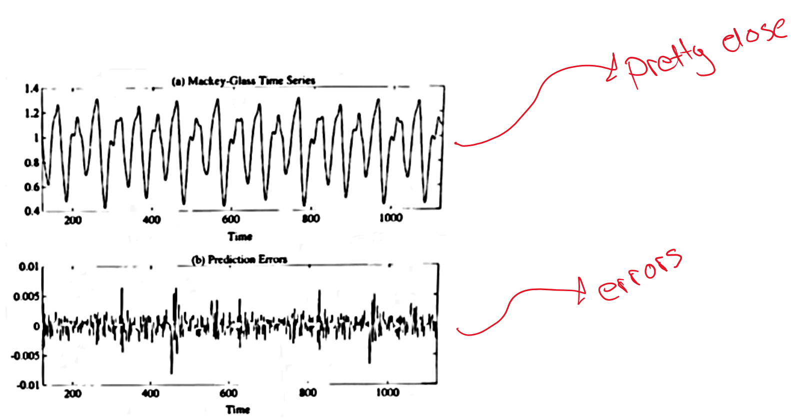
Input: $\{x(t-18),\ x(t-12),\ x(t-6),\ x(t)\}$
x_1 x_2 x_3 x_4

Output: $\{x(t+6)\}$
x_5

Each has 2 MFs $\begin{cases} L \\ S \end{cases}$

$2^4 = 16$ rules





Mackey-glass forecasting data system:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t)$$

$\tau = 17 \sim 30$ (dependent on individual person)
$dt = 1$ (selected)
$x(0) = 1.2$ (dependent on different application)

If you utilized the Mackey-Glass program to generate 2000 data points…

$$d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9 \; \ldots d_{2000}$$

If $s = 1$ (one − step − ahead prediction):

$$1^{st} \text{ training data pair: } (d_1, d_2, d_3; d_4)$$
$$2^{nd} \text{ training data pair: } (d_2, d_3, d_4; d_5)$$
$$3^{rd} \text{ training data pair: } (d_3, d_4, d_5; d_6)$$
$$\vdots$$
$$997^{th} \text{ training data pair: } (d_{997}, d_{998}, d_{999}; d_{1000})$$

If $s = 2$ (two − steps − ahead prediction):

$$(d_1, d_3, d_5; d_7)$$
$$(d_2, d_4, d_6; d_8)$$
$$(d_3, d_5, d_7; d_9)$$
$$\vdots$$
$$(d_{994}, d_{996}, d_{998}; d_{1000})$$

s − steps − ahead prediction:

$$x(t)$$

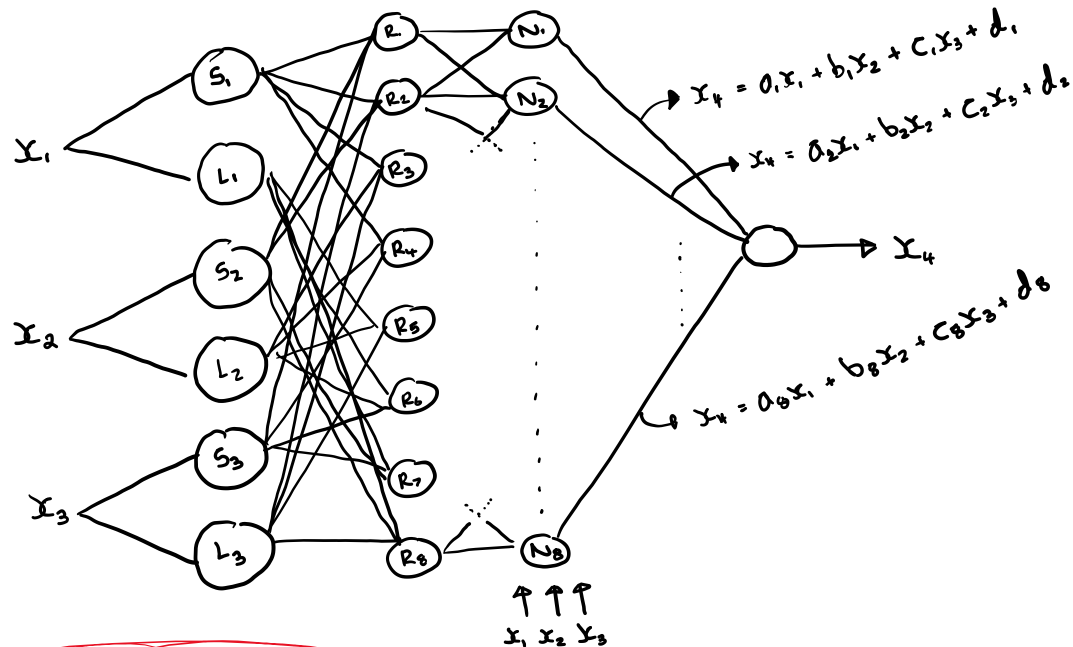
$$\{\underbrace{x(t-2s)}_{x_3},\ \underbrace{x(t-s)}_{x_2},\ \underbrace{x(t)}_{x_1}\ ;\ \underbrace{x(t+s)}_{x_4}\}$$

does order really matter?

