SOFT COMPUTING

By
K.Sai Saranya,Assistant Professor,Department of CSE

UNIT-1

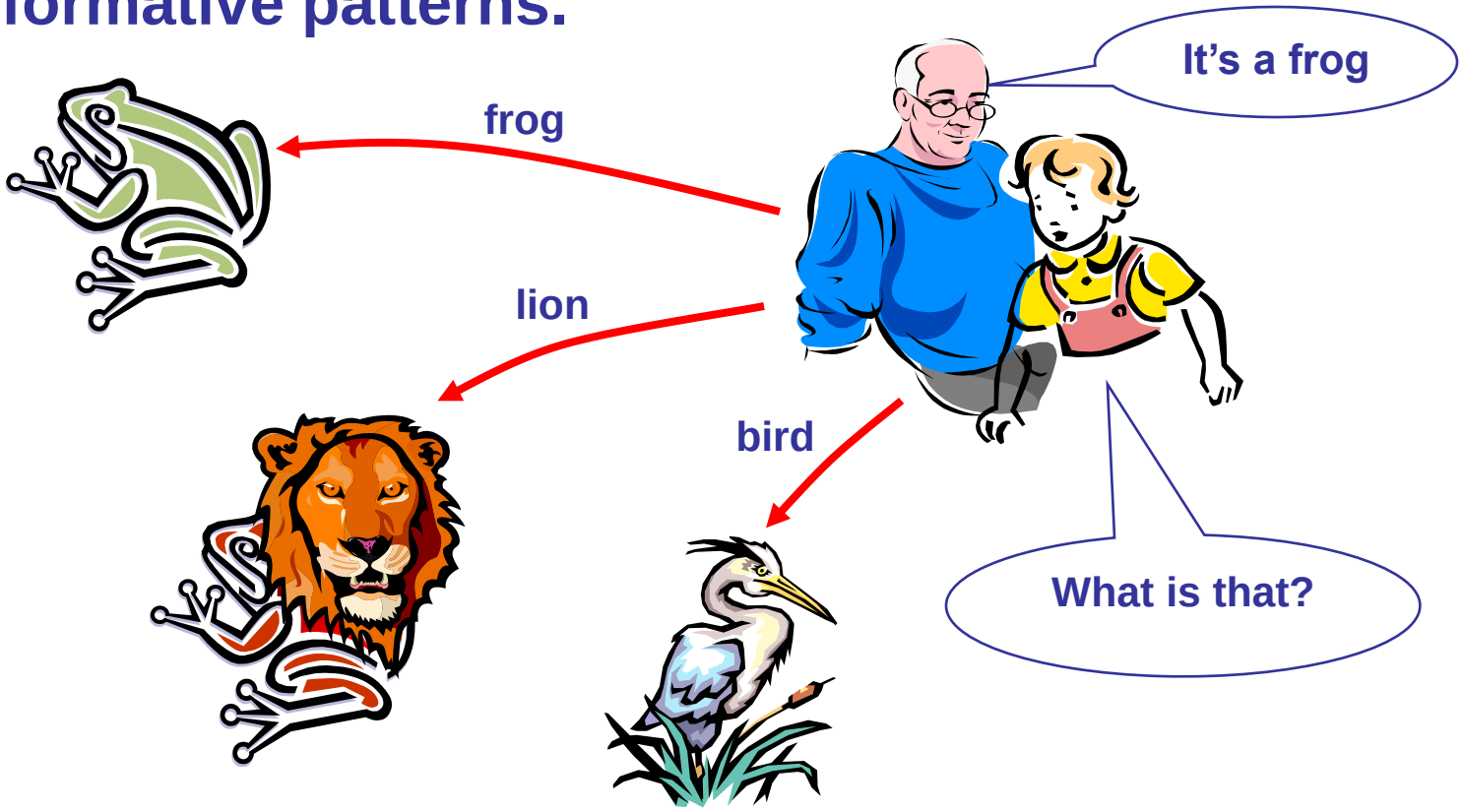
INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS (ANN)

Outline

- Definition, why and how are neural networks being used in solving problems
- Human biological neuron
- Artificial Neuron
- Applications of ANN
- Comparison of ANN vs conventional AI methods

The idea of ANNs..?

- **NNs learn relationship between cause and effect or organize large volumes of data into orderly and informative patterns.**



Neural networks to the rescue…

- **Neural network:** *information processing paradigm inspired by biological nervous systems, such as our brain*
- Structure: large number of highly interconnected processing elements (*neurons*) working together
- Like people, they learn *from experience* (by example)

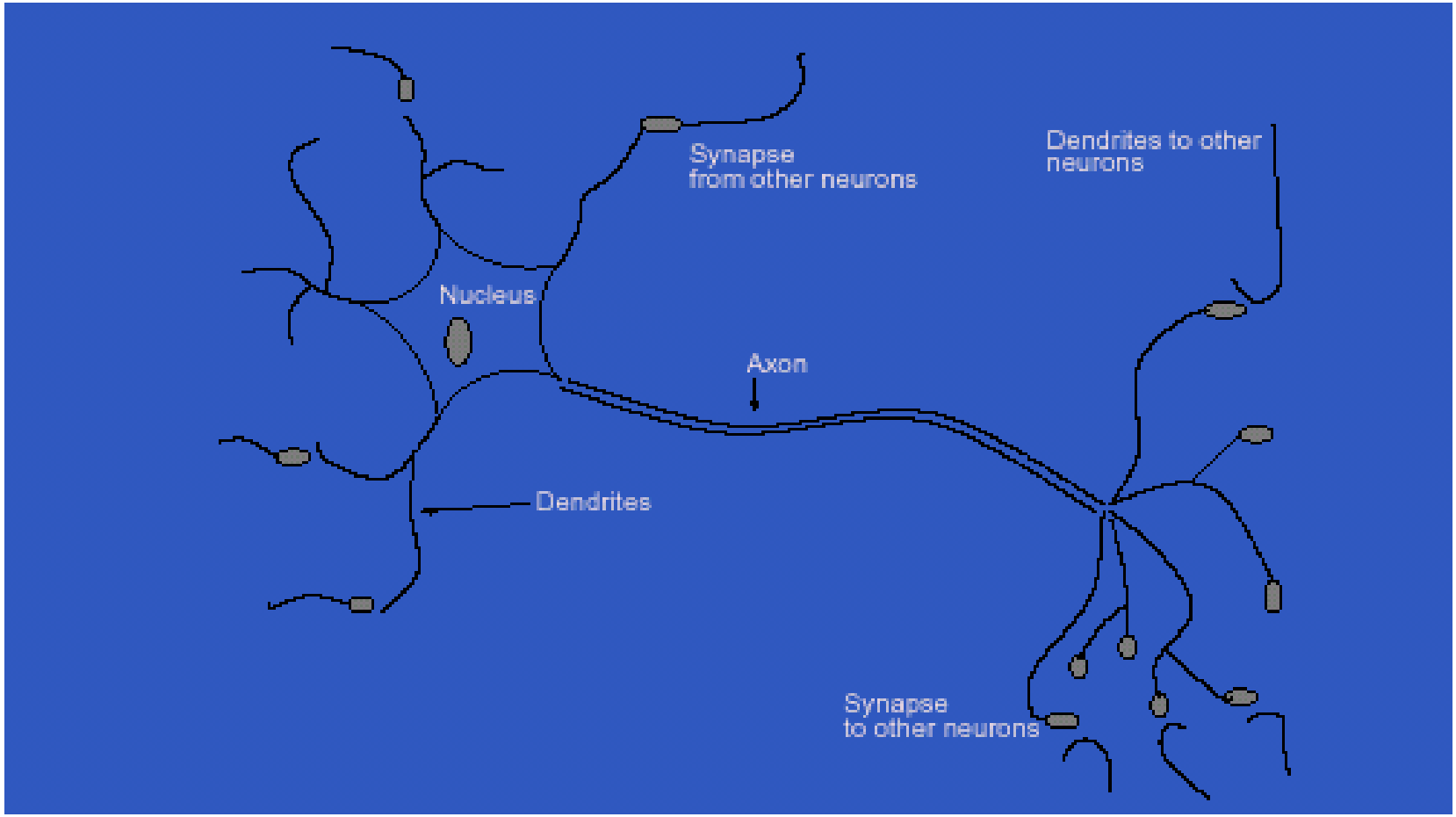
Definition of ANN

"Data processing system consisting of a large number of simple, highly interconnected processing elements (artificial neurons) in an architecture inspired by the structure of the cerebral cortex of the brain"

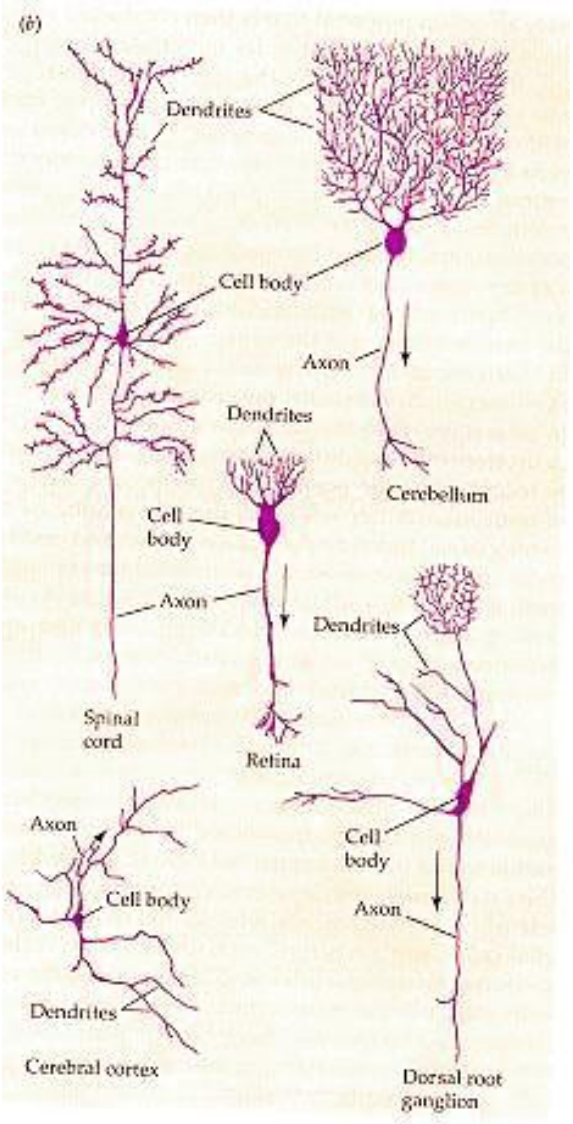
(Tsoukalas & Uhrig, 1997).

