Advanced Control

January 18, 2013

École Centrale Paris, Châtenay-Malabry, France

Anne Auger

INRIA Saclay – Ile-de-France

Dimo Brockhoff

INRIA Lille – Nord Europe

Course Overview

Date		Topic
Fri, 11.1.2013	DB	Introduction to Control, Examples of Advanced Control, Introduction to Fuzzy Logic
Fri, 18.1.2013	DB	Fuzzy Logic (cont'd), Introduction to Artificial Neural Networks
Fri, 25.1.2013	AA	Bio-inspired Optimization, discrete search spaces
Fri, 1.2.2013	AA	The Traveling Salesperson Problem
Fri, 22.2.2013	AA	Continuous Optimization I
Fri, 1.3.2013	AA	Continuous Optimization II
Fr, 8.3.2013	DB	Controlling a Pole Cart
Do, 14.3.2013	DB	Advanced Optimization: multiobjective optimization, constraints, ...
Tue, 19.3.2013		written exam (paper and computer)

all classes + exam at **8h00-11h15** (incl. a 15min break around 9h30)

Remark to last exercise

All information also available at

`http://researchers.lille.inria.fr/~brockhof/advancedcontrol/`

(exercise sheets, lecture slides, additional information, links, ...)

Remarks to Last Exercise

Result for angle = 0 and pos = 0 showed that

- both "bang-bang" and "continuous force" controller with random k_1, ..., k_4 resulted in 100% stable controllers

I said

"should not be the case"

What happened?

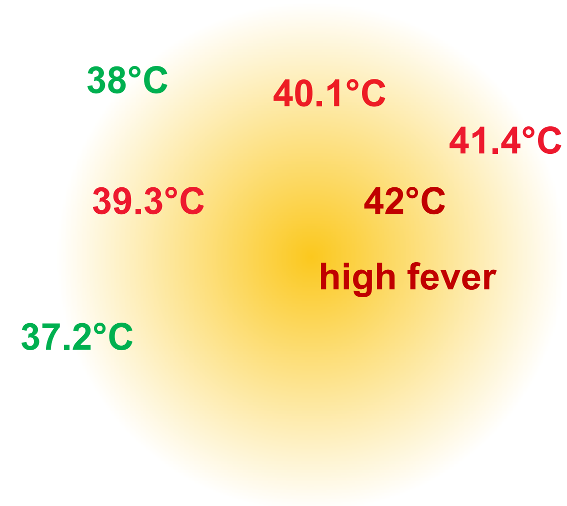
- "bang-bang" was not really bang-bang in my simulation: (F=0 in the beginning)
- "bang-bang" with $F_0 \neq F_m$ gives <<100% stable results

Fuzzy Logic

Fuzzy Logic

- a mathematical tool to deal with uncertainties
- often described as "*computing with words*"[1]
  - e.g. {low, medium, high} instead of {0,1}

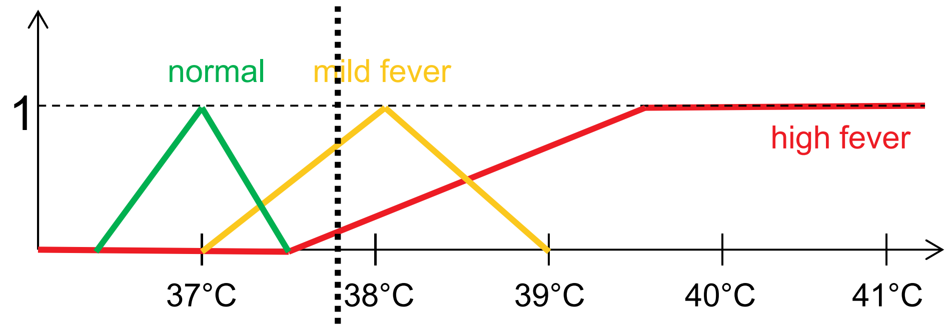
- standard sets: either a in A or a not in A
- fuzzy sets: a in A with probability p_a
- e.g. "high fever with probability 50% and mild fever with 30%"



Fuzzification and Membership Functions

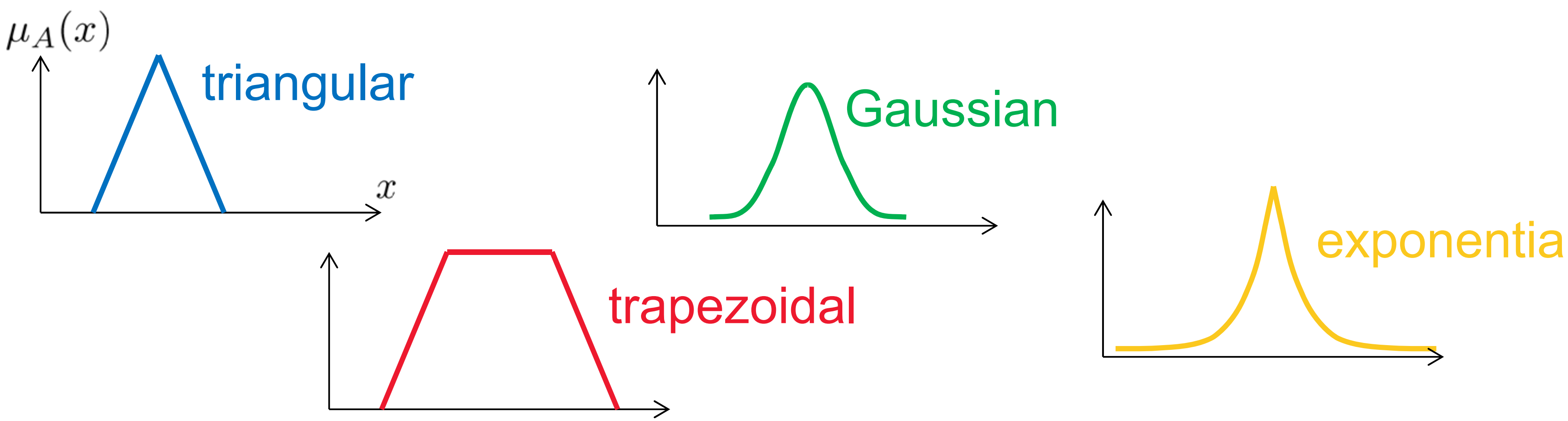
Fuzzification:

= transferring a real-valued variable into a fuzzy one



80% mild and 10% high fever

Several membership functions $\mu_A(x)$ known to do that:



In the end...

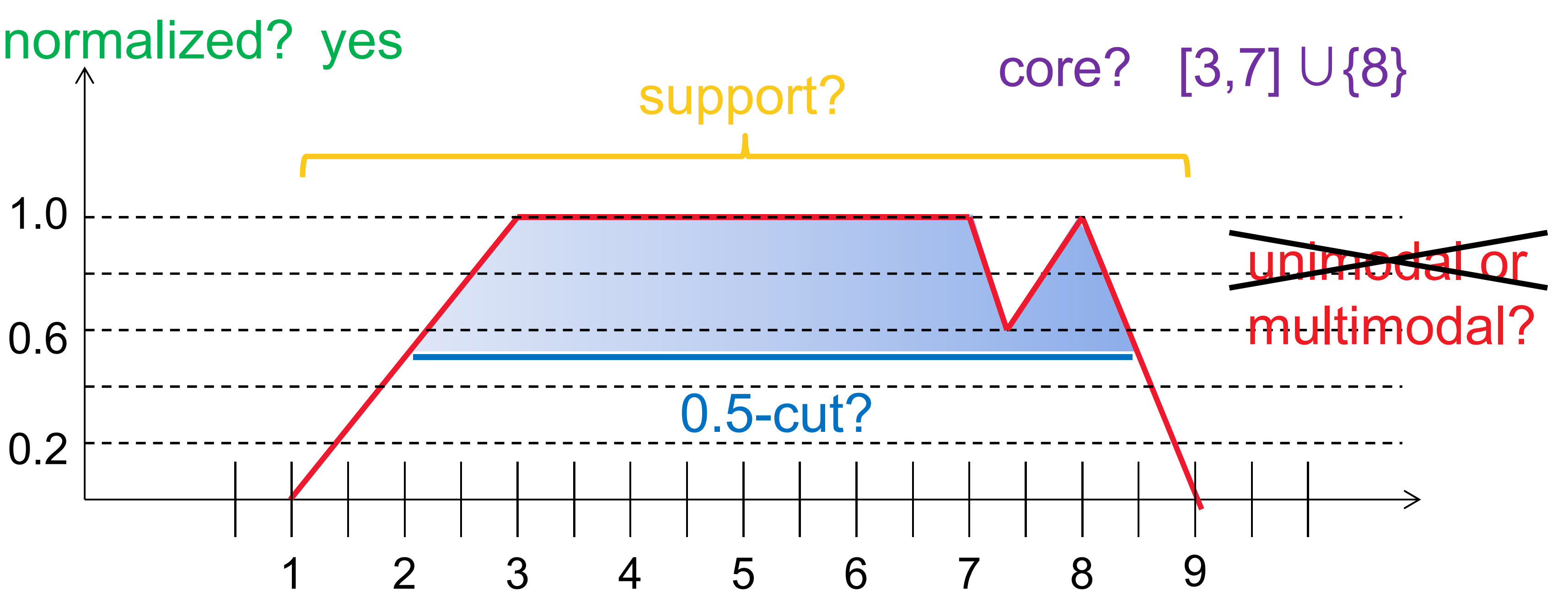
...everything is based on intuition (there are no strict rules)

Properties of Membership Functions

- μ_A is called normalized if its height is 1
- $\{x \,|\, \mu_A(x) = 1\}$ is called the support of ω_n
- $\{x \,|\, \mu_A(x) = 1\}$ is called the core of μ_A
- An α-cut of μ_A is the set $A_\alpha = \{x \,|\, \mu_A(x) \geq \alpha\}$
- If μ_A contains only one maximum, we call μ_A unimodal and A convex
- otherwise, μ_A is called multimodal and A nonconvex