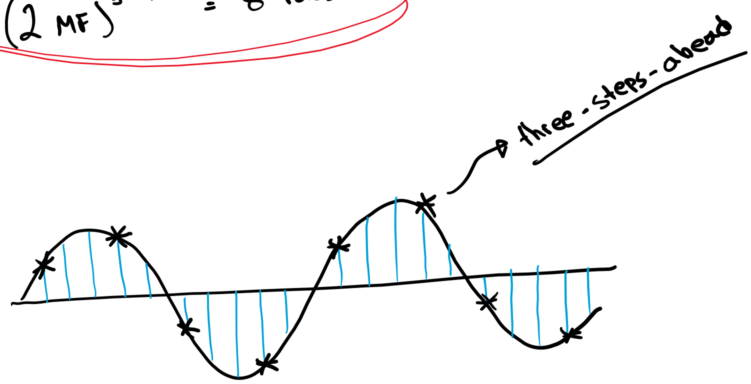
If $s = 6$:

$$\{\,x(t-12)\,,\ \ x(t-6)\,,\ \ x(t)\ \ ;\ \ x(t+6)\}$$

Neural network:



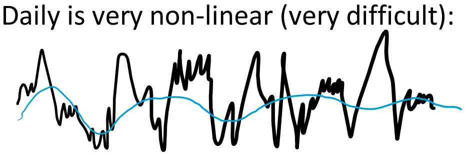
$(2\ MF)^{3\ inputs} = 8$ rules



- Can also do sunspot activity forecasting

RWC: Belgium World Data center
Records from $1700 \sim \text{now}$
Daily, weekly, monthly, annually, etc.

Daily is very non-linear (very difficult):



Weekly, monthly, annually may produce more reliable results (annual is preferred).

In this course we used a hybrid training method:
(a combination least-squares estimator and gradient descent)

1. Initial values of linear and nonlinear parameters. Usually, nonlinear parameters are related to the membership function parameters.
2. Choose an input-output pattern:

$$\{\ \vec{x}(k)\ ;\ t(k)\ \}$$

3. Propagate inputs and calculate the related node output.
4. Calculate the error:

$$E = E(k) + E(k-1)$$

5. Train the linear consequent parameters with non-linear MF parameters fixed.

LSE (least-squares estimator):

$$E(k) = \frac{1}{2}\sum_{i=1}^{n_L}\left(t_j - y_j\right)^2$$

For offline training:

$$\vec{\theta} = \left(\underline{A}^T\underline{A}\right)^{-1}\underline{A}^T\vec{y}$$
$$\vec{\theta} = \{p_1, q_1, r_1, p_2, q_2, r_2\}^T$$

For 8 rules:
Linear parameters: $4 \text{ x } 8 = 32$

Each sigmoid function has two MF (2 variables)
$2 \text{ x } 6 = 12 \sim$ non-linear parameters

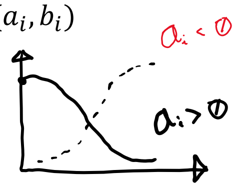
6. Train nonlinear parameters
   Linear parameters are fixed, and non-linear parameters are adjusted.

$$E(k) = \frac{1}{2}\sum_{j}\left(t_j - y_j\right)^2$$

Sigmoid MF:

$$o_i = \mu_A(x_i) = \frac{1}{1 + e^{-a_i(x_i + b_i)}}$$

(a_i, b_i)



$$a_i(k) = a_i(k-1) - \eta_a \frac{\partial E}{\partial a_i}$$
$$b_i(k) = b_i(k-1) - \eta_b \frac{\partial E}{\partial b_i}$$

$$\frac{\partial E}{\partial a_i} = \frac{1}{2}(2)\sum_j (t_j - y_j)(-1)\frac{\partial y_j}{\partial o_i}\frac{\partial o_i}{\partial a_i}$$

$$\frac{\partial o_i}{\partial a_i} = \frac{\partial \mu_A}{\partial a_i} = (-1)\left(1 + e^{-a_i(x_i - b_i)}\right)^{-2}\left(e^{-a_i(x_i - b_i)}\right)[-(x_i - b_i)]$$

$$= \frac{e^{-a_i(x_i - b_i)}(x_i - b_i)}{[1 + e^{-a_i(x_i - b_i)}]^2}$$

$$= d\mathrm{M}ai \text{ (in MATLAB)}$$

Similarly,

$$\frac{\partial E}{\partial b_i} = \frac{1}{2}(2)\sum_j (t_j - y_j)(-1)\frac{\partial y_j}{\partial o_i}\frac{\partial o_i}{\partial b_i}$$

$$\frac{\partial o_i}{\partial b_i} = \frac{\partial \mu_B}{\partial b_i} = (-1)\left(1 + e^{-a_i(x_i - b_i)}\right)^{-2}\left(e^{-a_i(x_i - b_i)}\right)(a_i)$$