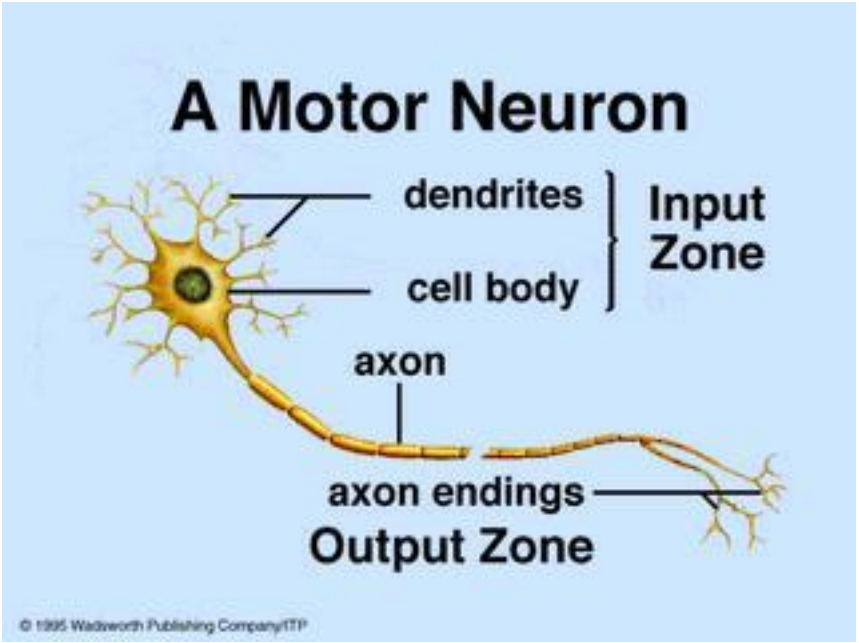
Inspiration from Neurobiology

Human Biological Neuron



Biological Neural Networks

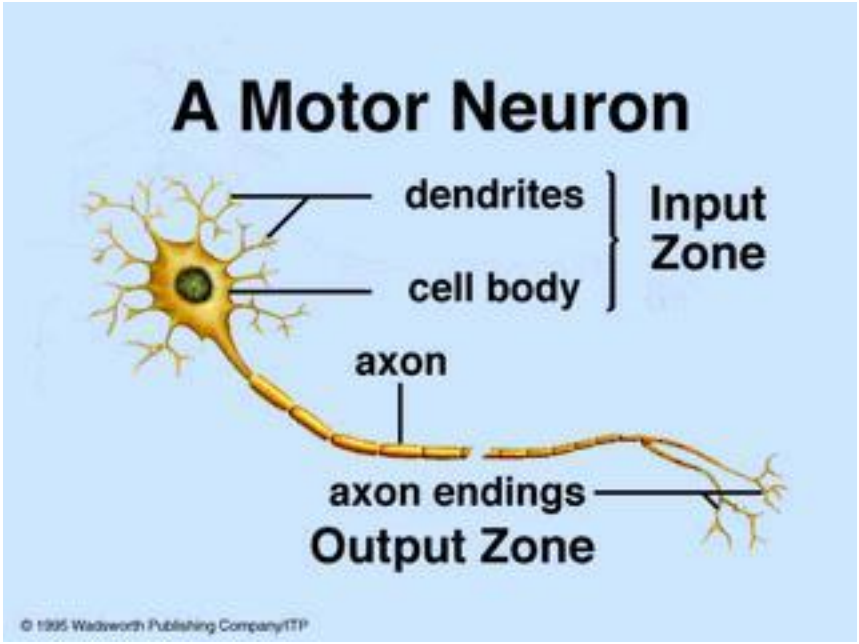




Biological neuron

Biological Neural Networks

- A biological neuron has three types of main components; dendrites, soma (or cell body) and axon.
- Dendrites receives signals from other neurons.



- The soma, sums the incoming signals. When sufficient input is received, the cell fires; that is it transmit a signal over its axon to other cells.

Artificial Neurons

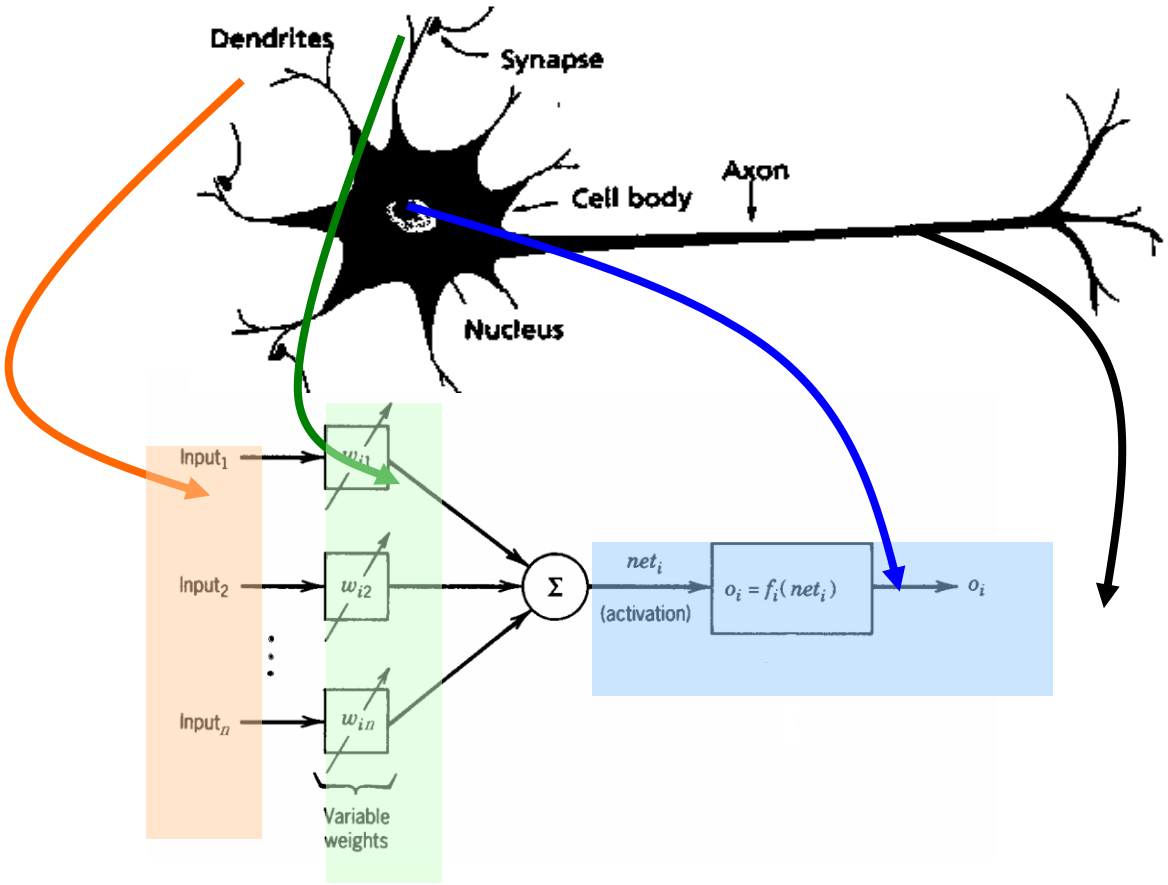
- **ANN is an information processing system that has certain performance characteristics in common with biological nets.**
- **Several key features of the processing elements of ANN are suggested by the properties of biological neurons:**

1. **The processing element receives many signals.**
2. **Signals may be modified by a weight at the receiving synapse.**
3. **The processing element sums the weighted inputs.**
4. **Under appropriate circumstances (sufficient input), the neuron transmits a single output.**
5. **The output from a particular neuron may go to many other neurons.**

Artificial Neurons

- From experience: examples / training data
- Strength of connection between the neurons is stored as a weight-value for the specific connection.
- Learning the solution to a problem = changing the connection weights

A physical neuron



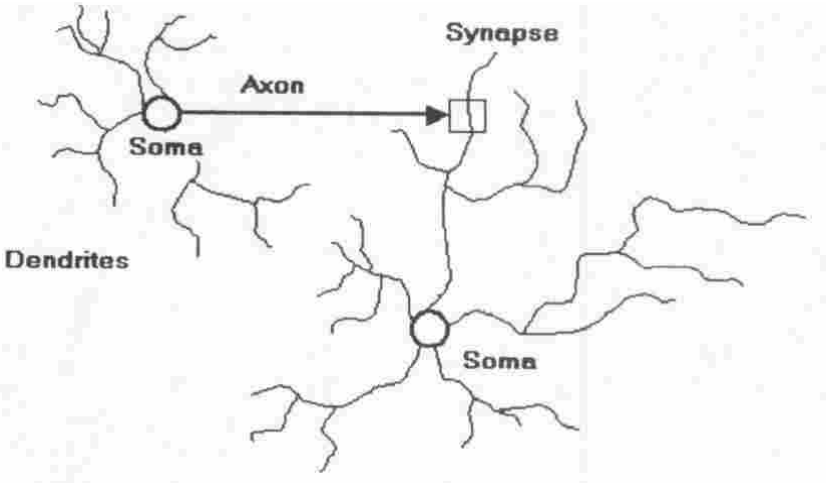
An artificial neuron

Artificial Neurons

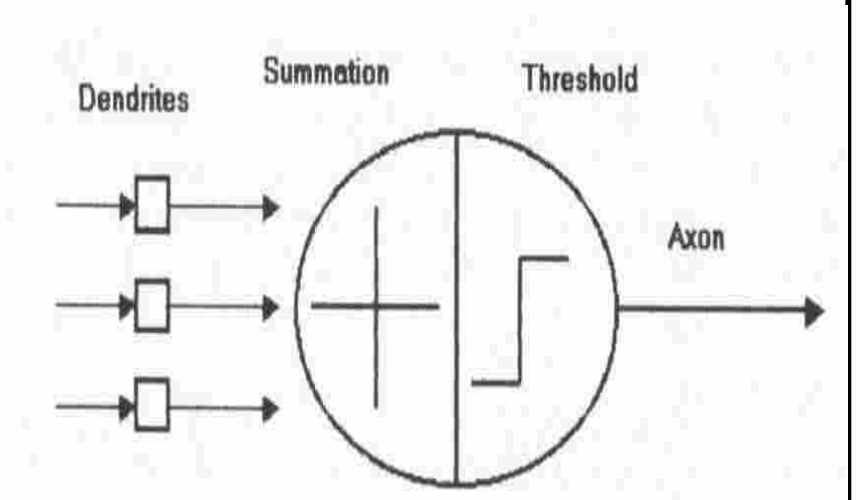
- **ANNs have been developed as generalizations of mathematical models of neural biology, based on the assumptions that:**

  1. **Information processing occurs at many simple elements called neurons.**
  2. **Signals are passed between neurons over connection links.**
  3. **Each connection link has an associated weight, which, in typical neural net, multiplies the signal transmitted.**
  4. **Each neuron applies an activation function to its net input to determine its output signal.**

Artificial Neuron

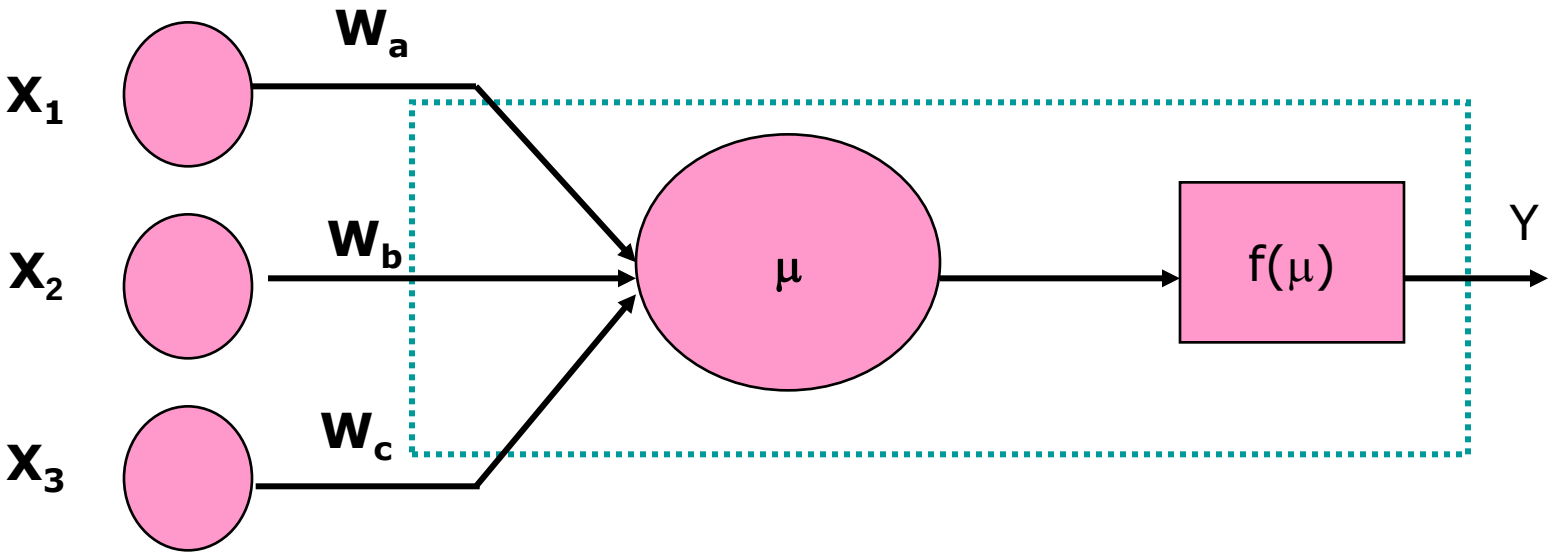


Four basic components of a human biological neuron



The components of a basic artificial neuron

Model Of A Neuron



Input units

Connection weights

Summing function

computation

(dendrite)

(synapse)

(soma)

(axon)

- A neural net consists of a large number of simple processing elements called neurons, units, cells or nodes.
- Each neuron is connected to other neurons by means of directed communication links, each with associated weight.
- The weight represent information being used by the net to solve a problem.