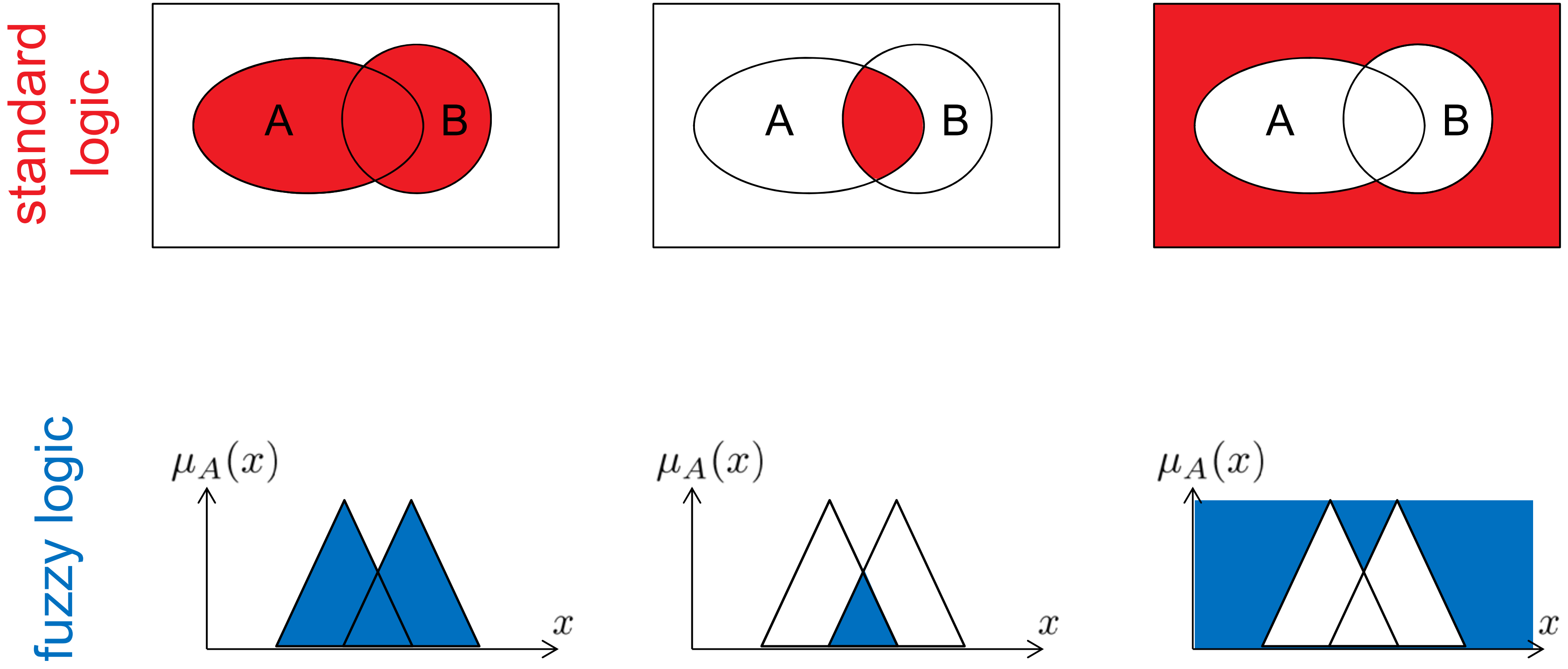
Properties of Membership Functions

- μ_A is called normalized if its height is 1
- $\{x \,|\, \mu_A(x) = 1\}$ is called the support of w_n
- $\{x \,|\, \mu_A(x) = 1\}$ is called the core of μ_A
- An α-cut of μ_A is the set $A_\alpha = \{x \,|\, \mu_A(x) \geq \alpha\}$
- If μ_A contains only one maximum, we call μ_A unimodal and A convex
- otherwise, μ_A is called multimodal and A nonconvex



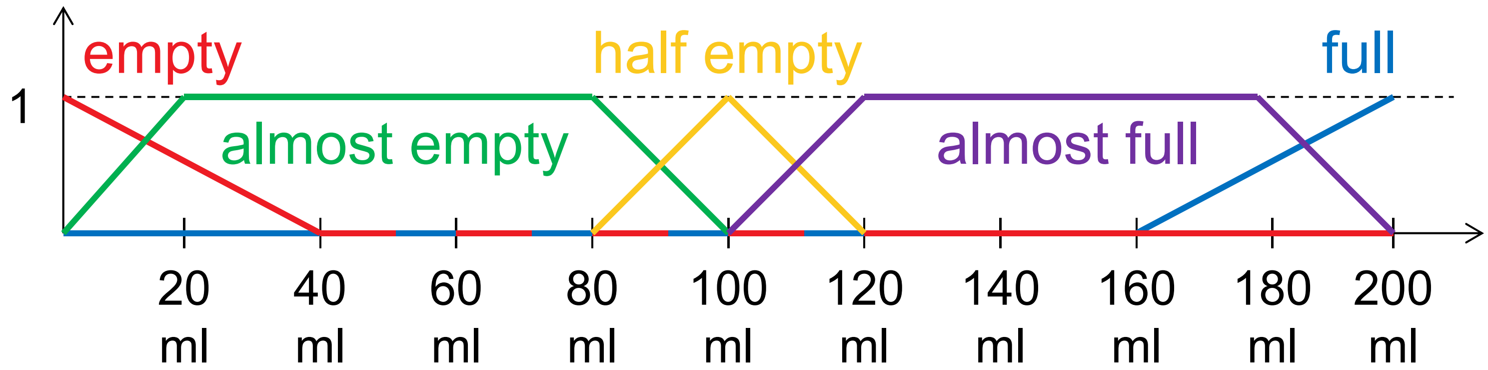
Operations on Fuzzy Sets

Union, intersection, and complement:



union = max intersection = min complement = 1-x

Defuzzifying



How do we get back “crisp” numbers (fuzzy set → real number)?

- there are many ways of doing it!

Maximum defuzzification: take x^* with $\forall x : \mu_A(x^*) \geq \mu_A(x)$

- simple but not accurate if μ_A multimodal

Centroid defuzzification: $x^* = \dfrac{\int \mu_A(x) x dx}{\int \mu_A(x) dx}$

- very accurate
- might be complicated to compute
- often used

Fuzzy Logic: Inferring Statements

Classical Logic:

- IF p THEN q
- equivalent to $\neg p \vee q$

	q = true	q = false
p = true	true	false
p = false	true	true

Fuzzy Logic:

- not so easy with fuzzy sets
  - interpretation as $\neg p \vee q$ results in some undesired effects
  - hence, rather "inference" than implication (for math. reasons)
- in general, implication is a function $\mu(x,y) = \Phi(\mu_A(x), \mu_B(y))$
- > 40 different implication rules proposed
- here, we consider only three (the easy and most used ones)

Fuzzy Logic: Inferring Statements

The sharp implication:

- $$\mu(x,y) = \Phi(\mu_A(x), \mu_B(y)) = \begin{cases} 1 & \text{if } \mu_A(x) \leq \mu_B(y) \\ 0 & \text{else} \end{cases}$$
- intuition: if Y and Y are crisp sets, then $X \Rightarrow Y$ iff $X \subseteq Y$

	q=0	q=0.5	q=1
p=0	1	1	1
p=0.5	0	1	1
p=1	0	0	1

Mamdani's inference[1]:

membership function of implication:

$$\mu(x,y) = \Phi(\mu_A(x), \mu_B(y)) = \min(\mu_A(x), \mu_B(y))$$

only ¼ of corner values equal to 2-valued logic!

inference, no implication

	q=0	q=0.5	q=1
p=0	0	0	0
p=0.5	0	0.5	0.5
p=1	0	0.5	1

[1] E. H. Mamdani. "Application of fuzzy logic to approximate reasoning using linguistic synthesis". IEEE Transactions on Computers, C-26(12):1182–1191, December 1977.

Fuzzy Logic: Inferring Statements

Larsen Product implication[1]:

membership function of implication:

$$\mu(x,y) = \Phi(\mu_A(x), \mu_B(y)) = \mu_A(x) \cdot \mu_B(y)$$

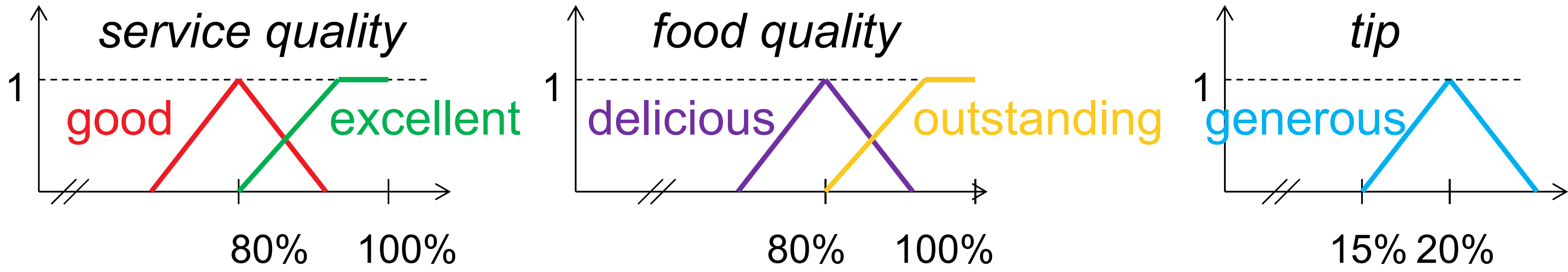
again: only ¼ of corner values equal 2-valued logic!

inference, no implication

	q=0	q=0.5	q=1
p=0	0	0	0
p=0.5	0	0.25	0.5
p=1	0	0.5	1

[1] P. M. Larsen, "Industrial Applications of Fuzzy Logic Control", International Journal of Man-Machine Studies, Vol. 12, No. 1, 1980, pp. 3-10.