$$= \frac{e^{-a_i(x_i - b_i)}(a_i)}{[1 + e^{-a_i(x_i - b_i)}]^2}$$

$$= d\mathrm{M}bi \text{ (in MATLAB)}$$

$$\frac{\partial y_j}{\partial o_i} = dyoi \text{ (in MATLAB)}$$

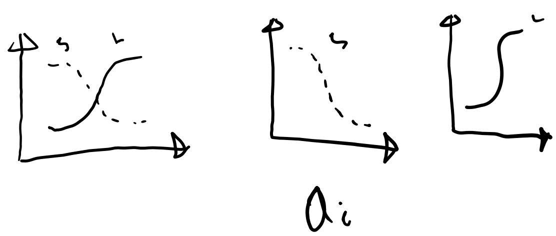
$$\frac{\partial E}{\partial a_i} = dEdai = -\sum_j (t_j - y_j) * dyoi * d\mathrm{M}ai$$

$$\frac{\partial E}{\partial b_i} = dEdbi = -\sum_j (t_j - y_j) * dyoi * d\mathrm{M}bi$$

For example,

$$a_i(k) = a_i(k-1) - \eta_a(dEdai);$$
$$b_i(k) = b_i(k-1) - \eta_b(dEdbi);$$

$i = 1, 2, \ldots, 6$



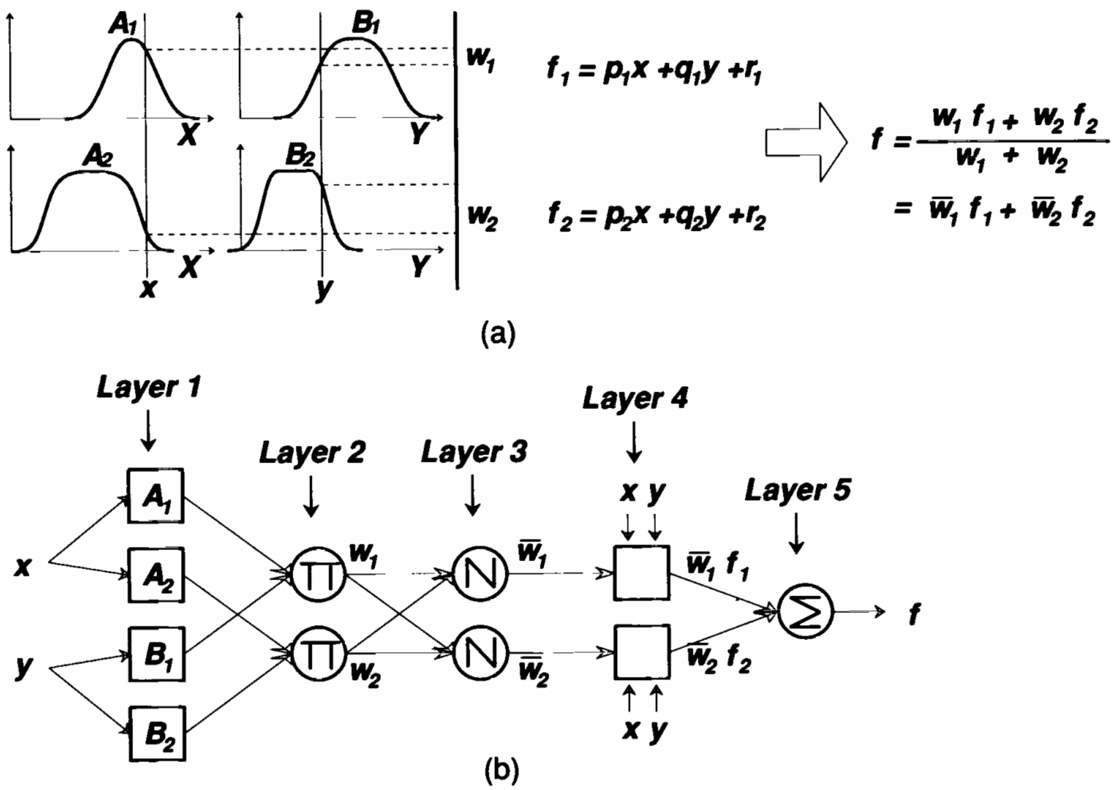
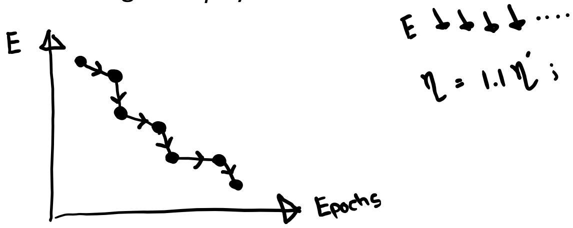


Figure 12.1. *(a) A two-input first-order Sugeno fuzzy model with two rules; (b) equivalent ANFIS architecture.*

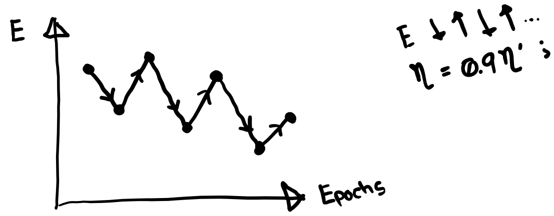
Learning Rules:

The learning rules of ANFIS are then as follows:

1. Propagate all patterns from training set and calculate the optimized consequent parameters using the LSE method, while fixing the antecedent parameters.
2. Propagate all training patterns again and tune (through one epoch only) the antecedent parameters using the LM/Gradient Descent method and backpropagation (as in MLP), while fixing the consequent parameters.
3. If the error was reduced in 4 consecutive steps (heading towards the right direction), then increase the learning rate η by 10%.