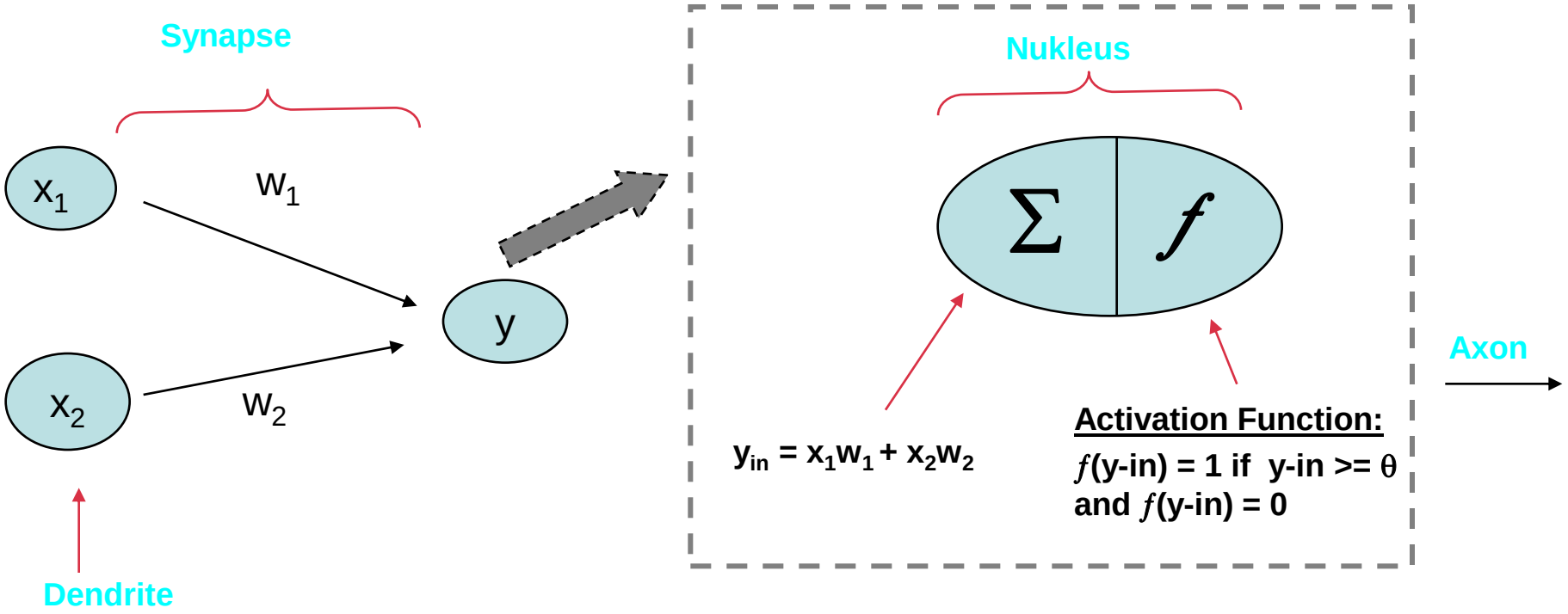
- Each neuron has an internal state, called its activation or activity level, which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons.

- It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

- Neural networks are configured for a specific application, such as pattern recognition or data classification, through a learning process
- In a biological system, learning involves adjustments to the synaptic connections between neurons

➔ same for artificial neural networks (ANNs)

Artificial Neural Network



-A neuron receives input, determines the strength or the weight of the input, calculates the total weighted input, and compares the total weighted with a value (threshold)

-The value is in the range of 0 and 1

- If the total weighted input greater than or equal the threshold value, the neuron will produce the output, and if the total weighted input less than the threshold value, no output will be produced

History

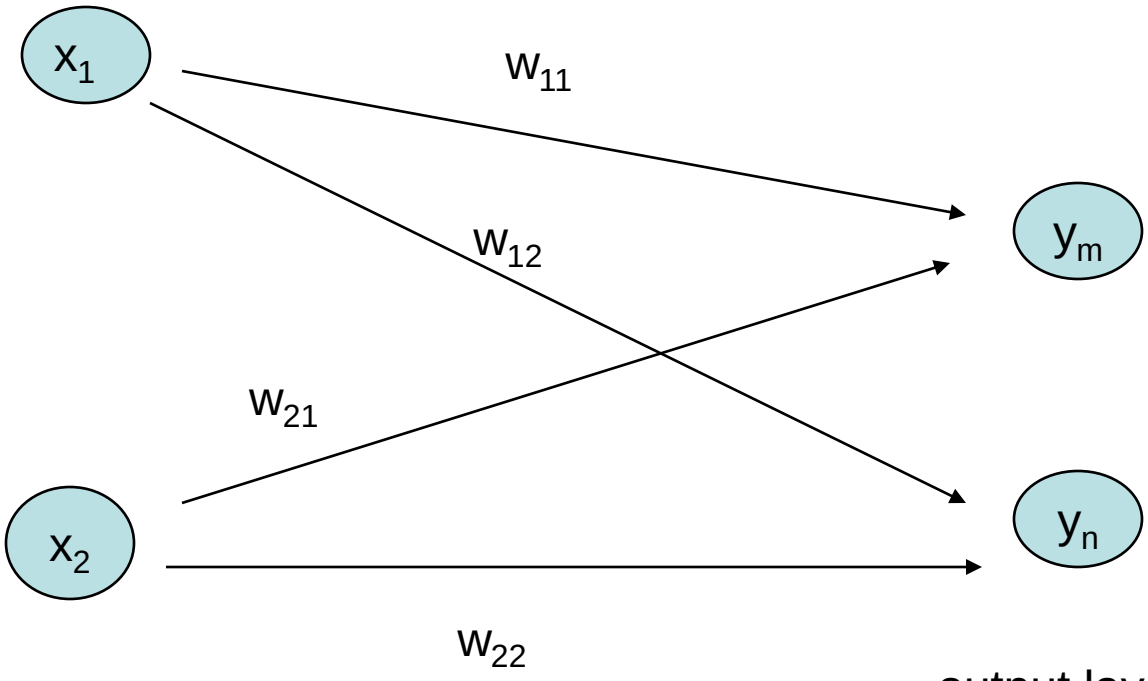
- 1943 McCulloch-Pitts neurons
- 1949 Hebb's law
- 1958 Perceptron (Rosenblatt)
- 1960 Adaline, better learning rule (Widrow, Huff)
- 1969 Limitations (Minsky, Papert)
- 1972 Kohonen nets, associative memory

- 1977 Brain State in a Box (Anderson)
- 1982 Hopfield net, constraint satisfaction
- 1985 ART (Carpenter, Grossfield)
- 1986 Backpropagation (Rumelhart, Hinton, McClelland)
- 1988 Neocognitron, character recognition (Fukushima)

Characterization

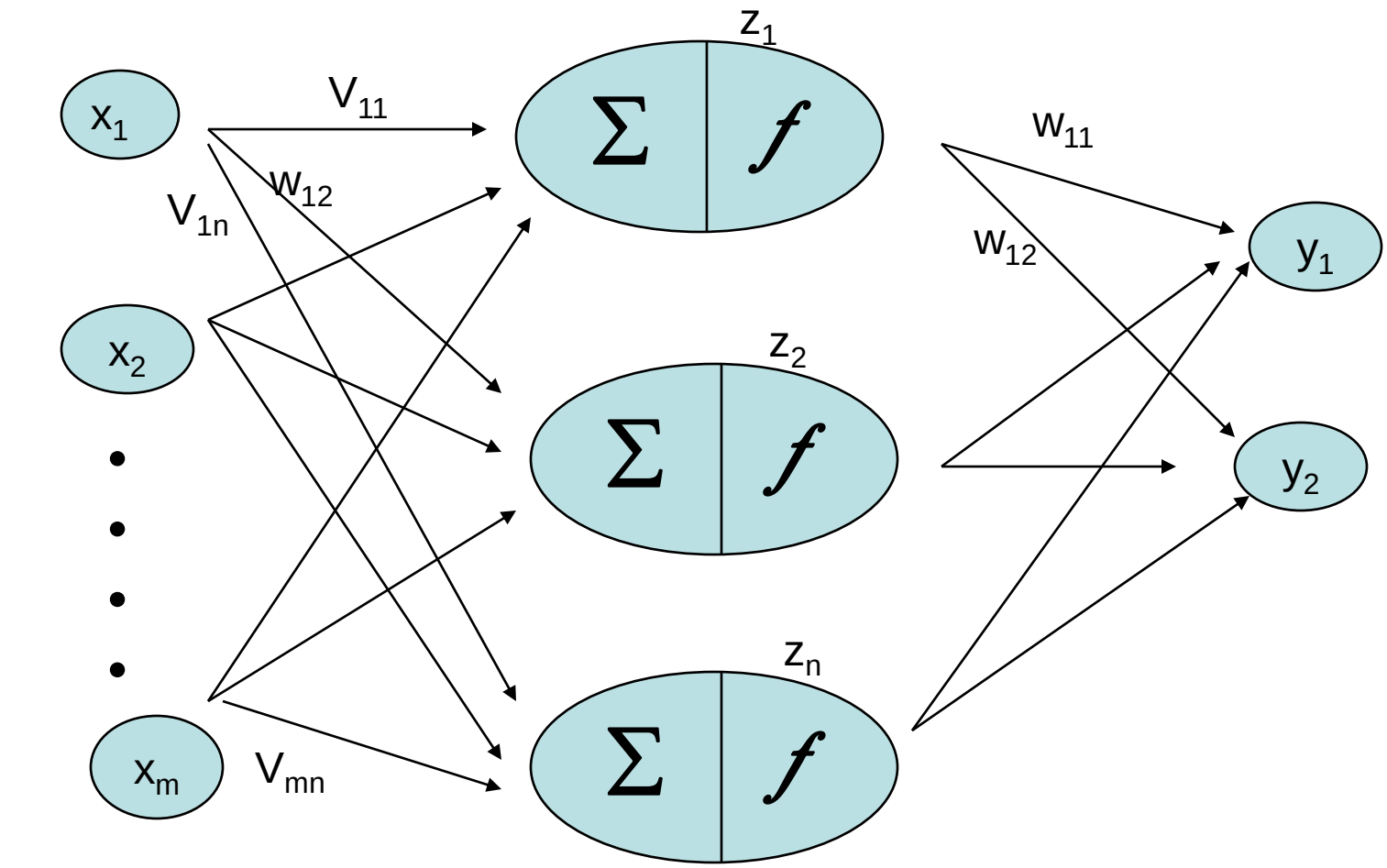
- Architecture
  - a pattern of connections between neurons
    - Single Layer Feedforward
    - Multilayer Feedforward
    - Recurrent
- Strategy / Learning Algorithm
  - a method of determining the connection weights
    - Supervised
    - Unsupervised
    - Reinforcement
- Activation Function
  - Function to compute output signal from input signal