Example

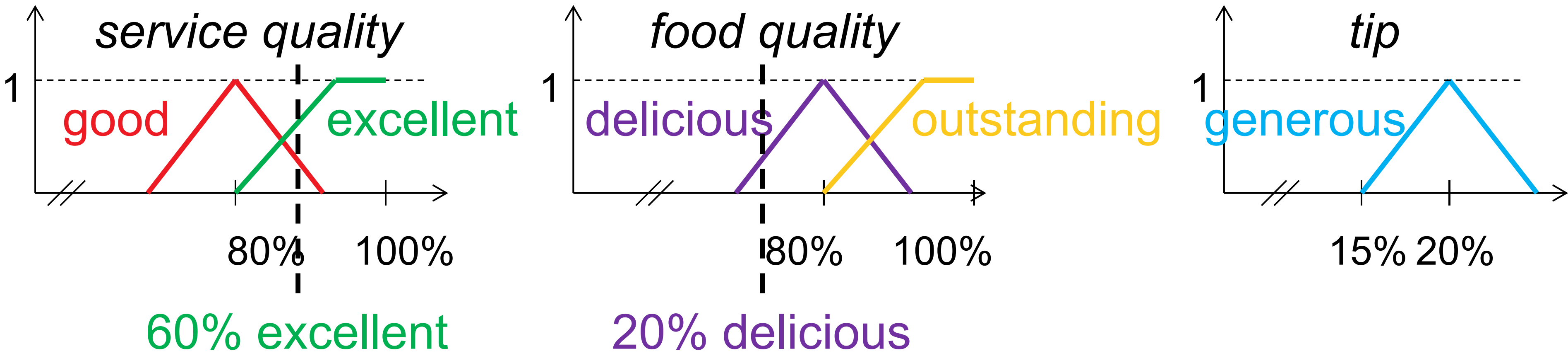


IF service is excellent AND food is delicious THEN tip is generous

What happens for different service and food qualities?

1. fuzzify inputs
2. compute value of left-hand side
3. then apply above rule (e.g. wrt. Mamdani's rule)
4. use defuzzification rule (e.g. centroid)

Example



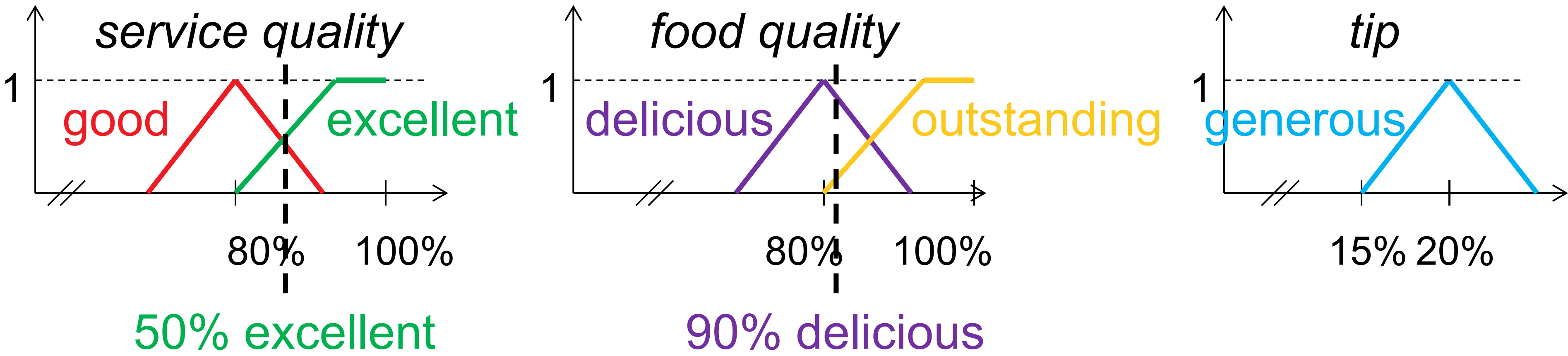
IF service is excellent AND food is delicious THEN tip is generous

What happens for different service and food qualities?

❶ fuzzify:

- 60% excellent AND 20% delicious

Example



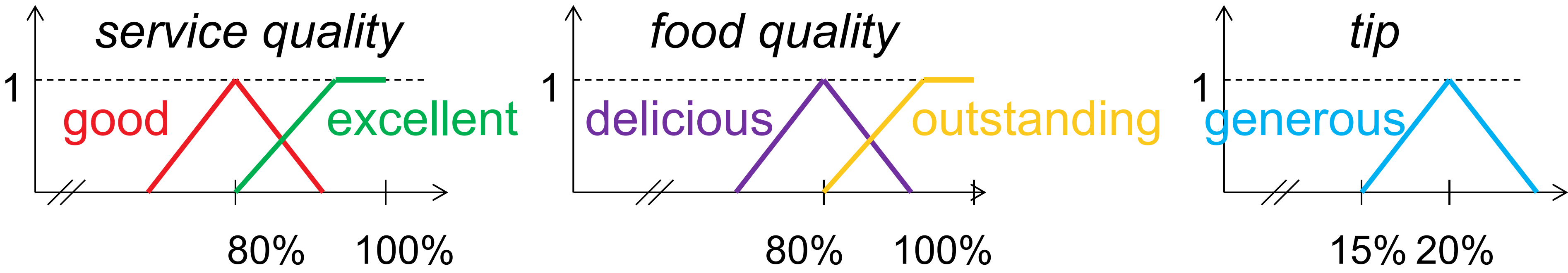
IF service is excellent AND food is delicious THEN tip is generous

What happens for different service and food qualities?

❶ fuzzify:
- 60% excellent AND 20% delicious
- 50% excellent AND 90% delicious

❷ compute value of left-hand: here “AND = min.”
- 20%
- 50%

Example

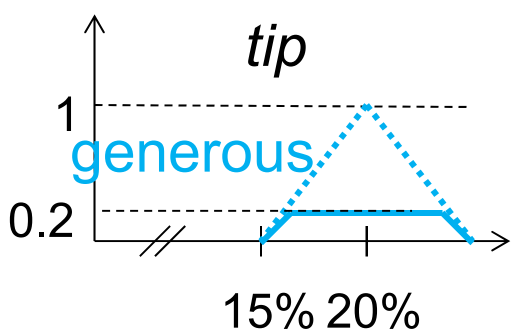


IF service is excellent AND food is delicious THEN tip is generous

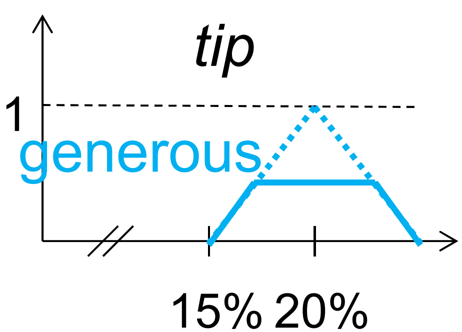
What happens for different service and food qualities?

❸ apply Mamdani's rule: $\mu(x,y) = \min(\mu_A(x), \mu_B(y))$

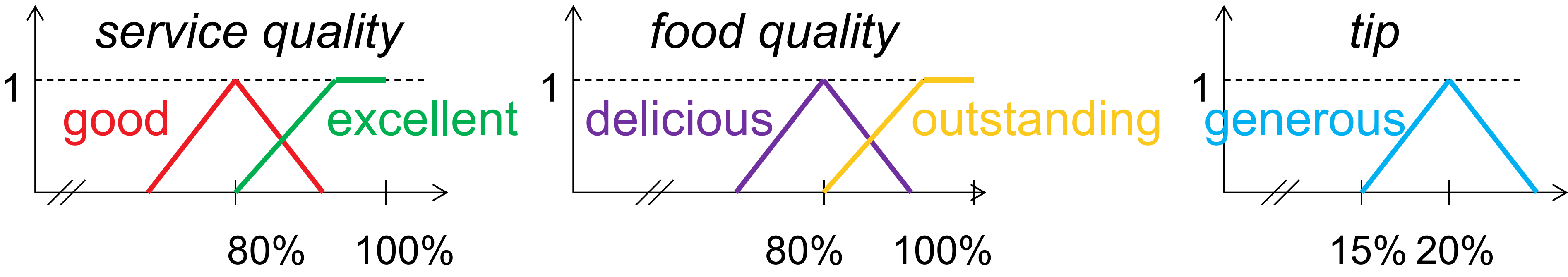
$\mu_A(x) = 20\%$



$\mu_A(x) = 50\%$



Example

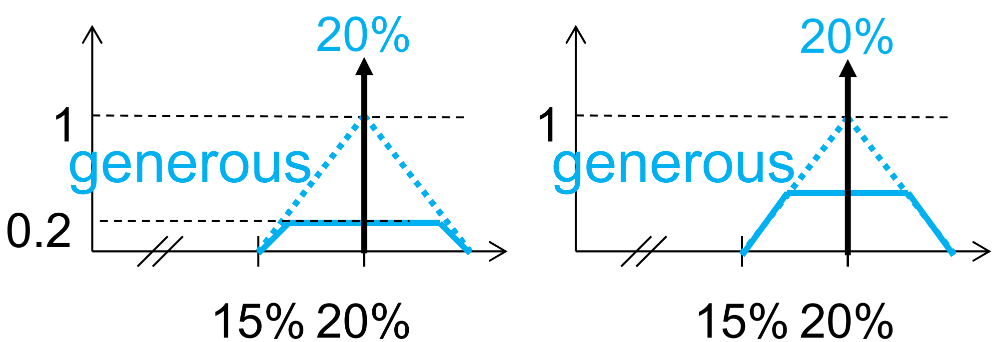


IF service is excellent AND food is delicious THEN tip is generous

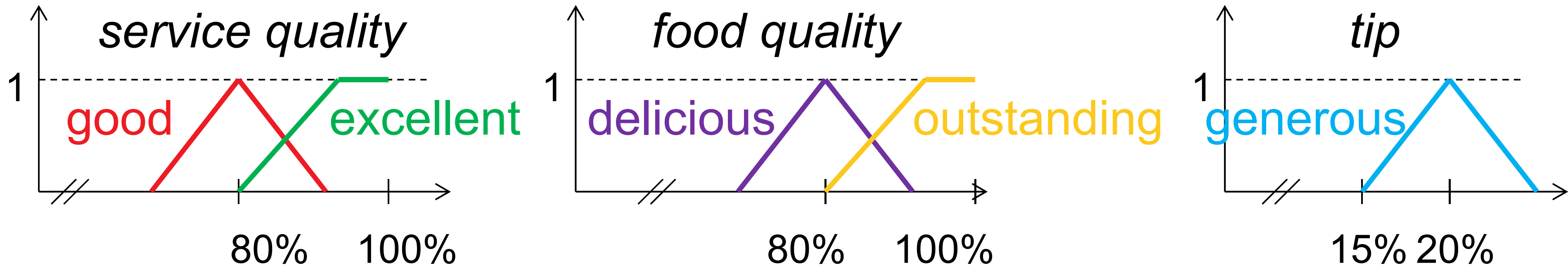
What happens for different service and food qualities?