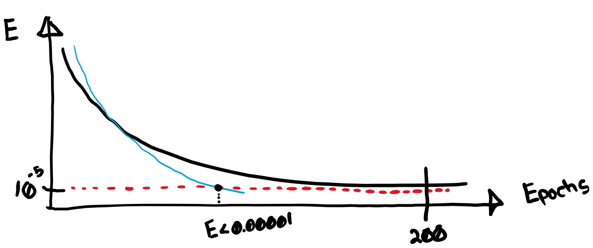
4. If the error in 4 consecutive steps was fluctuating (up and down), then decrease the learning rate η by 10%.



5. Stop if the error is small enough or the maximum number of epochs is reached; otherwise start over from Step 1.

Typically, "small enough" could be:

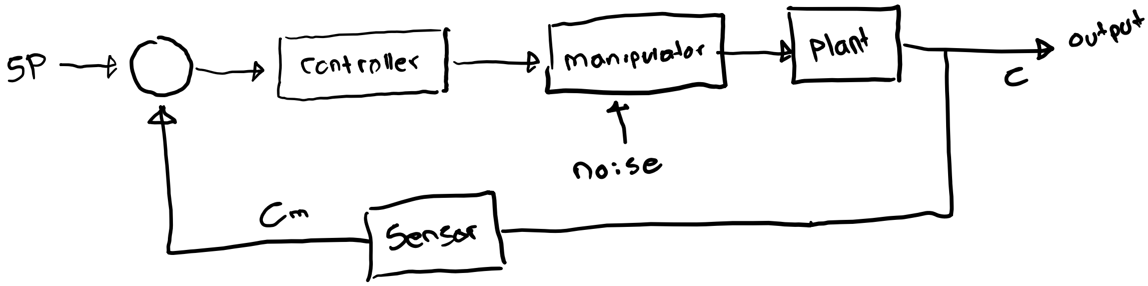
$$E < 0.00001$$
$$(\text{or } 10^{-5})$$



Chapter 6

6.1 Introduction

Classical control systems:

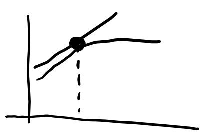


SP – setpoint
C_m – measured error

$\text{error} = SP - C_m \leq \text{threshold}$

Classical control
PI control, PD control, P control, gravitated complex control, PID control

P – proportional
I – integral
D – derivative



H_∞, adaptive, sliding mode
linear systems

If the system (plant) is very non-linear, parameters are time variant (e.g. process control)
environment – noisy

Plant model → linear PDEs
complex non-linear systems
I.C. ~ approximate reasoning

6.2 Fuzzy and NF Control

Fuzzy reasoning → fuzzy logic
fuzzy system parameters → trained

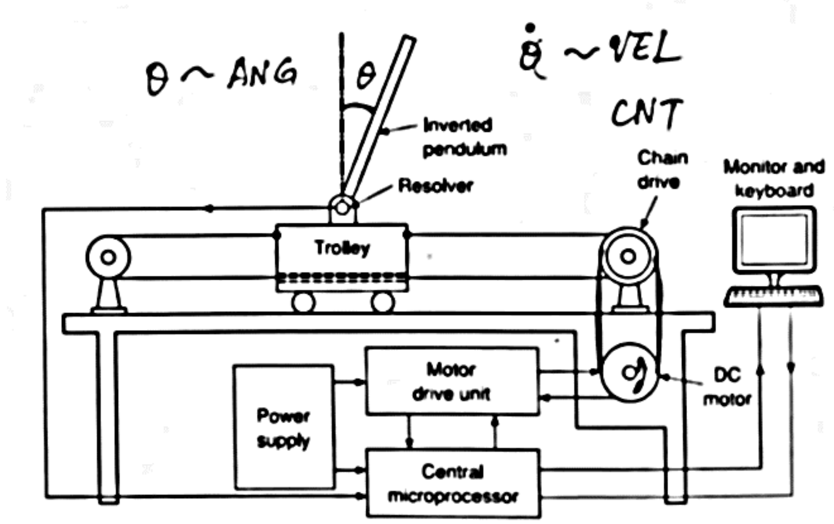
Question 3.3 (Book 1)