Single Layer Feedforward NN



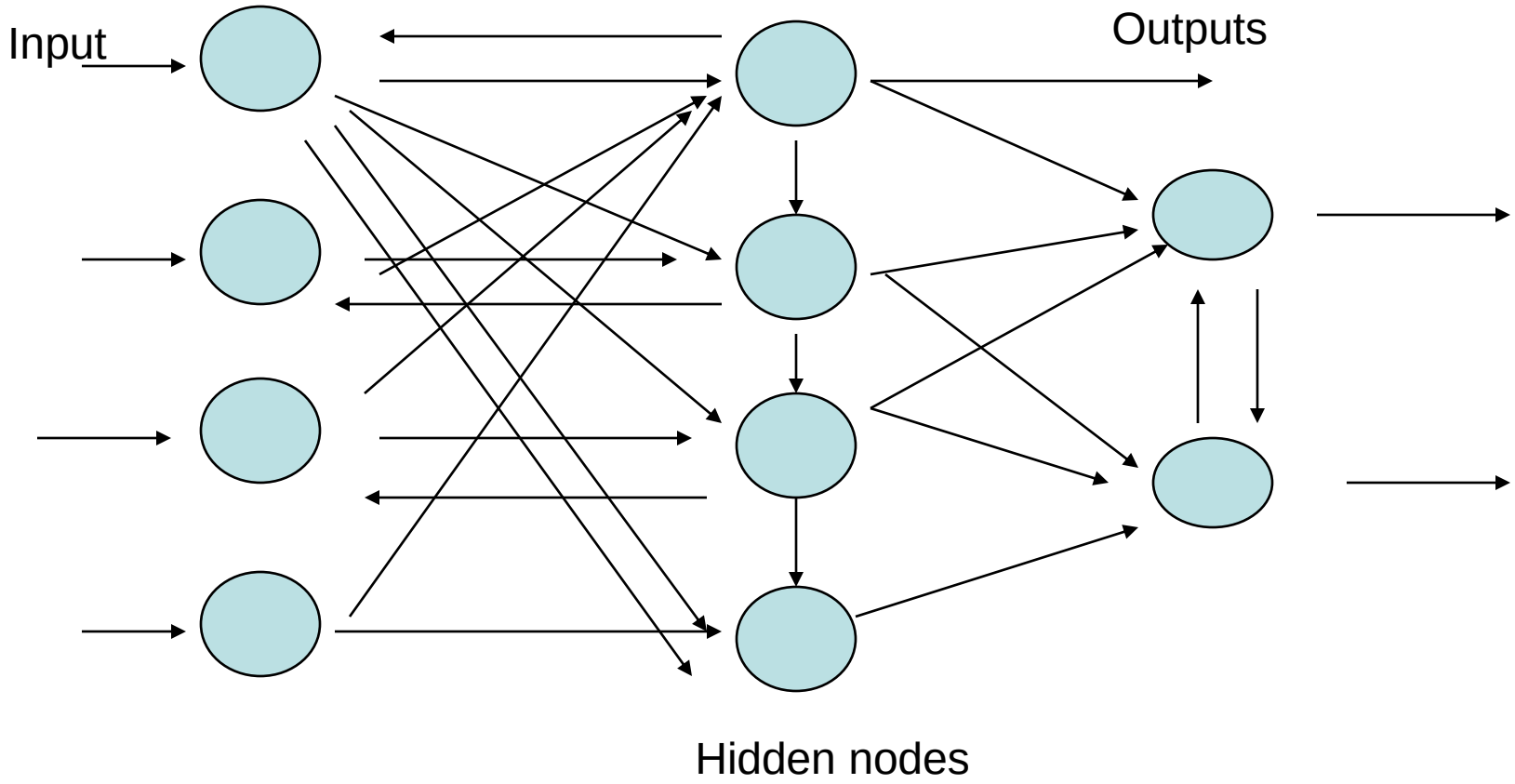
Contoh: **ADALINE, AM, Hopfield, LVQ, Perceptron, SOFM**

Multilayer Neural Network



Contoh: **CCN, GRNN, MADALINE, MLFF with BP, Neocognitron, RBF, RCE**

Recurrent NN



Contoh: **ART, BAM, BSB, Boltzman Machine, Cauchy Machine, Hopfield, RNN**

Strategy / Learning Algorithm

Supervised Learning

- Learning is performed by presenting pattern with target
- During learning, produced output is compared with the desired output
  - The difference between both output is used to modify learning weights according to the learning algorithm
- Recognizing hand-written digits, pattern recognition and etc.
- Neural Network models: perceptron, feed-forward, radial basis function, support vector machine.

Unsupervised Learning

- Targets are not provided
- Appropriate for clustering task
  - Find similar groups of documents in the web, content addressable memory, clustering.
- Neural Network models: Kohonen, self organizing maps, Hopfield networks.

Reinforcement Learning

- Target is provided, but the desired output is absent.
- The net is only provided with guidance to determine the produced output is correct or vise versa.
- Weights are modified in the units that have errors

Activation Functions

- Identity

  $f(x) = x$

- Binary step

  $f(x) = 1$ if $x >= \theta$

  $f(x) = 0$ otherwise

- Binary sigmoid

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

- Bipolar sigmoid

  $f(x) = -1 + 2 / (1 + e^{-\sigma x})$

- Hyperbolic tangent

  $f(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$

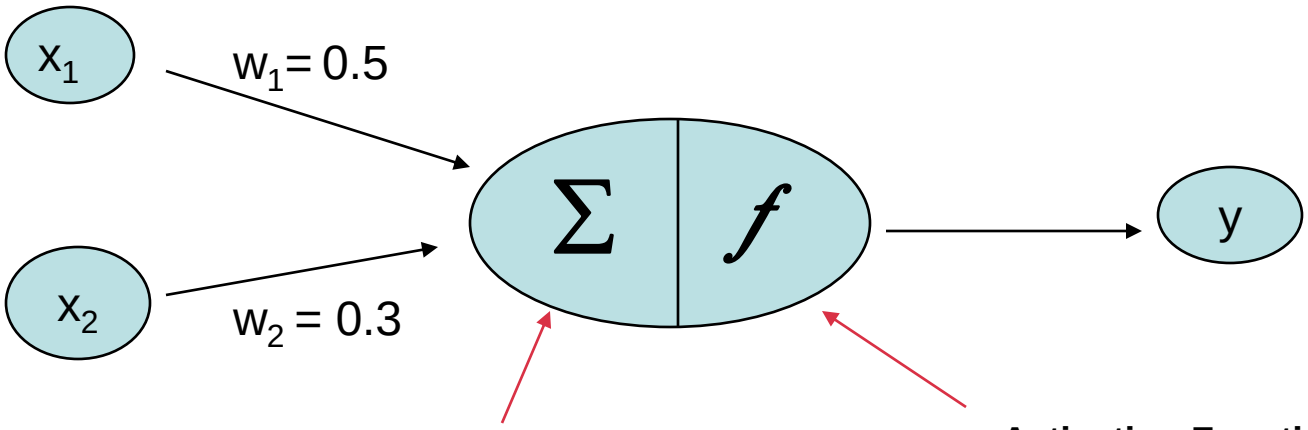
Exercise

- 2 input AND

1	1	1
1	0	0
0	1	0
0	0	0

- 2 input OR

1	1	1
1	0	1
0	1	1
0	0	0



$y_{in} = x_1w_1 + x_2w_2$

Activation Function:
Binary Step Function
$\theta = 0.5$,

f(y-in) = 1 if y-in >= θ
dan f(y-in) = 0

Where can neural network systems help…

- when we can't formulate an algorithmic solution.
- when we ***can*** get lots of examples of the behavior we require.

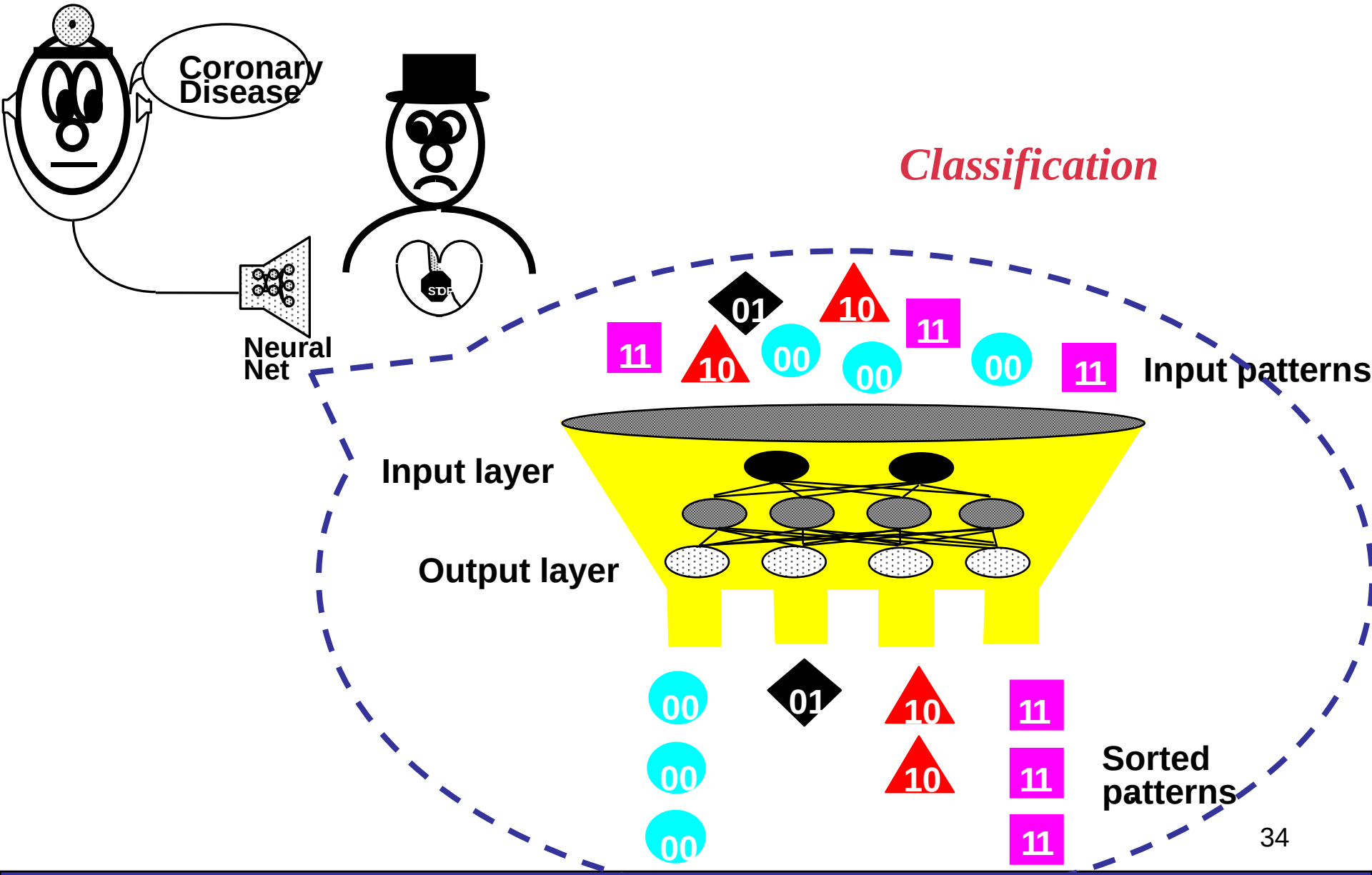
  'learning from experience'
- when we need to pick out the structure from existing data.

Who is interested?...