4. use defuzzification rule (e.g. centroid)

   here: same result, but also only 1 rule applied



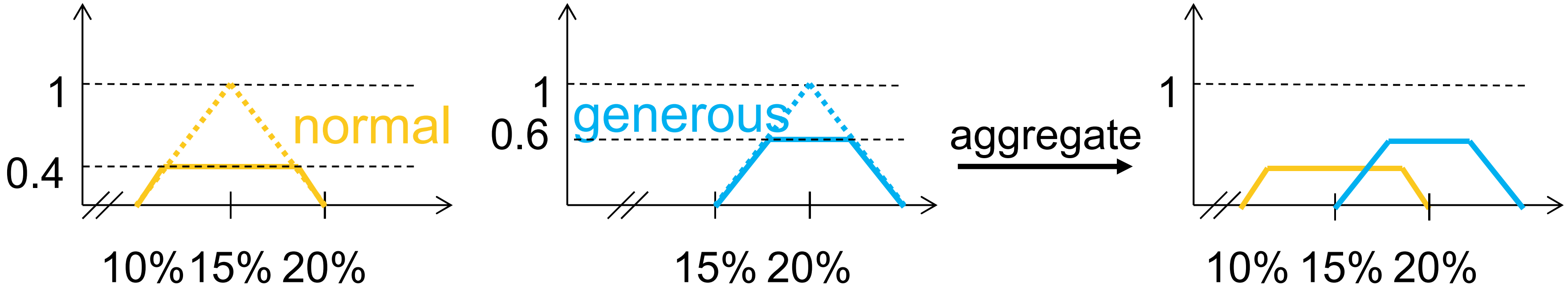
Example



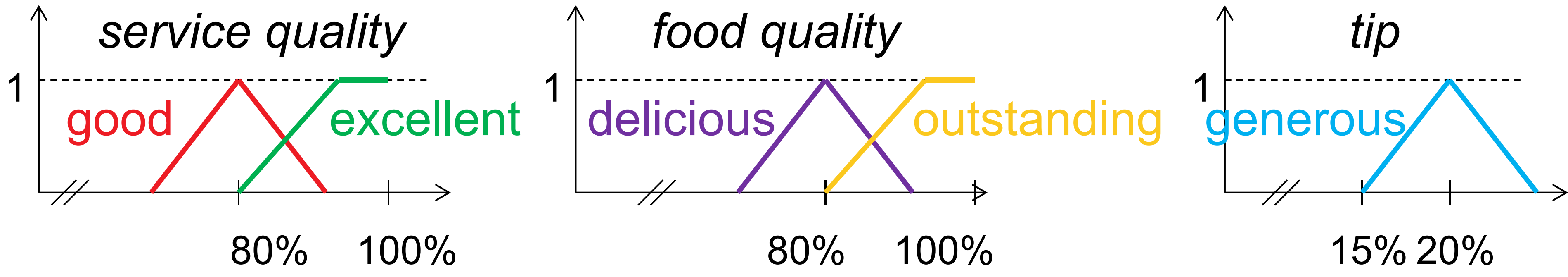
IF service is normal AND food is normal THEN tip is normal
IF service is excellent AND food is delicious THEN tip is generous

Multiple rules

- a) apply all inference rules
- b) aggregate resulting membership functions (e.g. with max.)



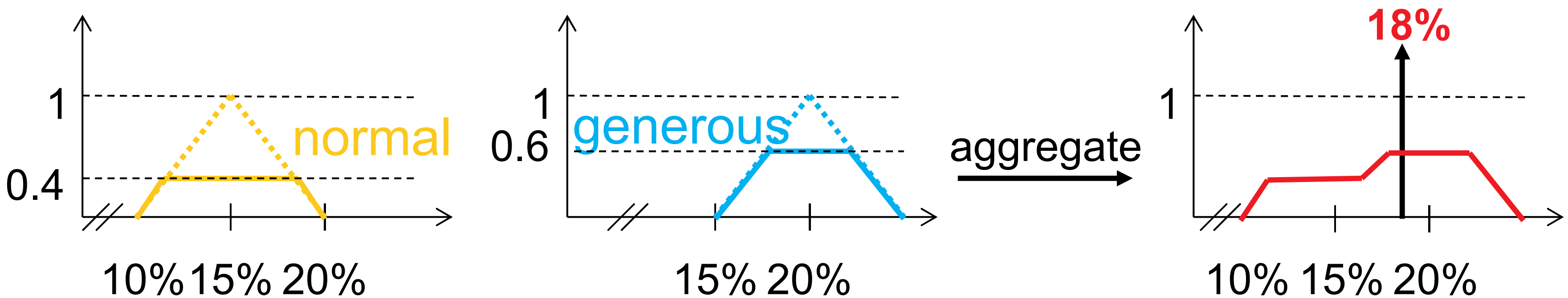
Example



IF service is normal AND food is normal THEN tip is normal
IF service is excellent AND food is delicious THEN tip is generous

Multiple rules

- a) apply all inference rules
- b) aggregate resulting membership functions (e.g. with max.)



Fuzzy Control

“Classical” control:

- mathematical (“crisp”) formulations
- based on mathematical models, especially ODEs
- e.g. "210°C < TEMP < 220°C"

Fuzzy control:

- design formalized by words
- based on experience of the designer
- e.g. "IF (process is too cool) AND (process is getting colder) THEN (add heat to the process)" or "IF (process is too hot) AND (process is heating rapidly) THEN (cool the process quickly)"

A Simple Rule Matrix

Back to the water tap problem from last week:

- imagine measurements of temperature and water flow (e.g. per second) and the controllable inputs “hot water” and “cold water”
- further assume the inputs are fuzzified as {too cold, fine, too hot} (for the temperature) and {not enough, fine, too much} (for the water flow)

Then, a 3x3 rule matrix can show the responses:

	too cold	fine	too hot
not enough			
fine			
too much			

A Simple Rule Matrix

Back to the water tap problem from last week:

- imagine measurements of temperature and water flow (e.g. per second) and the controllable inputs "hot water" and "cold water"
- further assume the inputs are fuzzified as {too cold, fine, too hot} (for the temperature) and {not enough, fine, too much} (for the water flow)

Then, a 3x3 rule matrix can show the responses:

	too cold	fine	too hot
not enough	increase hot	increase hot & cold	increase cold
fine	decrease cold & increase hot	do nothing	increase cold & decrease hot
too much	decrease cold	decrease hot & cold	decrease hot

e.g. IF temperature is fine AND water flow is not enough THEN increase both cold and hot water

Another Rule Matrix

Example: electric heater

- given: goal temperature T_{opt}
- measured: temperature T and temperature change dT/dt
- controlled inputs: heat (heating on) and cool (fan on)
- fuzzify: $T-T_{opt}$ and $d(T-T_{opt})/dt$ in {negative, zero, positive}

		temperature: $T-T_{opt}$		
		negative	zero	positive
temperature change: $d(T-T_{opt})/dt$	negative			
	zero			
	positive			

Another Rule Matrix

Example: electric heater

- given: goal temperature T_{opt}
- measured: temperature T and temperature change dT/dt
- controlled inputs: heat (heating on) and cool (fan on)
- fuzzify: $T-T_{opt}$ and $d(T-T_{opt})/dt$ in {negative, zero, positive}

		temperature: $T-T_{opt}$		
		negative	**zero**	**positive**
temperature change: $d(T-T_{opt})/dt$	**negative**	heat	heat	cool
	zero	heat	do nothing	cool
	positive	heat	cool	cool

Remarks on Rule Matrices

- nothing fancy, but assisting to not forget a rule
- not much helpful if >2 input variables
- not always necessary to define output for all input combinations
- not usable if rules are not of the form “IF a AND b THEN c”
- odd number of rows and columns often helpful (to have a “zero” state with no change)

Again: What if a fuzzified “crisp” input value fire >1 rule?

- then: aggregation (union, max) of output membership functions

How to Design a Fuzzy Controller

1) Define control objectives and criteria

   What am I trying to control? What do I have to do to control the system? What kind of response do I need? What are the possible (probable) system failure modes?

2) Determine input/output relationships and choose the variables.

3) Break the control problem down into a series of IF X AND Y THEN Z rules (or similar) that define the desired system output response for given system input conditions.

   ! If possible, use at least one variable and its time derivative.

4) Create Fuzzy Logic membership functions and decide on inference rules that define the meaning (values) of the Input/Output terms used in your rules.

5) Implement the system in software (or hardware).