Consider the experimental setup of an inverted pendulum shown in Figure P3.3. Suppose that direct fuzzy logic control is used to keep the inverted pendulum upright. The process measurements are the angular position, about the vertical (ANG) and the angular velocity (VEL). The control action (CNT) is the current of the motor driving the positioning trolley. The variable ANG takes two fuzzy states: positive large (PL) and negative large (NL). Their memberships are defined in the support set $[-30°,\ 30°]$ and are trapezoidal. Specifically,

$$\mu_{PL} = \begin{cases} 0 & \text{for } ANG = \{-30°,\ -10°\} \\ \text{linear}(0, 1.0) & \text{for } ANG = \{-10°,\ 20°\} \\ 1 & \text{for } ANG = \{20°,\ 30°\} \end{cases}$$

NF

$$\mu_{NL} = \begin{cases} 0 & \text{for } ANG = \{-30°,\ -20°\} \\ \text{linear}(1.0, 0) & \text{for } ANG = \{-20°,\ 10°\} \\ 1 & \text{for } ANG = \{10°,\ 30°\} \end{cases}$$



The variable VEL takes two fuzzy states PL and NL which are quite similarly defined in the support set $[-60°/s\ \ 60°/s]$. The control inference CNT can take two states: positive large (PL), no change (NC), and negative large (NL). Their membership functions are defined in the support set $[-3A,\ 3A]$ and are either trapezoidal or triangular. Specifically,

$$\mu_{PL} = \begin{cases} 0 & \text{for } CNT = \{-3A,\ 0\} \\ \text{linear}(0, 1.0) & \text{for } CNT = \{0,\ 2A\} \\ 1 & \text{for } CNT = \{2A,\ 3A\} \end{cases}$$

$$\mu_{NC} = \begin{cases} 0 & \text{for } CNT = \{-3A,\ -2A\} \\ \text{linear}(0, 1.0) & \text{for } CNT = \{-2A,\ 0\} \\ \text{linear}(1.0, 0) & \text{for } CNT = \{0,\ 2A\} \\ 0 & \text{for } CNT = \{2A,\ 3A\} \end{cases}$$

$$\mu_{PL} = \begin{cases} 1.0 & \text{for } CNT = \{-3A,\ -2A\} \\ \text{linear}(1.0, 0) & \text{for } CNT = \{-2A,\ 0\} \\ 0 & \text{for } CNT = \{0,\ 3A\} \end{cases}$$

The following four fuzzy rules are used in control:

If ANG is PL and VEL is PL then CNT is NL

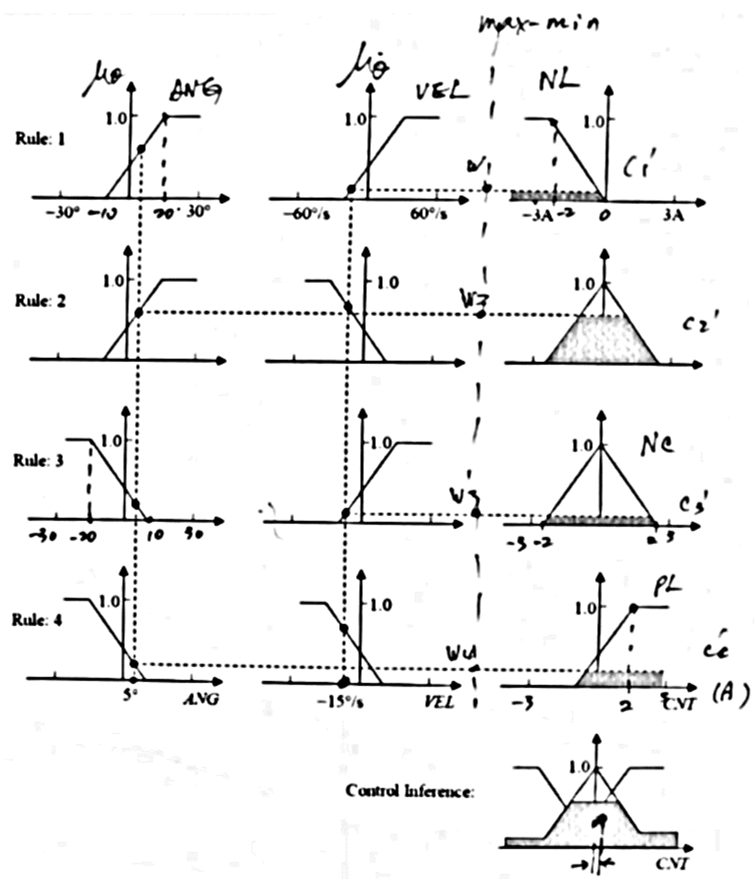
elseif If ANG is PL and VEL is NL then CNT is NC

elseif If ANG is NL and VEL is PL then CNT is NC

elseif If ANG is NL and VEL is NL then CNT is PL

end if

a) Sketch the four rules in a membership function diagram for the purpose of making control inferences using individual rule-based inference.
b) If the process measurements of $ANG = 5°$ and $VEL = 15\ °/s$ are made, indicate on your sketch the corresponding control inference.