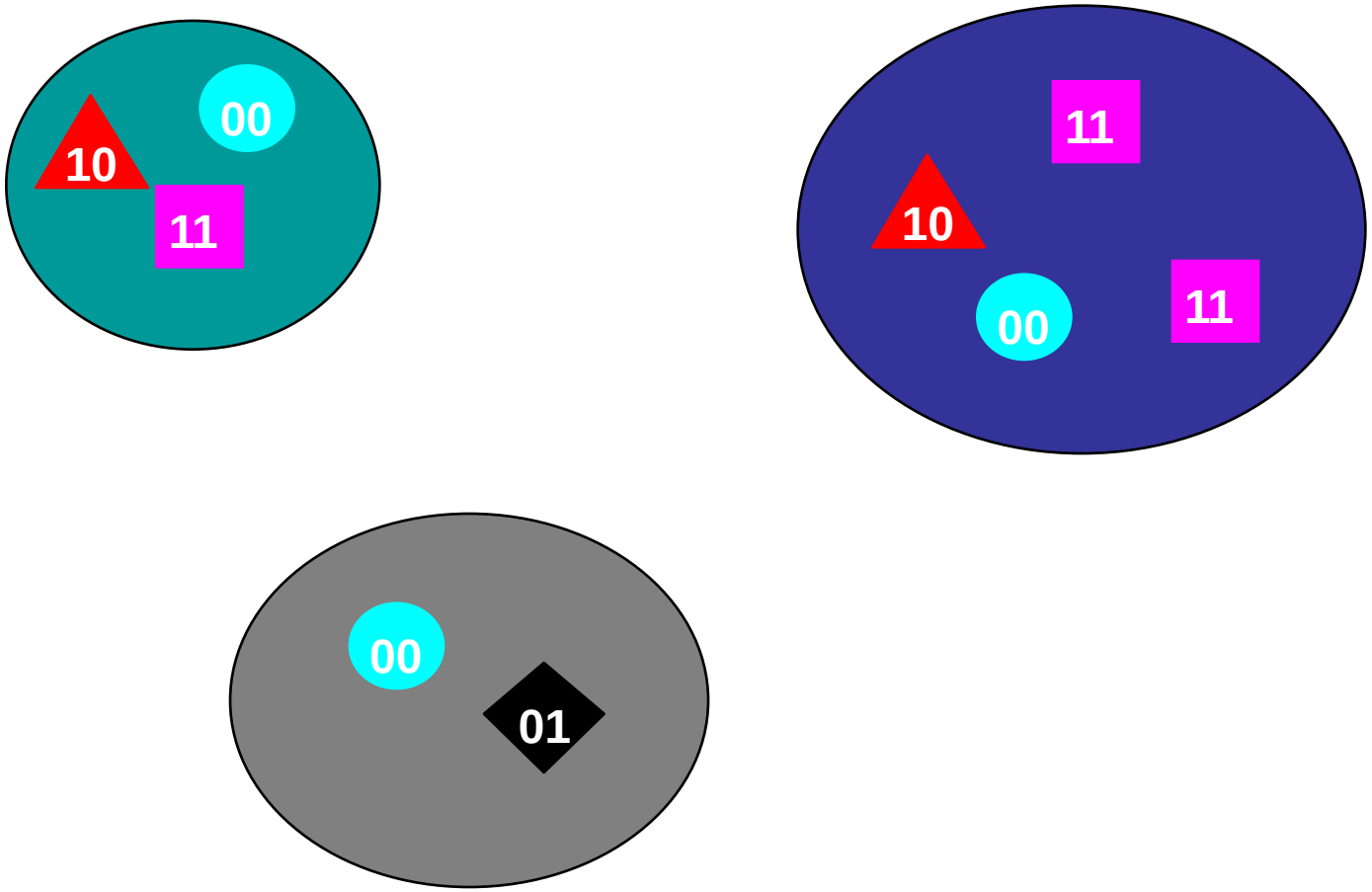
- Electrical Engineers – signal processing, control theory
- Computer Engineers – robotics
- Computer Scientists – artificial intelligence, pattern recognition
- Mathematicians – modelling tool when explicit relationships are unknown

Problem Domains

- Storing and recalling patterns
- Classifying patterns
- Mapping inputs onto outputs
- Grouping similar patterns
- Finding solutions to constrained optimization problems

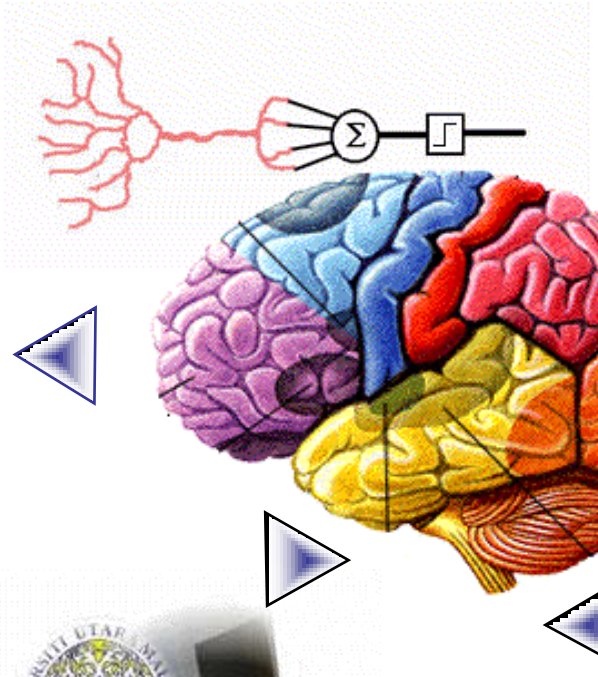

Coronary Disease
STOP
Neural Net
Classification
01
10
11
00
Input patterns
Input layer
Output layer
Sorted patterns

Clustering



ANN Applications

Medical Applications



Chemistry

Information
Searching & retrieval

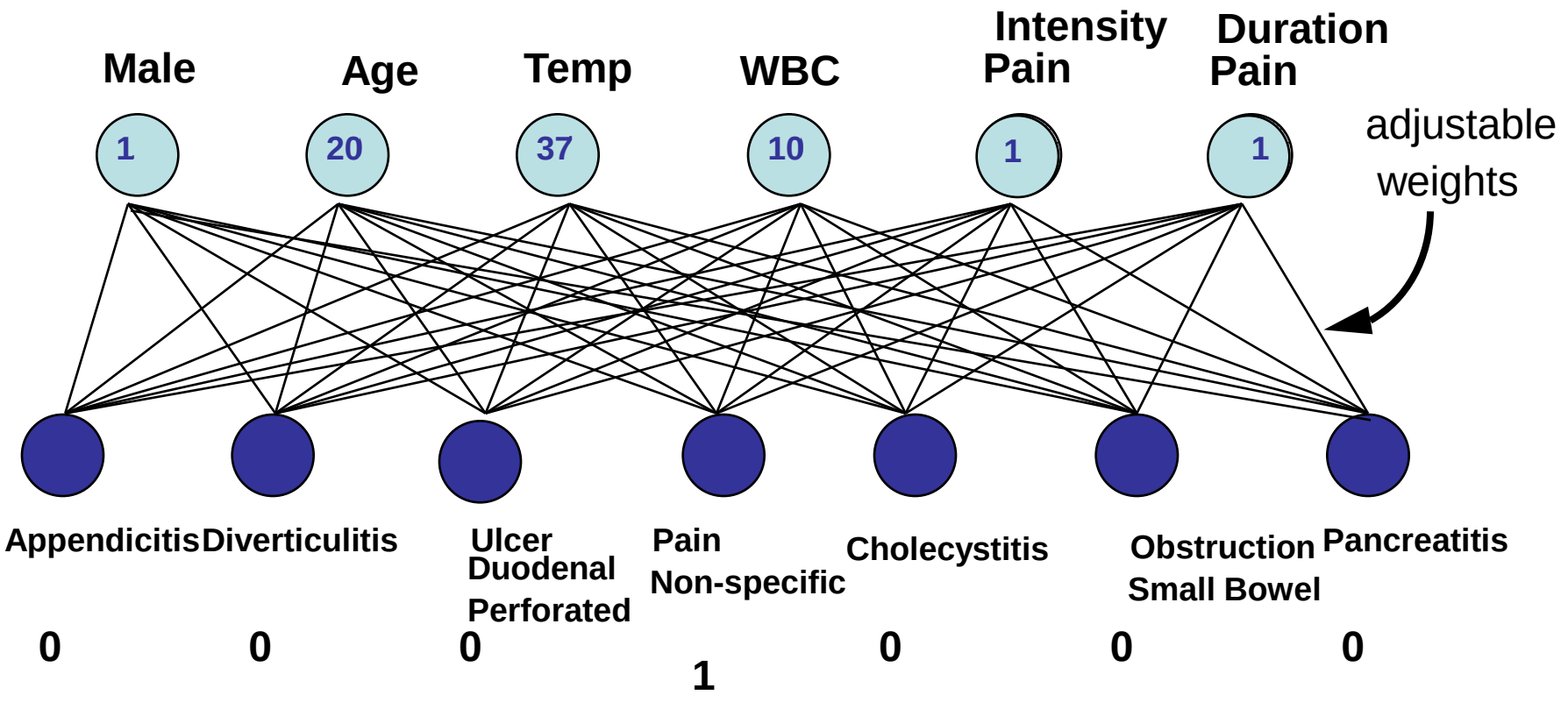
Education

Business & Management

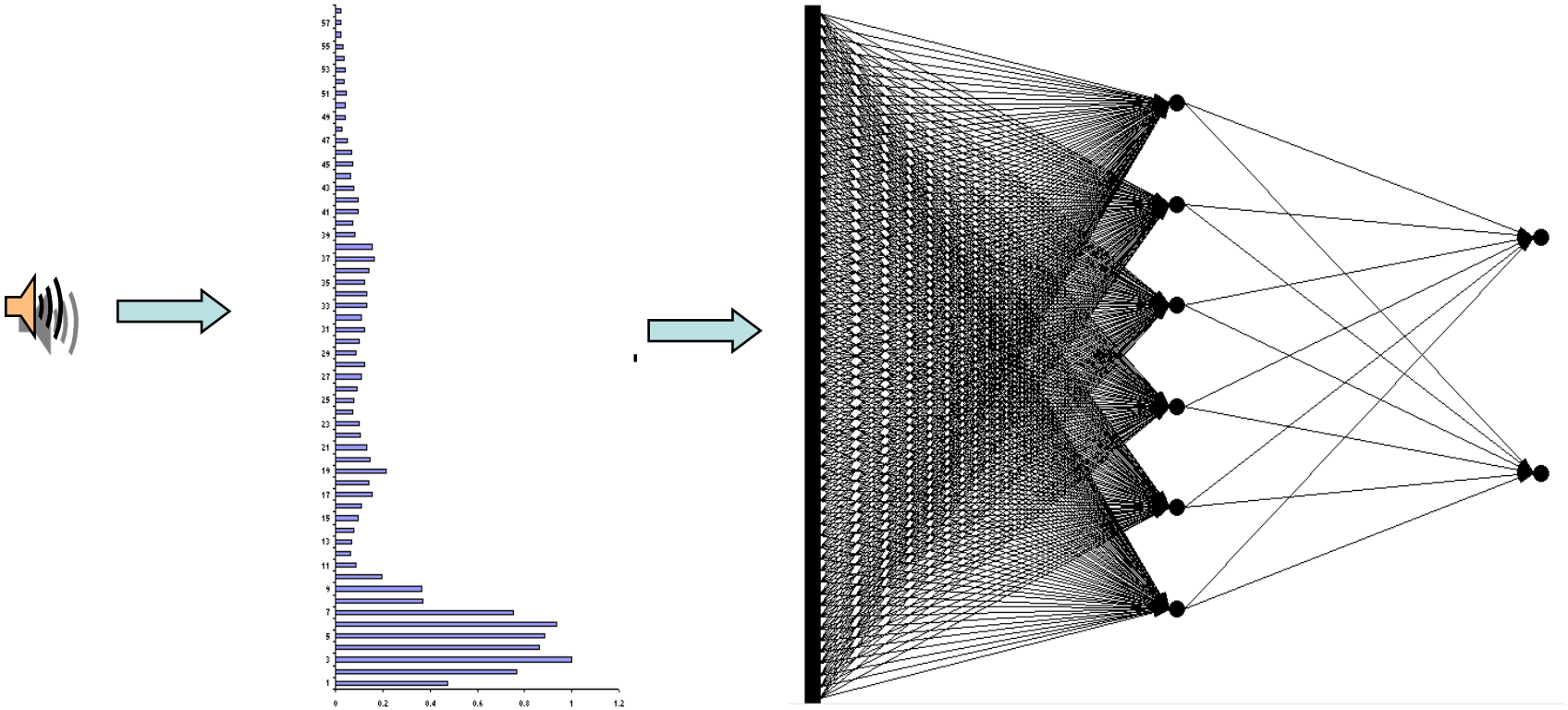
Applications of ANNs

- Signal processing
- Pattern recognition, e.g. handwritten characters or face identification.
- Diagnosis or mapping symptoms to a medical case.
- Speech recognition
- Human Emotion Detection
- Educational Loan Forecasting