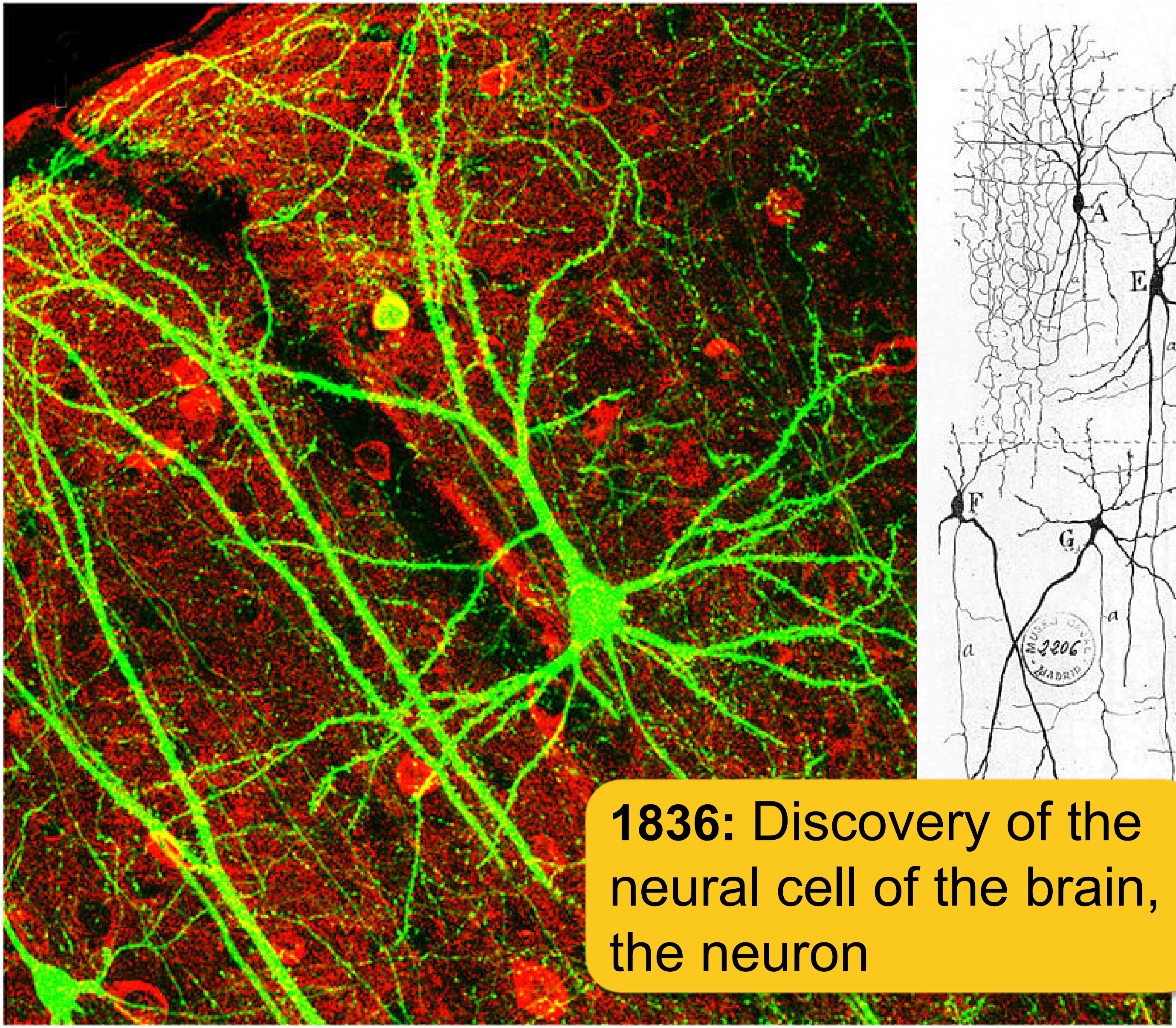
6) Test, evaluate, and tune the rules and membership functions, until satisfactory results are obtained.

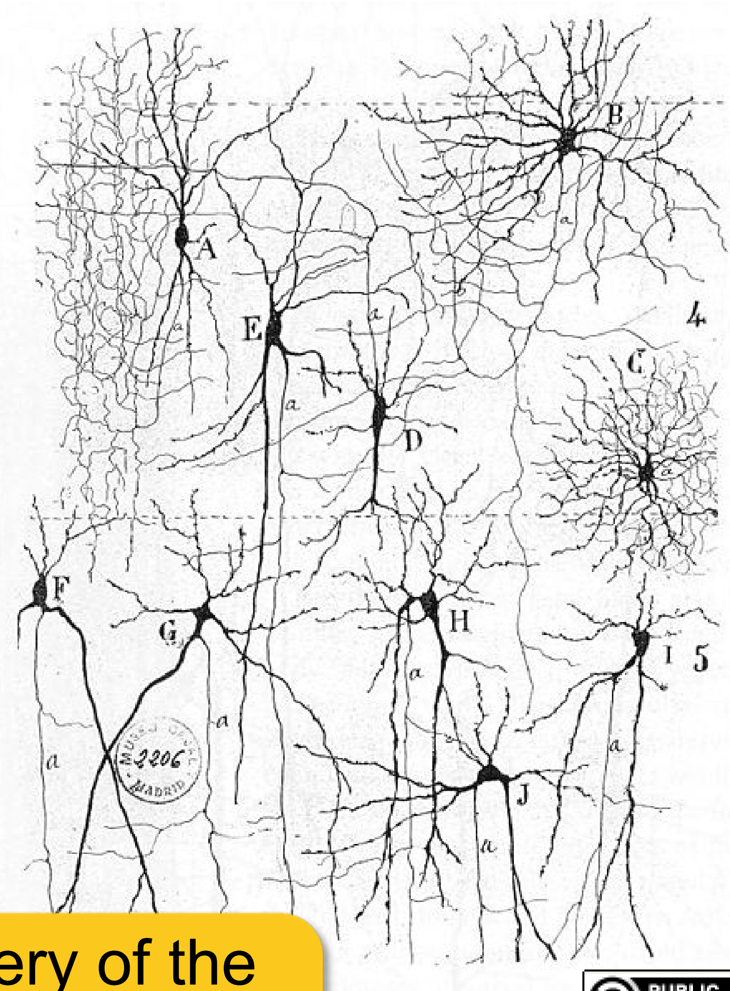
according to the Fuzzy Logic Tutorial by Steven D. Kaehler
http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html