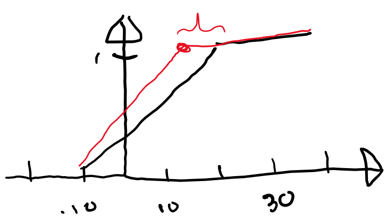


(2) NF control:

Optimize MF parameters and consequent parameters.

If MF parameters are non-linear,

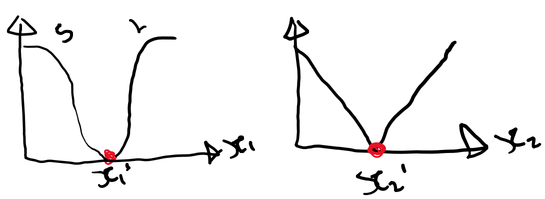
- If the consequent parameters are linear
  Then $\rightarrow GD + LSE$
- If the consequent parameters are non-linear
  Then $\rightarrow GD + GD\ (or\ NG, LM, etc.)$
  Even if GD is used twice, it is still a hybrid method, since they are used independently (and for different system aspects)



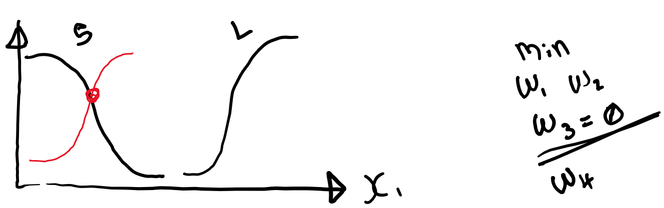
(3) Properties of Fuzzy Control (or NF Control)

1) Completeness
Rule base should be "complete"

Given an input, there is at least one active rule.



2) Continuity
There is no gap between MFs



3) Consistency

No contradictory rules

$\mathcal{R}_3$: if x is L then y is M
...
$\mathcal{R}_9$: if x is L then y is L

4) No interaction

Interaction: rules are coupled

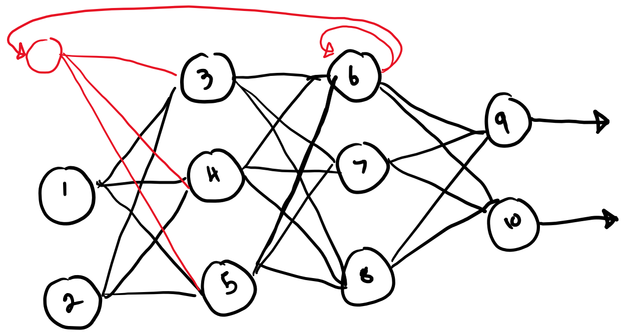
if A_1 and B_1 then C_1 and D_1
else if A_2 and B_2 then C_2 and D_2
else if …

5) Other rules

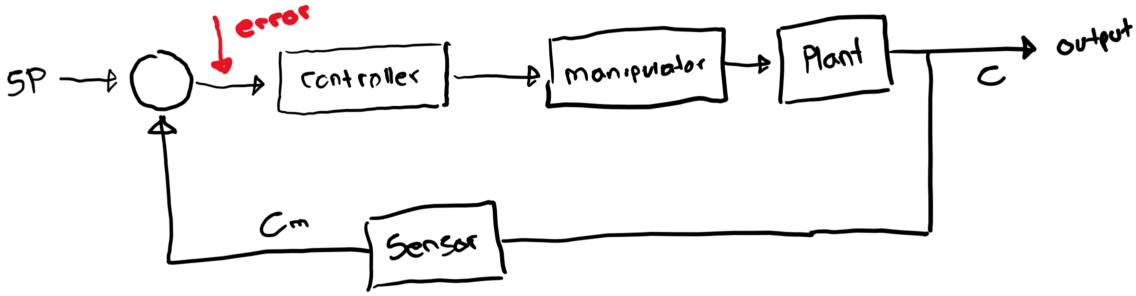
Validity, …, etc.