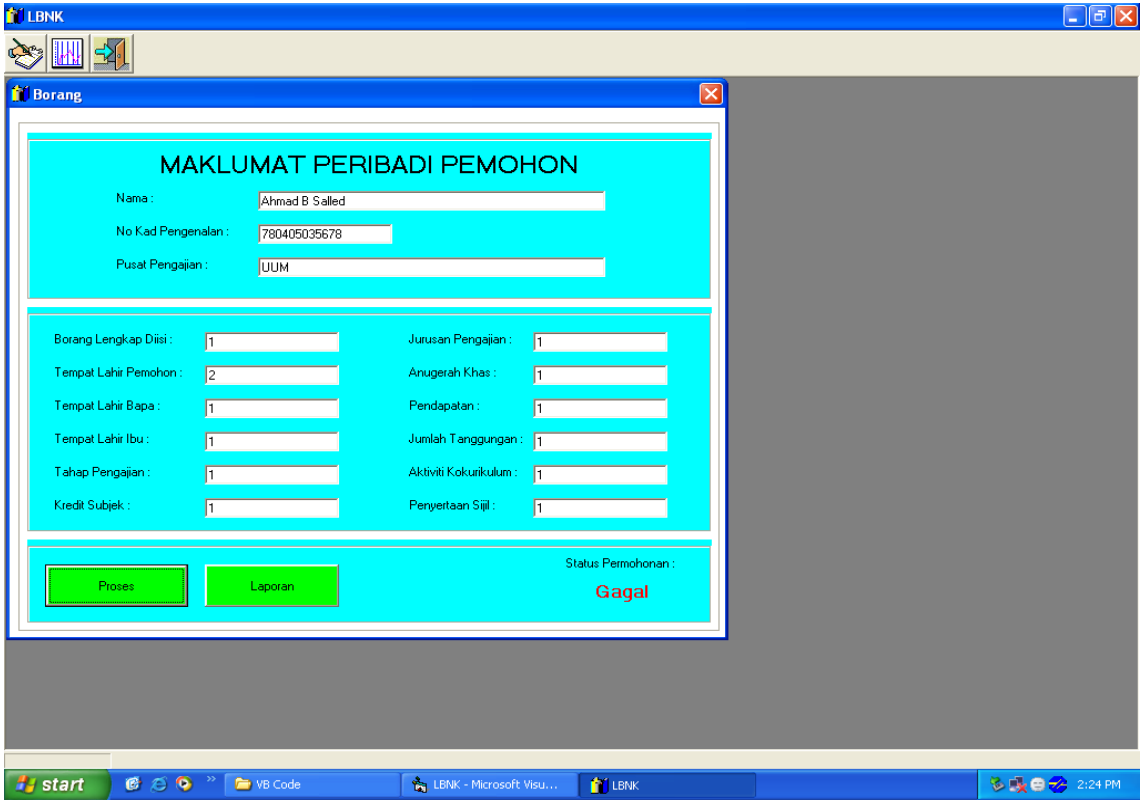
Abdominal Pain Prediction



Voice Recognition



Educational Loan Forecasting System



Advantages Of NN

NON-LINEARITY

It can model non-linear systems

INPUT-OUTPUT MAPPING

It can derive a relationship between a set of input & output responses

ADAPTIVITY

The ability to learn allows the network to adapt to changes in the surrounding environment

EVIDENTIAL RESPONSE

It can provide a confidence level to a given solution

Advantages Of NN

CONTEXTUAL INFORMATION

Knowledge is presented by the structure of the network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, contextual information is dealt with naturally in the network.

FAULT TOLERANCE

Distributed nature of the NN gives it fault tolerant capabilities

NEUROBIOLOGY ANALOGY

Models the architecture of the brain

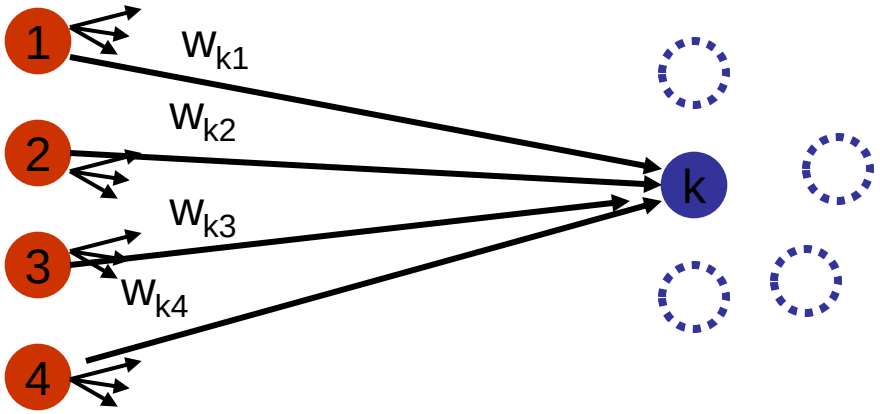
Comparison of ANN with conventional AI methods

CHARACTERISTICS	TRADITIONAL COMPUTING (including Expert Systems)	ARTIFICIAL NEURAL NETWORKS
Processing style	Sequential	Parallel
Functions	Logically (left brained) via Rules Concepts Calculations	Gestault (right brained) via Images Pictures Controls
Learning Method	by rules (didactically)	by example (Socratically)
Applications	Accounting, word processing, math, inventory, digital communications	Sensor processing, speech recognition, pattern recognition, text recognition

UNIT-2

ASSCIATIVE MEMORY AND UNSUPERVISED LEARNING NETWORKS

Weight Vectors in Clustering Networks



- Node k represents a particular class of input vectors, and the weights into k encode a prototype/centroid of that class.
- So if prototype(class(k)) = [i_{k1}, i_{k2}, i_{k3}, i_{k4}], then:

  $w_{km} = f_e(i_{km})$ for m = 1..4, where f_e is the encoding function.
- In some cases, the encoding function involves normalization. Hence $w_{km} = f_e(i_{k1} \ldots i_{k4})$.
- The weight vectors are learned during the unsupervised training phase.

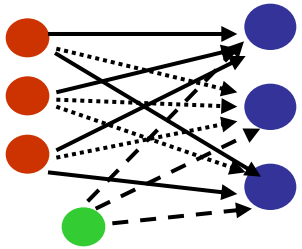
Unsupervised Learning with Artificial Neural Networks

- The ANN is given a set of patterns, P, from space, S, but little/no information about their classification, evaluation, interesting features, etc. It must learn these by itself!

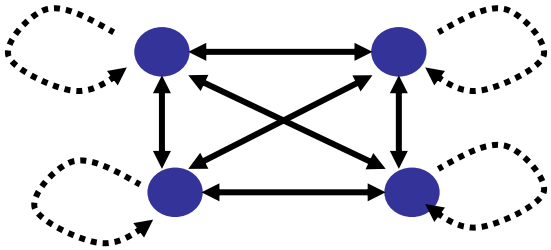
- Tasks
  - Clustering - Group patterns based on similarity (Focus of this lecture)
  - Vector Quantization - Fully divide up S into a small set of regions (defined by codebook vectors) that also helps cluster P.
  - Probability Density Approximation - Find small set of points whose distribution matches that of P.

Network Types for Clustering

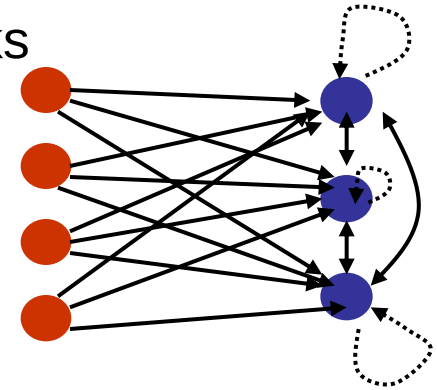
- Winner-Take-All Networks
  - Hamming Networks



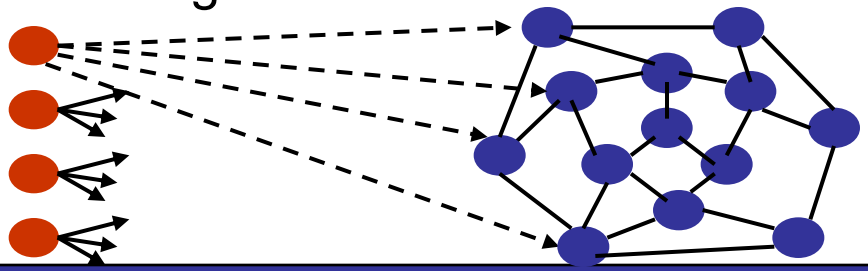
  - Maxnet



  - Simple Competitive Learning Networks



- Topologically Organized Networks
  - Winner & its neighbors “take some”



Hamming Networks

Given: a set of patterns m patterns, P, from an n-dim input space, S.

Create: a network with n input nodes and m simple linear output nodes (one per pattern), where the incoming weights to the output node for pattern p is based on the n features of p.

i_{pj} = jth input bit of the pth pattern. i_{pj} = 1 or -1

Set $w_{pj} = i_{pj}/2$.

Also include a threshold input of -n/2 at each output node.

Testing: enter an input pattern, I, and use the network to determine which member of P that is closest to I. Closeness is based on the Hamming distance (# non-matching bits in the two patterns).

Given input I, the output value of the output node for pattern p =

$$\sum_{k=1}^{n} w_{pk} I_k - \frac{n}{2} \equiv \sum_{k=1}^{n} \frac{i_{pk}}{2} I_k - \frac{n}{2} \equiv \frac{1}{2}\left(\sum_{k=1}^{n} i_{pk} I_k - n \right)$$

Hamming Networks (2)

Proof: (that output of output node p is the negative of the Hamming distance between p and input vector I).

Assume: k bits match.

Then: n - k bits do not match, and n-k is the Hamming distance.

And: the output value of p's output node is:

$$\frac{1}{2}\left(\sum_{k=1}^{n} i_{pk} I_k - n\right) \equiv \frac{1}{2}\left(k - (n-k) - n\right) \equiv k - n \equiv -(n-k)$$

k matches, where each match gives (1)(1) or (-1)(-1) = 1

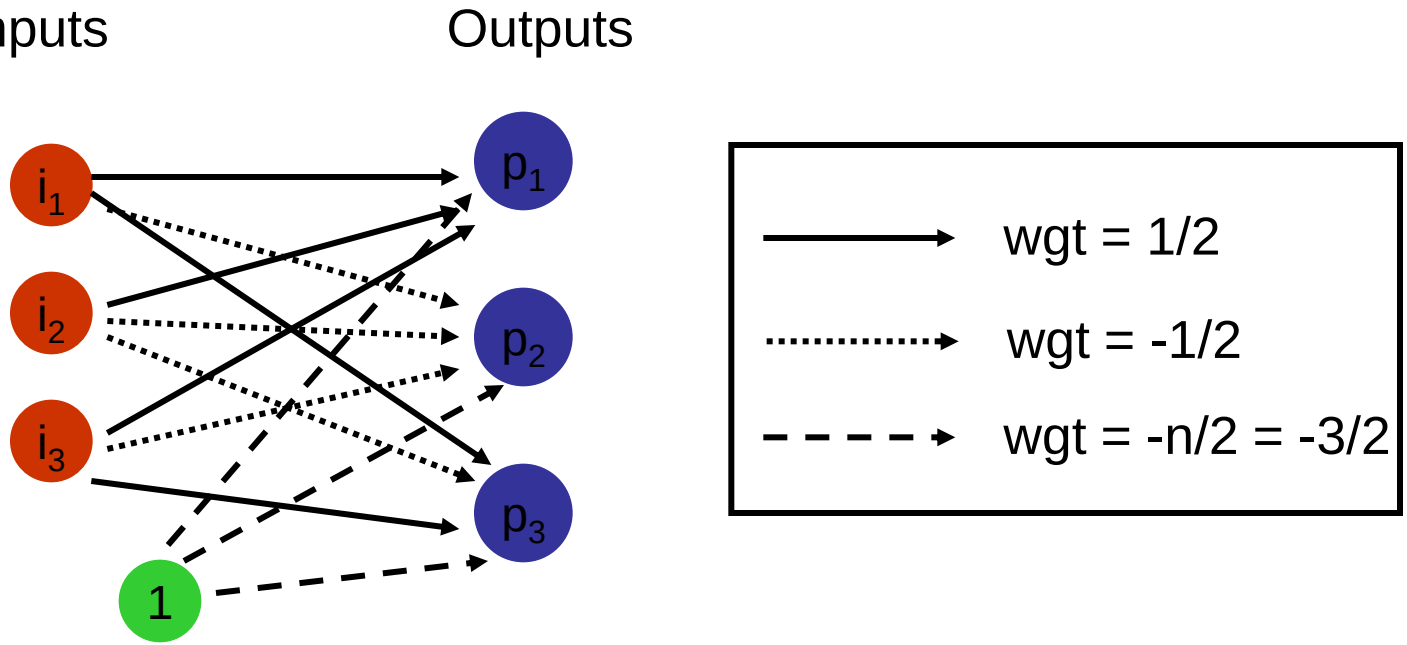
Neg. Hamming distance

n-k mismatches, where each gives (-1)(1) or (1)(-1) = -1

The pattern p* with the largest negative Hamming distance to I is thus the pattern with the smallest Hamming distance to I (i.e. the nearest to I). Hence,
the output node that represents p* will have the highest output value of all