Exercise:
A Fuzzy Controller for the Pole Balancing Problem

Artificial Neural Networks

The Biological Neuron



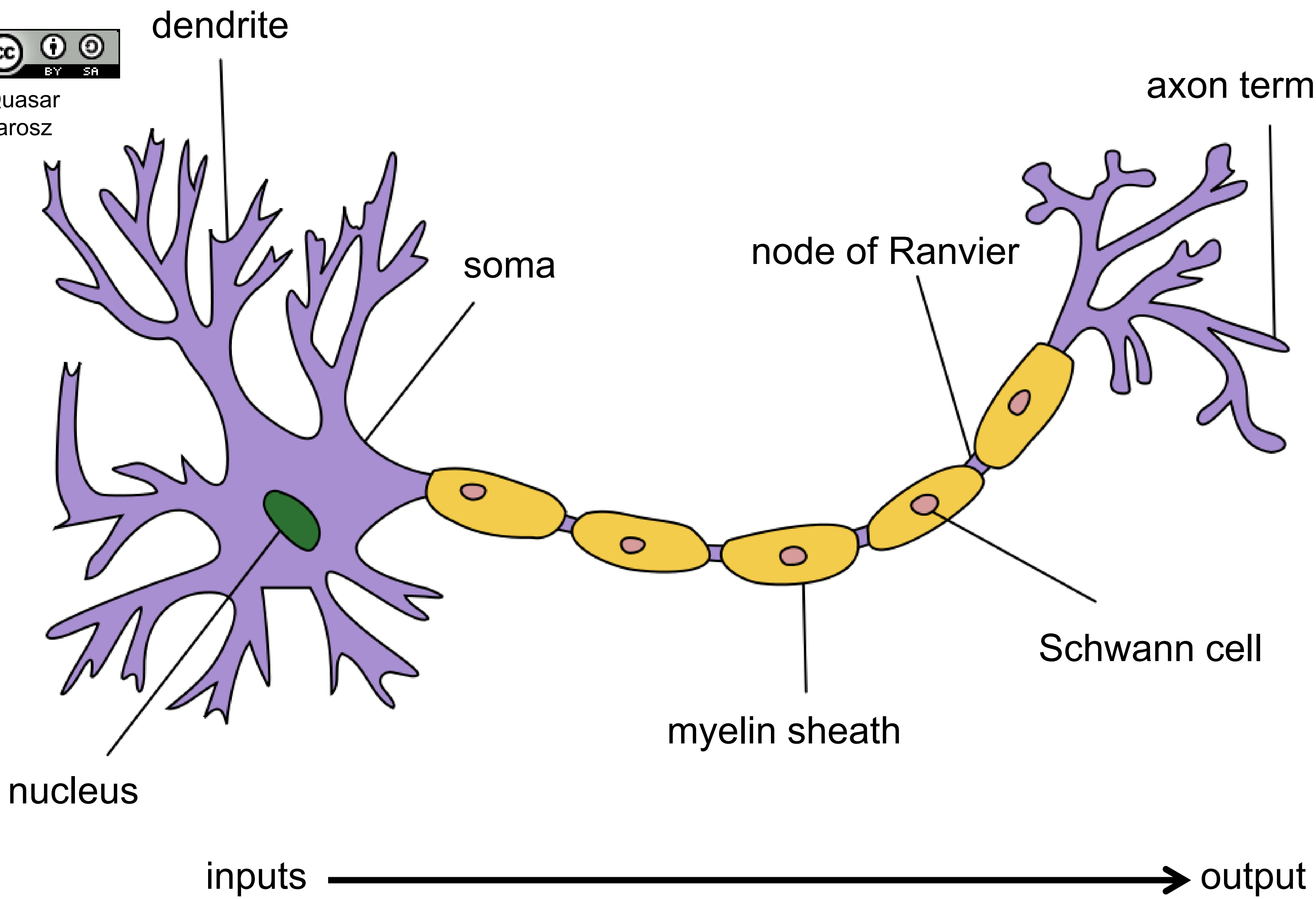


1836: Discovery of the neural cell of the brain, the neuron

 W.-C. A. Lee, H. Huang, G. Feng, J. R. Sanes, E. N. Brown, P. T. So, E. Nedivi

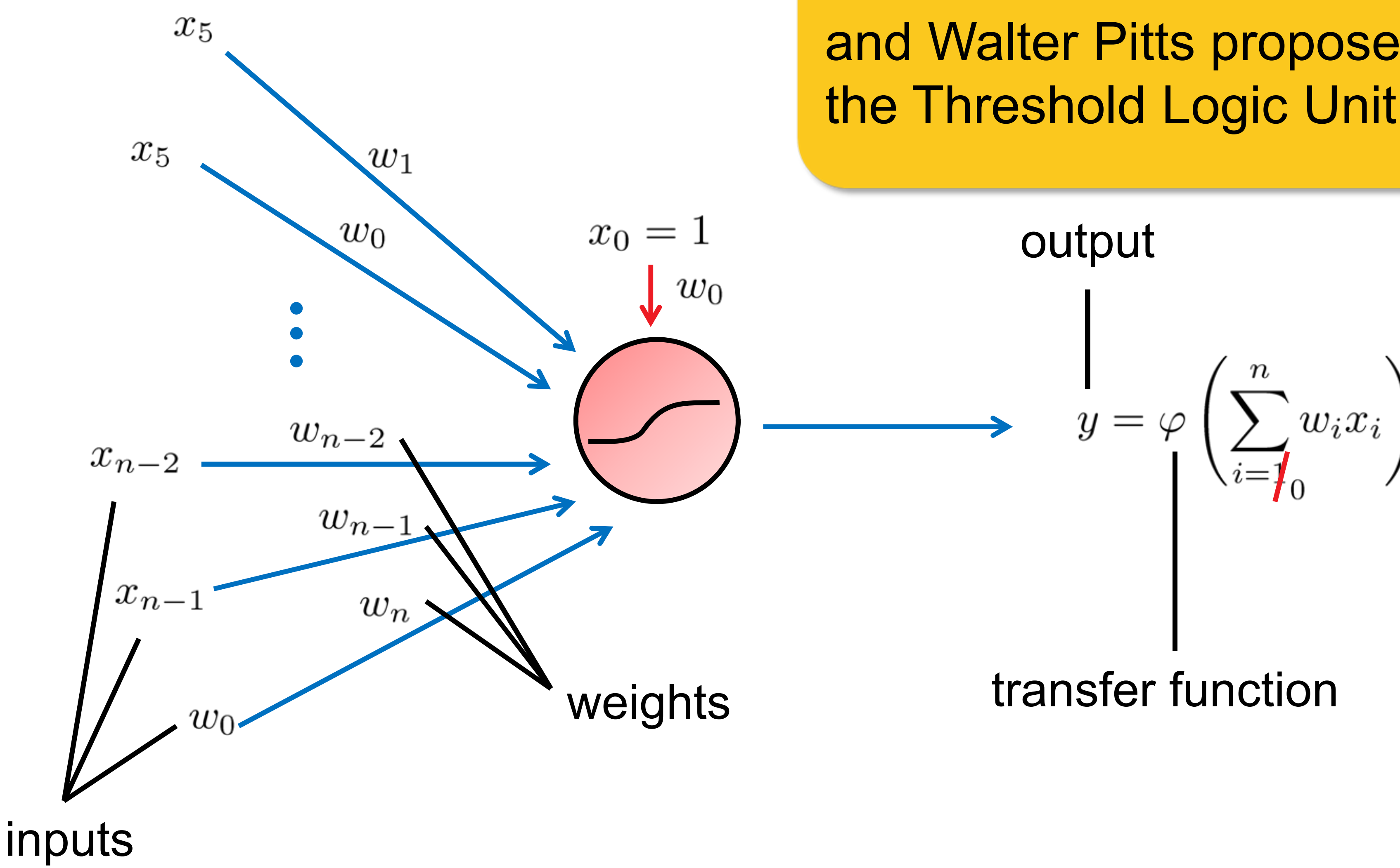
The Biological Neuron

Quasar Jarosz



An Artificial Neuron

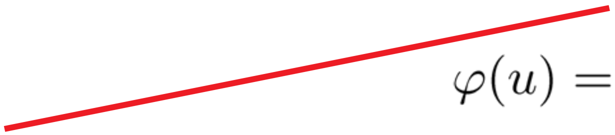
1943: Warren McCulloch and Walter Pitts propose the Threshold Logic Unit



Types of Transfer Functions

$$y = \varphi\Big(\underbrace{\sum_{i=1}^{n} w_i x_i}_{u}\Big)$$

linear

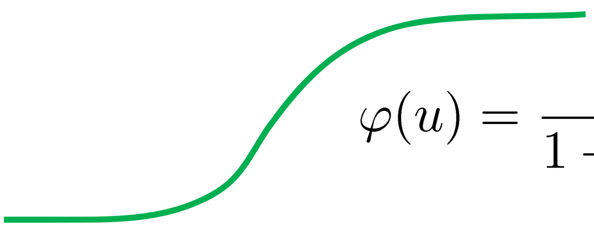


$$\varphi(u) = u + x_0$$

step

$$\varphi(u) = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{if } u < \theta \end{cases}$$