6.3 NN-based System Identification and Control

ANNs, Recurrent NNs, feed-forward NNs

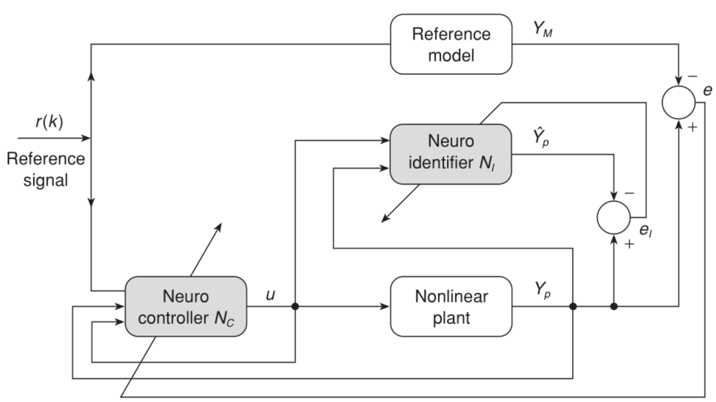


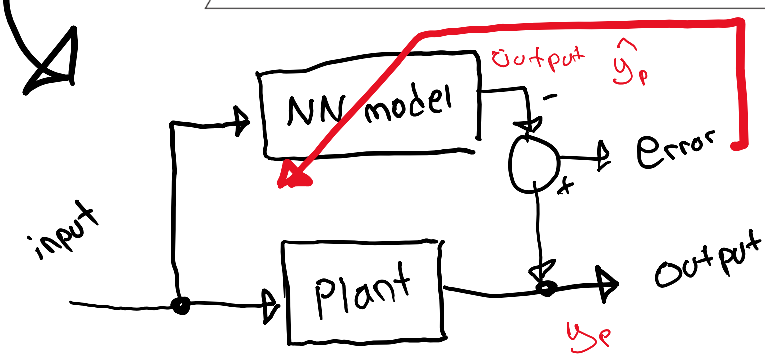
- NN-based controller



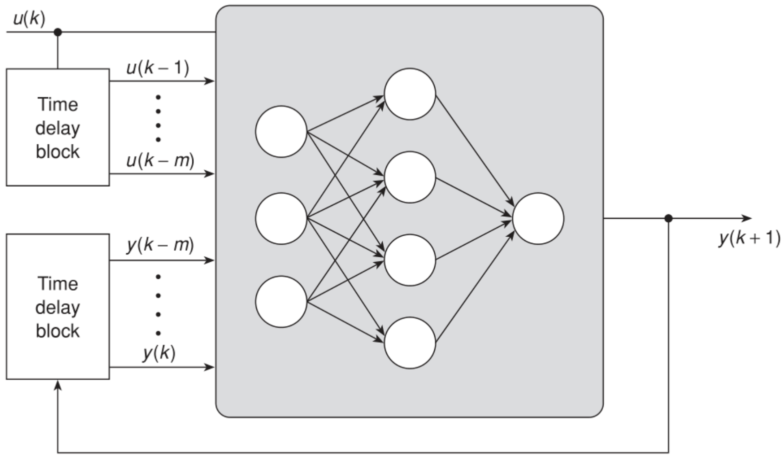
- NNs to model plant

System identification to identify system model.

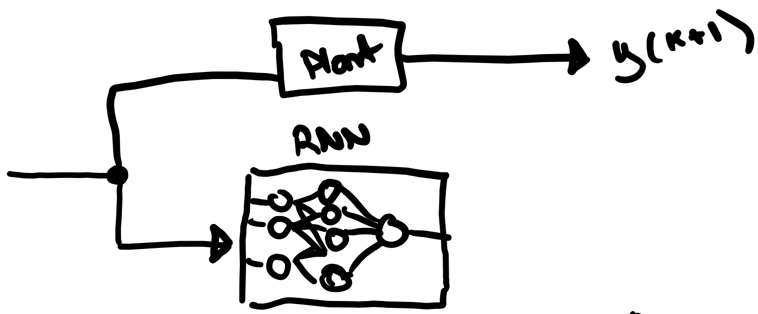




Time delayed recurrent neural network:



- Series-parallel



$$\{u(k), u(k-1), u(k-2), \dots, u(k-m)\}$$

Previous output:

$$\{y(k), y(k-1), y(k-2), \dots, y(k-m)\}$$

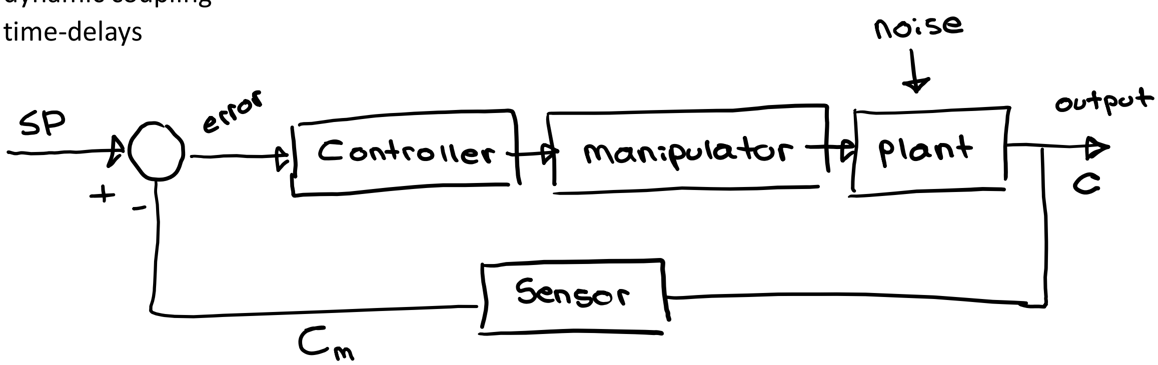
Plant's real output: $y(k+1)$
RRN output: $\hat{y}(k+1)$
Error:

$$e(k+1) = y(k+1) - \hat{y}(k+1)$$

- There is also parallel method (next page)