Hamming Network Example

P = {(1 1 1), (-1 -1 -1), (1 -1 1)} = 3 patterns of length 3



Given: input pattern I = (-1 1 1)

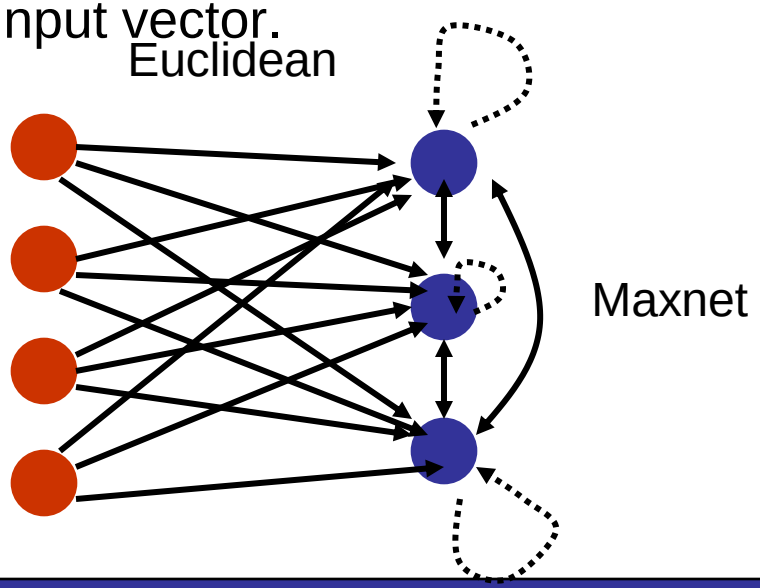
Output (p_1) = -1/2 + 1/2 + 1/2 - 3/2 = -1 (Winner)

Output (p_2) = 1/2 - 1/2 - 1/2 - 3/2 = -2

Output (p_3) = -1/2 - 1/2 + 1/2 - 3/2 = -2

Simple Competitive Learning

- Combination Hamming-like Net + Maxnet with learning of the input-to-output weights.
  - Inputs can be real valued, not just 1, -1.
    - So distance metric is actually Euclidean or Manhattan, not Hamming.
- Each output node represents a centroid for input patterns it wins on.
- Learning: winner node's incoming weights are updated to move closer to the input vector.



Winning & Learning

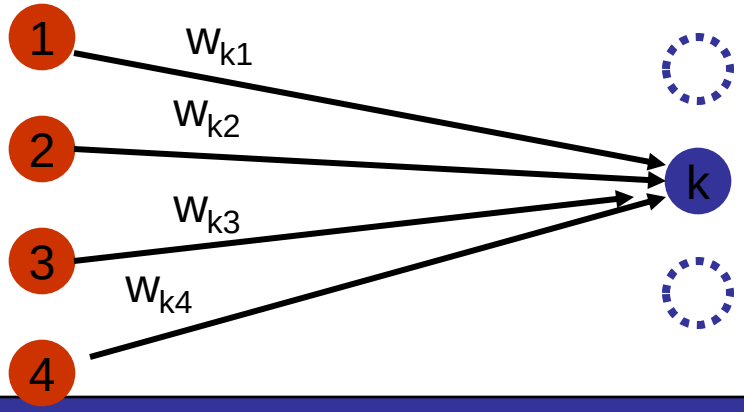
``*Winning isn't everything…it's the ONLY thing"* - Vince Lombardi

- Only the incoming weights of the winner node are modified.
- Winner = output node whose incoming weights are the shortest Euclidean distance from the input vector.

$$\sqrt{\sum_{i=1}^{n}(I_i - w_{ki})^2}$$ = Euclidean distance from input vector I to the vector represented by output node k's incoming weights

- <u>Update formula:</u> If j is the winning output node:

$$\forall i : w_{ji}(new) = w_{ji}(old) + \eta(I_i - w_{ji}(old))$$



Note: The use of real-valued inputs & Euclidean distance means that the simple product of weights and inputs does not correlate with "closeness" as in binary networks using Hamming distance.

SCL Examples (1)

6 Cases:

(0 1 1) (1 1 0.5)
(0.2 0.2 0.2) (0.5 0.5 0.5)
(0.4 0.6 0.5) (0 0 0)

Learning Rate: 0.5

Initial Randomly-Generated Weight Vectors:

[0.14 0.75 0.71]
[0.99 0.51 0.37] Hence, there are 3 classes to be learned
[0.73 0.81 0.87]

Training on Input Vectors

Input vector # 1: [0.00 1.00 1.00]
Winning weight vector # 1: [0.14 0.75 0.71] Distance: 0.41
Updated weight vector: [0.07 0.87 0.85]
Input vector # 2: [1.00 1.00 0.50]
Winning weight vector # 3: [0.73 0.81 0.87] Distance: 0.50
Updated weight vector: [0.87 0.90 0.69]

SCL Examples (2)

```
Input vector # 3:   [ 0.20  0.20  0.20 ]
   Winning weight vector # 2: [ 0.99  0.51  0.37 ] Distance:  0.86
   Updated weight vector: [ 0.59  0.36  0.29 ]
Input vector # 4:   [ 0.50  0.50  0.50 ]
   Winning weight vector # 2: [ 0.59  0.36  0.29 ] Distance:  0.27
   Updated weight vector: [ 0.55  0.43  0.39 ]
Input vector # 5:   [ 0.40  0.60  0.50 ]
   Winning weight vector # 2: [ 0.55  0.43  0.39 ] Distance:  0.25
   Updated weight vector: [ 0.47  0.51  0.45 ]
Input vector # 6:   [ 0.00  0.00  0.00 ]
   Winning weight vector # 2: [ 0.47  0.51  0.45 ] Distance:  0.83
   Updated weight vector: [ 0.24  0.26  0.22 ]
```

Weight Vectors after epoch 1:

```
  [ 0.07  0.87  0.85 ]
  [ 0.24  0.26  0.22 ]
  [ 0.87  0.90  0.69 ]
```

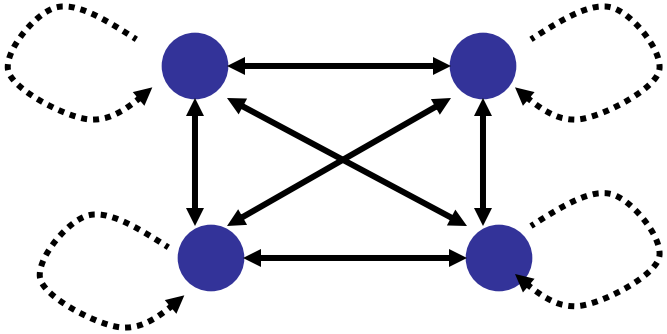
SCL Examples (3)

```
Clusters after epoch 1:
 Weight vector # 1: [ 0.07  0.87  0.85 ]
    Input vector # 1:   [ 0.00  1.00  1.00 ]
 Weight vector # 2: [ 0.24  0.26  0.22 ]
    Input vector # 3:   [ 0.20  0.20  0.20 ]
    Input vector # 4:   [ 0.50  0.50  0.50 ]
    Input vector # 5:   [ 0.40  0.60  0.50 ]
    Input vector # 6:   [ 0.00  0.00  0.00 ]
 Weight vector # 3: [ 0.87  0.90  0.69 ]
    Input vector # 2:   [ 1.00  1.00  0.50 ]

Weight Vectors after epoch 2:
  [ 0.03  0.94  0.93 ]
  [ 0.19  0.24  0.21 ]
  [ 0.93  0.95  0.59 ]

Clusters after epoch 2:
   unchanged.
```

Maxnet



$wgt = -\varepsilon$

$wgt = \theta$

e.g. $\theta = 1 \wedge \varepsilon \leq \frac{1}{n}$

Simple network to find node with largest initial input value.

Topology: clique with self-arcs, where all self-arcs have a small positive (excitatory) weight, and all other arcs have a small negative (inhibitory) weight.

Nodes: have transfer function f_T = max(sum, 0)

Algorithm:

Load initial values into the clique

Repeat:

Synchronously update all node values via f_T

Until: all but one node has a value of 0

Winner = the non-zero node

Maxnet Examples

- Input values: (1, 2, 5, 4, 3) with epsilon = 1/5 and theta = 1

0.000	0.000	3.000	1.800	0.600
0.000	0.000	2.520	1.080	0.000
0.000	0.000	2.304	0.576	0.000
0.000	0.000	2.189	0.115	0.000
0.000	0.000	2.166	0.000	0.000
0.000	0.000	2.166	0.000	0.000

= (1)5 - (0.2)(1+2+4+3)

Stable attractor

- Input values: (1, 2, 5, 4.5, 4.7) with epsilon = 1/5 and theta = 1

0.000	0.000	2.560	1.960	2.200
0.000	0.000	1.728	1.008	1.296
0.000	0.000	1.267	0.403	0.749
0.000	0.000	1.037	0.000	0.415
0.000	0.000	0.954	0.000	0.207
0.000	0.000	0.912	0.000	0.017
0.000	0.000	0.909	0.000	0.000
0.000	0.000	0.909	0.000	0.000