sigmoidal

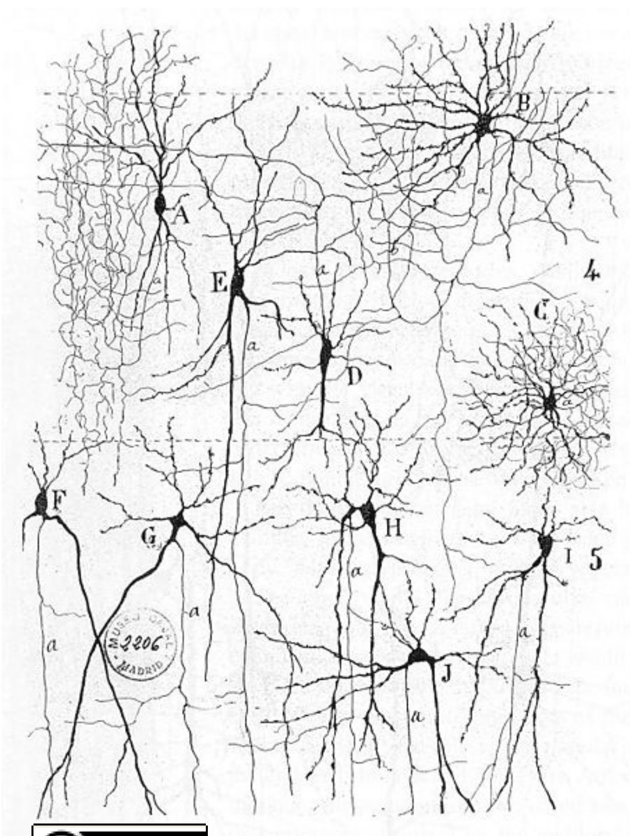


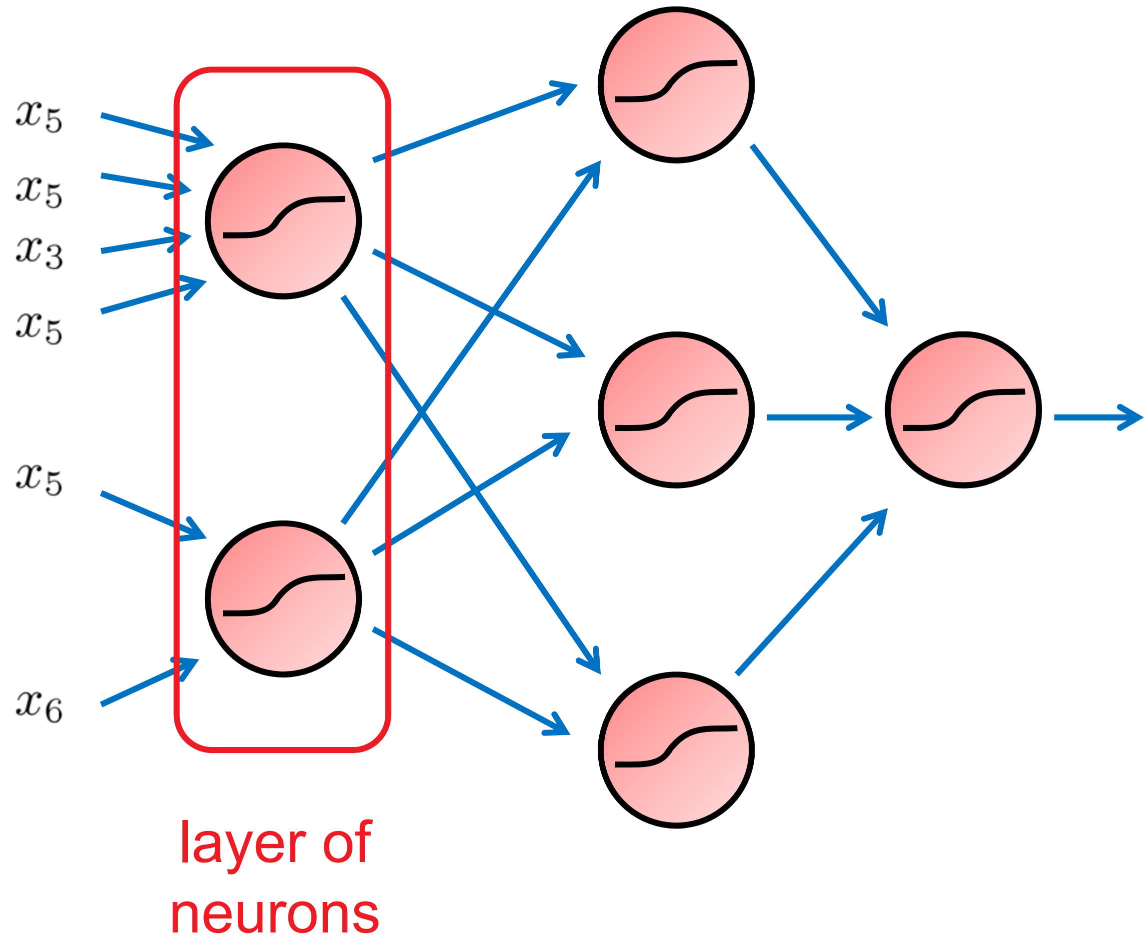
$$\varphi(u) = \frac{1}{1 + e^{-cu}}$$

advantage: differentiable

Combining Artificial Neurons

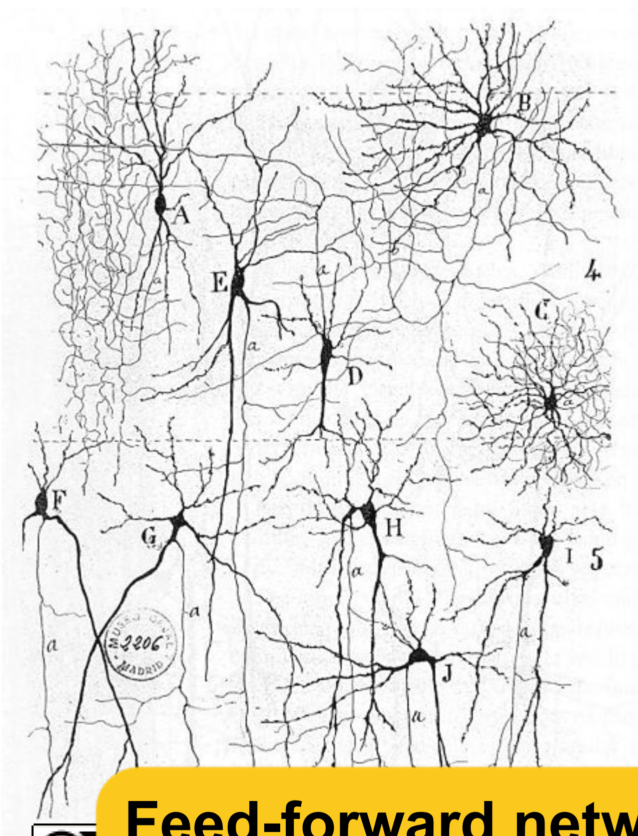
Artificial Neurol Networks = a network of artificial neurons

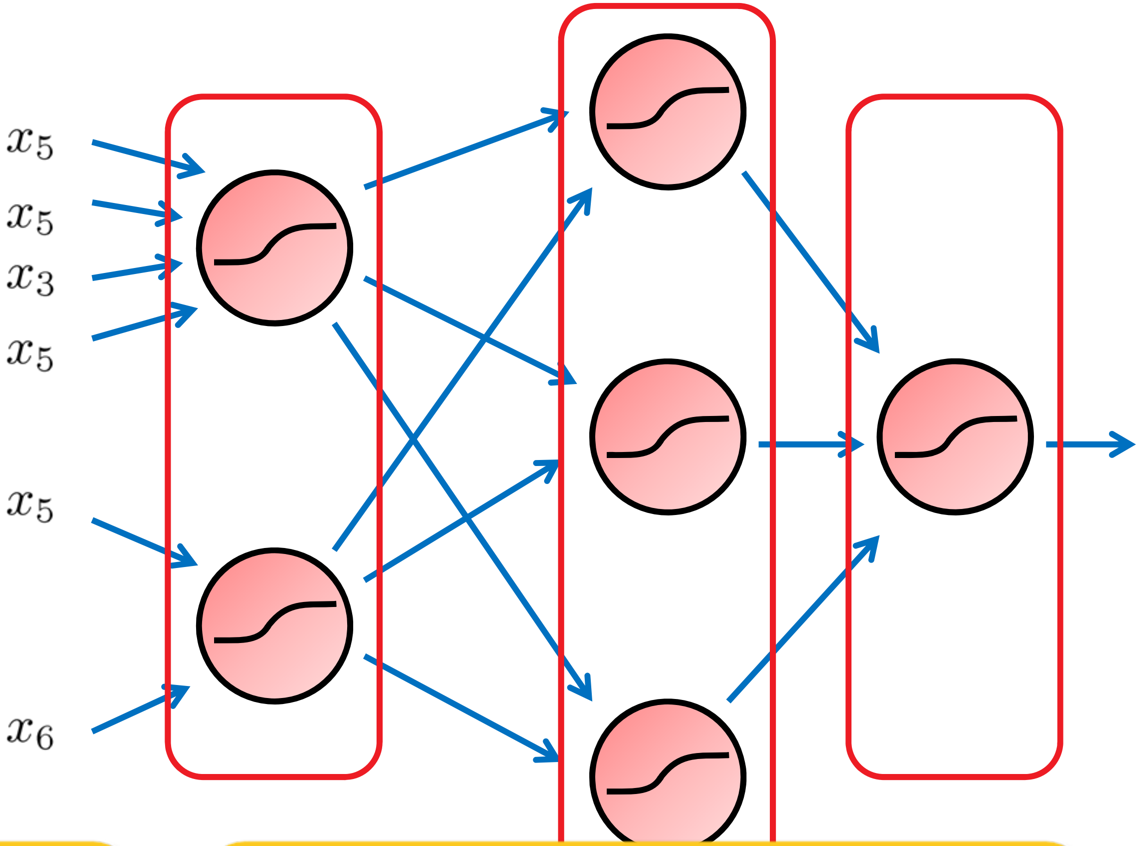




Combining Artificial Neurons

Artificial Neural Networks (ANNs) = a network of artificial neurons





Feed-forward network: no “backwards” flow of information

Linear transfer functions: multi-layer networks can be simulated by a single-layer ANN

Optimizing Weights in Order to Optimize Output

Supervised learning scenario:

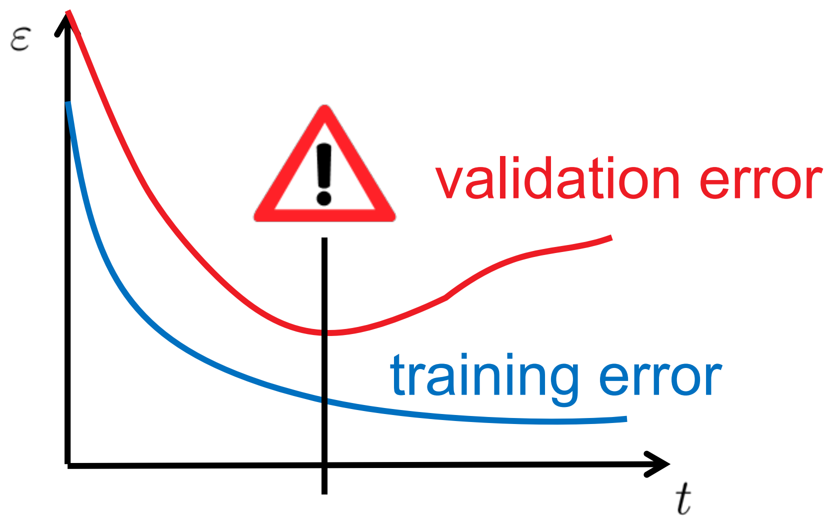
- neural network with n inputs and m outputs
- given a set of training data $(\vec{x}_1, \vec{d}_1), \ldots, (\vec{x}_p, \vec{d}_p)$
- what are "optimal" weights such that

$$E(\vec{w}) = \sum_{j=1}^{p} \sum_{k=1}^{m} ||y_{j,k} - d_{j,k}||^2 = \sum_{j=1}^{p} \sum_{k=1}^{m} ||\varphi_k(\vec{x}_j, \vec{w}) - d_{j,k}||^2$$

is minimal?

training data set vs. testing data set

training error vs. validation error



Generalization vs. Overfitting

- generalization behaviour desired
- overfitting especially when not much training data available

Gradient Descent to Optimize

Optimization: $$\min_{x \in \mathbb{R}^n} f(x)$$

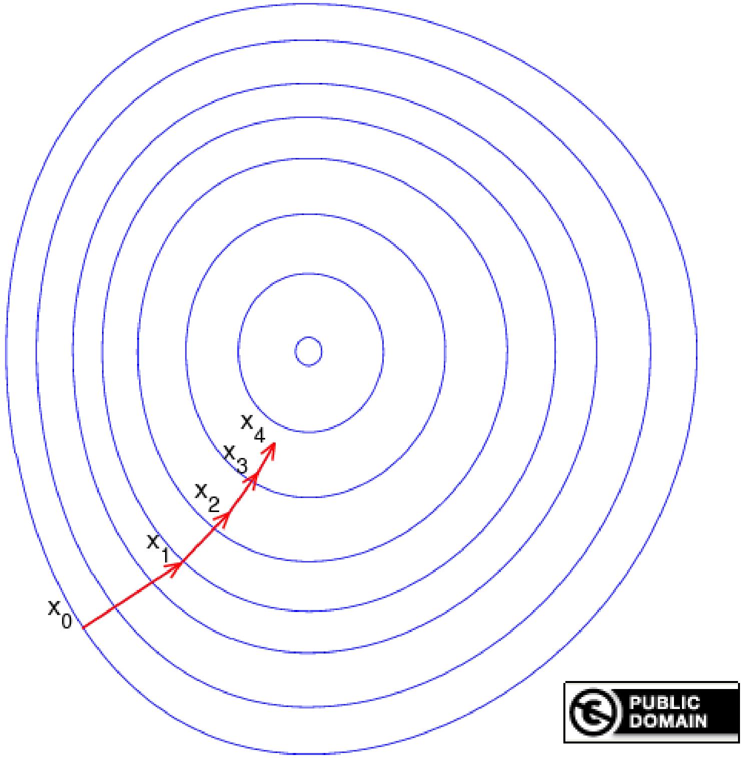
Gradient Descent Algorithm

initialize $x_0 \in \mathbb{R}^n$

At each iteration t :