6.3 NN-based System Identification and Control

Highly non-linear
time-varying
dynamic coupling
time-delays

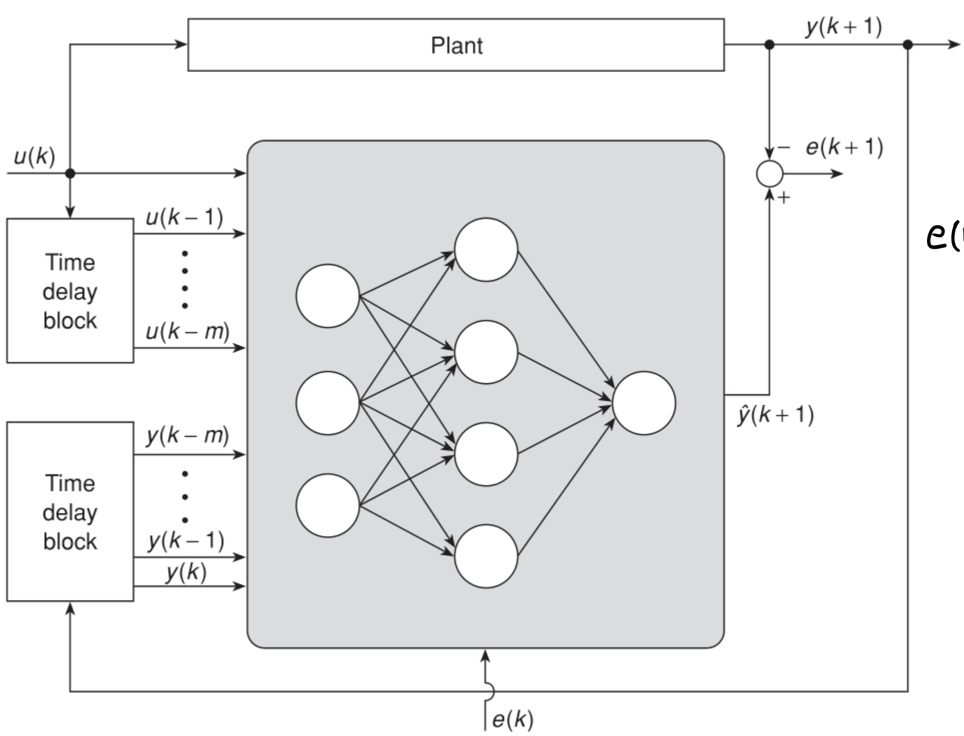


$\text{error} = \text{SP} - \text{C}_\text{m}$

model → plant dynamics
PDE modeling

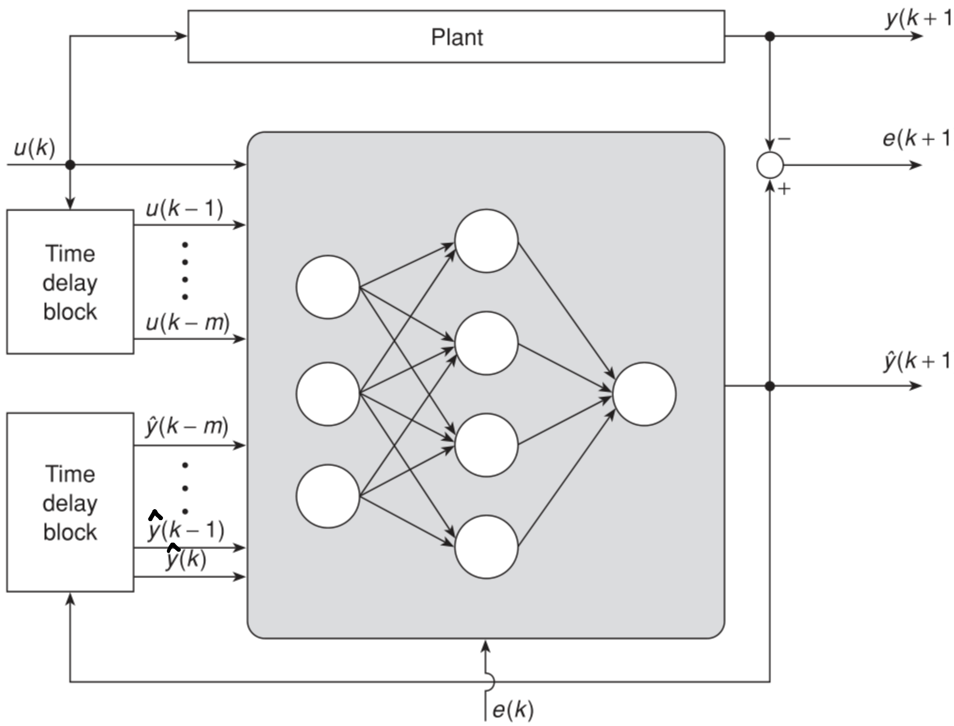
FFNN-based } models
RNN-based

- Series-parallel method:



error
$e(k+1) = y(k+1) - \hat{y}(k+1)$

- Parallel method:



- NN-controllers

PID ~ P gain, I gain, D gain