- compute gradient ∇f
- $x_{t+1} = x_t - \gamma \nabla f(x_t)$

learning rate



Gradient Descent to Optimize

Optimization: $\min_{x \in \mathbb{R}^n} f(x)$

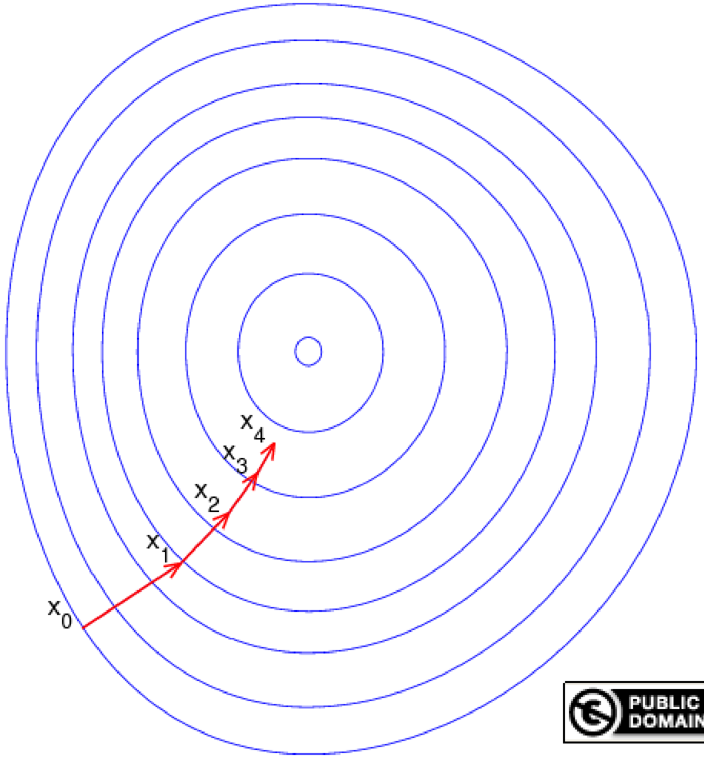
Gradient Descent Algorithm

initialize $x_0 \in \mathbb{R}^n$

At each iteration t :

- compute gradient ∇f
- $x_{t+1} = x_t - \gamma \nabla f(x_t)$

learning rate



! can be slow close to optimum

other algorithms might be favorable

Gradient Descent to Optimize

Optimization: $\min_{x \in \mathbb{R}^n} f(x)$

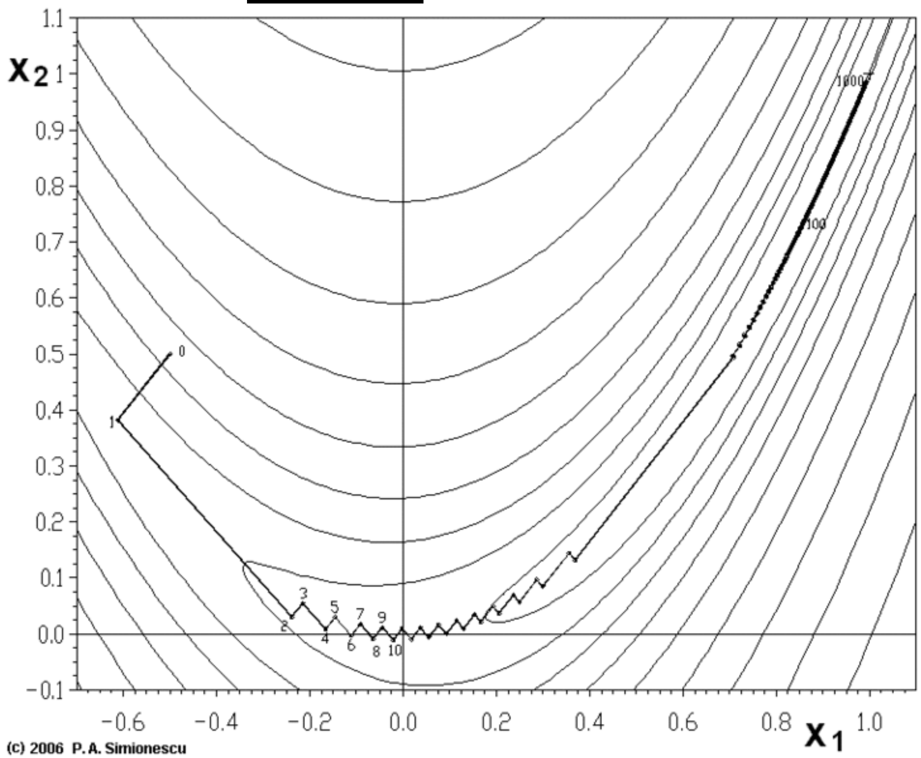
Gradient Descent Algorithm

initialize $x_0 \in \mathbb{R}^n$

At each iteration t :

- compute gradient ∇f
- $x_{t+1} = x_t - \gamma \nabla f(x_t)$

learning rate



! can be slow close to optimum

other algorithms might be favorable

(keyword: natural gradient)

Example:
Rosenbrock function

Optimizing Weights in a Layered Network

How to choose the weights in a multi-layered ANN?

Why not optimize weights directly?

$$E(\vec{w}) = \sum_{j=1}^{p} \sum_{k=1}^{m} ||y_{j,k} - d_{j,k}||^2 = \sum_{j=1}^{p} \sum_{k=1}^{m} ||\varphi(\vec{x}_j, \vec{w}) - d_{j,k}||^2$$

$$\vec{w} = \vec{w} - \nabla \left(\sum_{j=1}^{p} \sum_{k=1}^{m} ||\varphi(\vec{x}_p, \vec{w}) - d_{j,k}||^2 \right)$$

since complicated*, better:

- gradient descent after each training sample
  = stochastic gradient descent (SGD, online gradient descent)

$$w = w - \nabla \left(\sum_{k=1}^{m} ||\varphi(\vec{x}_j, \vec{w}) - d_{j,k}||^2 \right)$$

- descent steps can be performed multiple times over the training set (e.g. with random shuffling)

* complicated: difficult analytically, numerically expensive

Optimizing Weights in a Layered Network

The Backpropagation Algorithm

- introduced around 1970, it gave rise to a renaissance of ANNs
- mainly useful for feed-forward networks
- all transfer functions must be differentiable
- **main idea:**

$$\nabla\left(||\varphi(\vec{x}_j,\vec{w})-d_{j,k}||^2\right) \ : \ \frac{\partial\left(||\varphi(\vec{x}_j,\vec{w})-d_{j,k}||^2\right)}{\partial w_{ij}^{(l)}}$$

  an efficient stochastic gradient descent by updating all weights at once in a smart way
- for simplicity here: only one output layer

Backpropagation: Notations

$$w_{ij}^{(l)} \quad \begin{cases} 1 \le l \le L & \text{layers} \\ 0 \le i \le d^{(l-1)} & \text{inputs} \\ 1 \le j \le d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right)$$

Goal:

$$\min E(\varphi(\vec{x}_n, y_n)) = \min e(\vec{w})$$

using stochastic gradient descent, we need

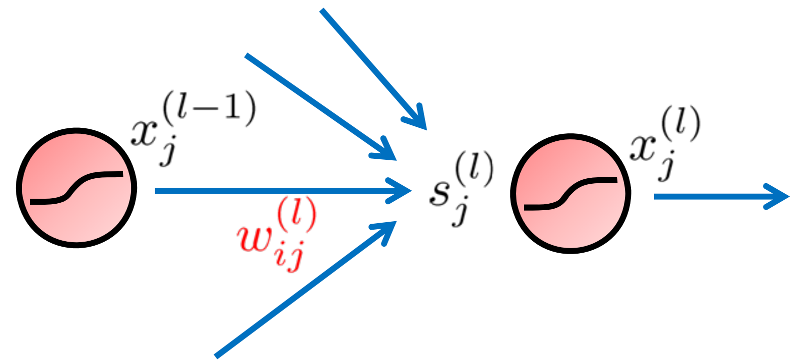
$$\nabla e(\vec{w}) \; : \; \frac{\partial e(\vec{w})}{\partial w_{ij}^{(l)}} \text{ for all } i, j, l$$

Computing the Partial Derivatives

we can compute $\nabla e(\vec{w}) \, : \, \dfrac{\partial e(\vec{w})}{\partial w_{ij}^{(l)}}$ for all i, j, l one by one

- analytically or numerically
- or with a trick at once for all i,j,l

$$\frac{\partial e(\vec{w})}{\partial w_{ij}^{(l)}} = \underbrace{\boxed{\frac{\partial e(\vec{w})}{\partial s_j^{(l)}}}}_{\delta_j^{(l)}} \underbrace{\boxed{\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}}}_{x_i^{(l-1)}}$$



- what needs to be done is to compute the $\delta_j^{(l)}$
- we can start with the output layer and propagate the information backwards by means of the derivative of θ

Computing the $\delta_j^{(l)}$

If time allows:
math for this part in the end of the course

Notes:

- stochastic gradient descent converges to local minimum
- random initial values, restarts
- more about optimization algorithms within the next weeks

Applications of Neural Networks

Many application areas: e.g.

- identification problems
  - face recognition
  - medical diagnoses
  - character recognition in mobile devices
- predictions/forecasting
  - stock market
  - electronic nose
- control

 Ralf Pfeifer

At the end of the course:
exercise using ANNs for
the pole balancing problem

Conclusions

I hope it became clear...

...how to build a fuzzy controller (at least in principle)
...what artificial neural networks are
...and that designing a good controller is not always easy