Stable attractor

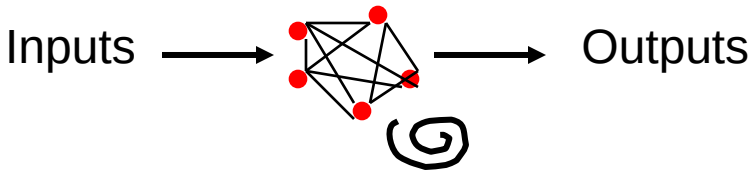
Associative-Memory Networks

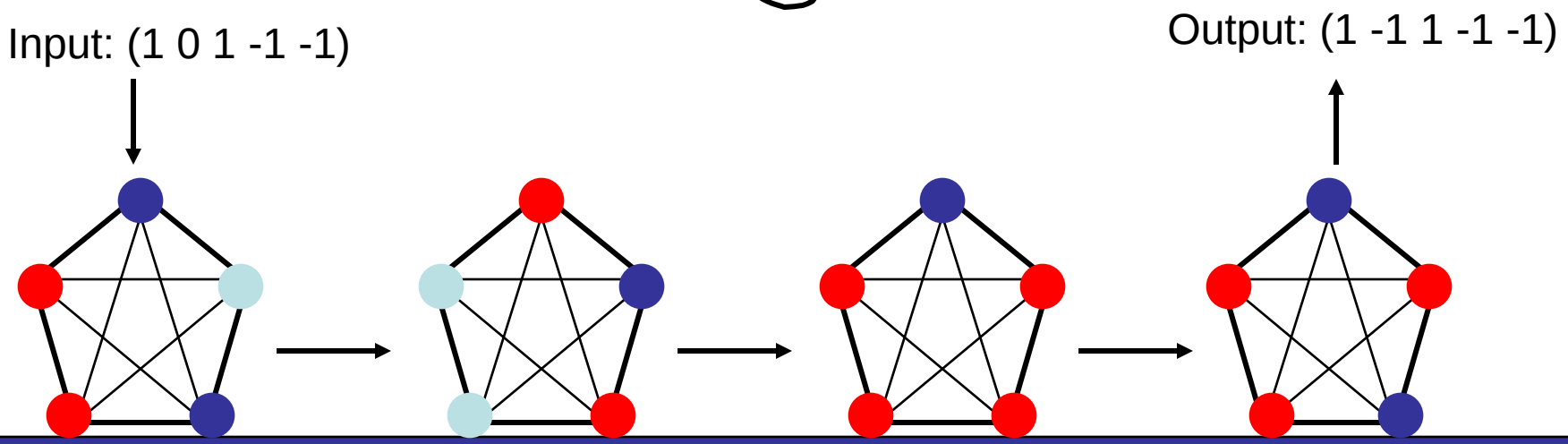
Input: Pattern (often noisy/corrupted)

Output: Corresponding pattern (complete / relatively noise-free)

Process

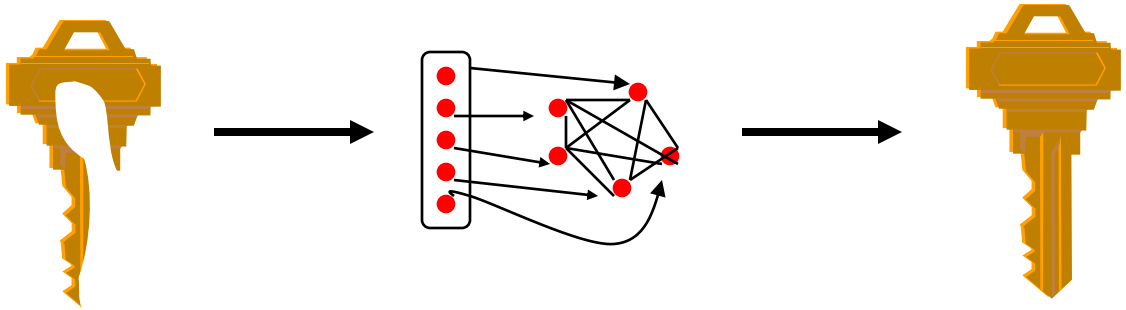
1. Load input pattern onto core group of highly-interconnected neurons.
2. Run core neurons until they reach a steady state.
3. Read output off of the states of the core neurons.





Associative Network Types

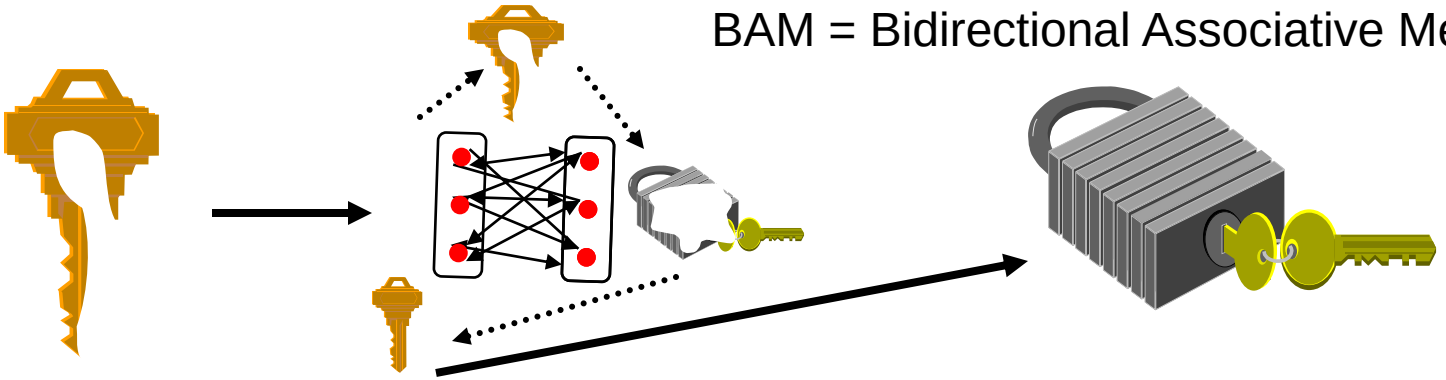
1. Auto-associative: X = Y



*Recognize noisy versions of a pattern

2. Hetero-associative Bidirectional: X <> Y

BAM = Bidirectional Associative Memory



*Iterative correction of input and output

Hebb's Rule

Connection Weights ~ Correlations

``When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell." (Hebb, 1949)

In an associative neural net, if we compare two pattern components (e.g. pixels within many patterns and find that they are frequently in:

a) the same state, then the arc weight between their NN nodes should be pos

b) different states, then ” ” ” negati

Matrix Memory:

The weights must store the average correlations between all pattern compon across all patterns. A net presented with a partial pattern can then use the corr to recreate the entire pattern.

Quantifying Hebb's Rule

Compare two nodes to calc a weight change that reflects the state correlation:

Auto-Association: $\Delta w_{jk} \propto i_{pk} i_{pj}$

* When the two components are the same (different),
increase (decrease) the weight

Hetero-Association: $\Delta w_{jk} \propto i_{pk} o_{pj}$

i = input component
o = output component

Ideally, the weights will record the average correlations across all patterns:

Auto: $w_{jk} \propto \sum_{p=1}^{P} i_{pk} i_{pj}$

Hetero: $w_{jk} \propto \sum_{p=1}^{P} i_{pk} o_{pj}$

<u>Hebbian Principle:</u> If all the input patterns are known prior to retrieval time, then init weights as:

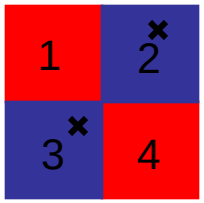
Auto: $w_{jk} \equiv \frac{1}{P} \sum_{p=1}^{P} i_{pk} i_{pj}$

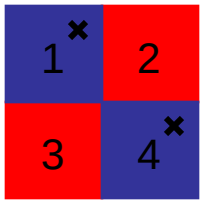
Hetero: $w_{jk} \equiv \frac{1}{P} \sum_{p=1}^{P} i_{pk} o_{pj}$

Weights = Average Correlations

Auto-Associative Memory

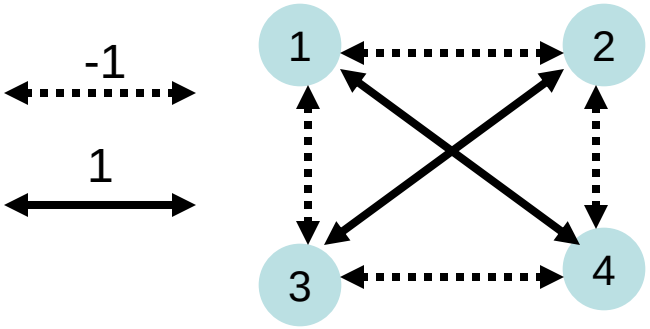
1. Auto-Associative Patterns to Remember





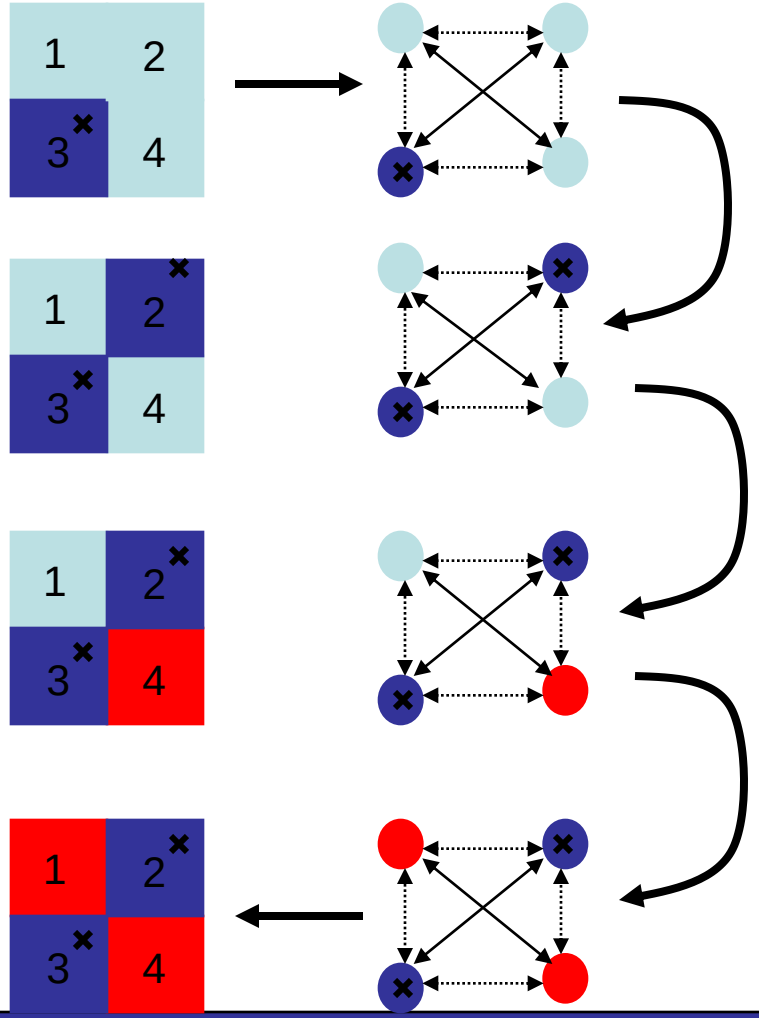
Comp/Node value legend:
dark (blue) with x => +1
dark (red) w/o x => -1
light (green) => 0

2. Distributed Storage of All Patterns:



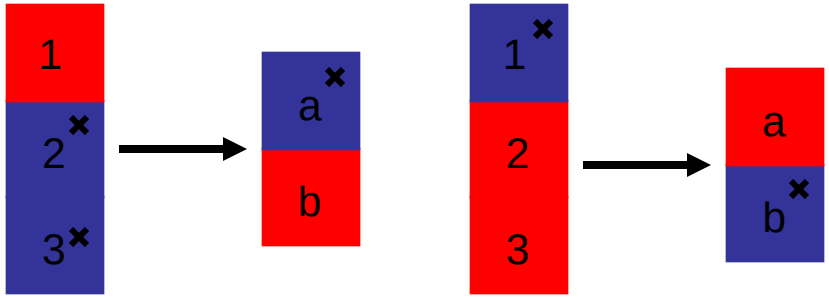
- 1 node per pattern unit
- Fully connected: clique
- Weights = avg correlations across all patterns of the corresponding units

3. Retrieval

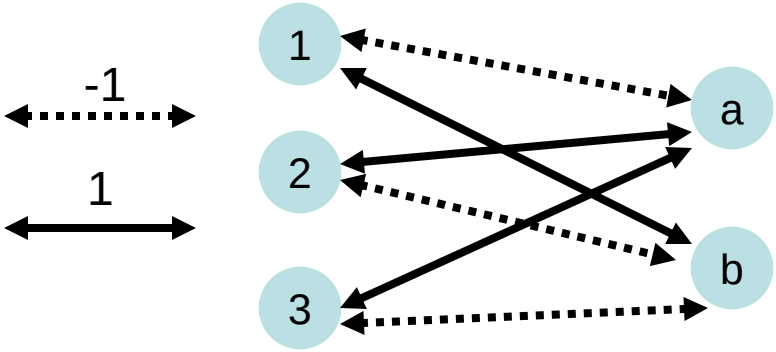


Hetero-Associative Memory

1. Hetero-Associative Patterns (Pairs) to Remember

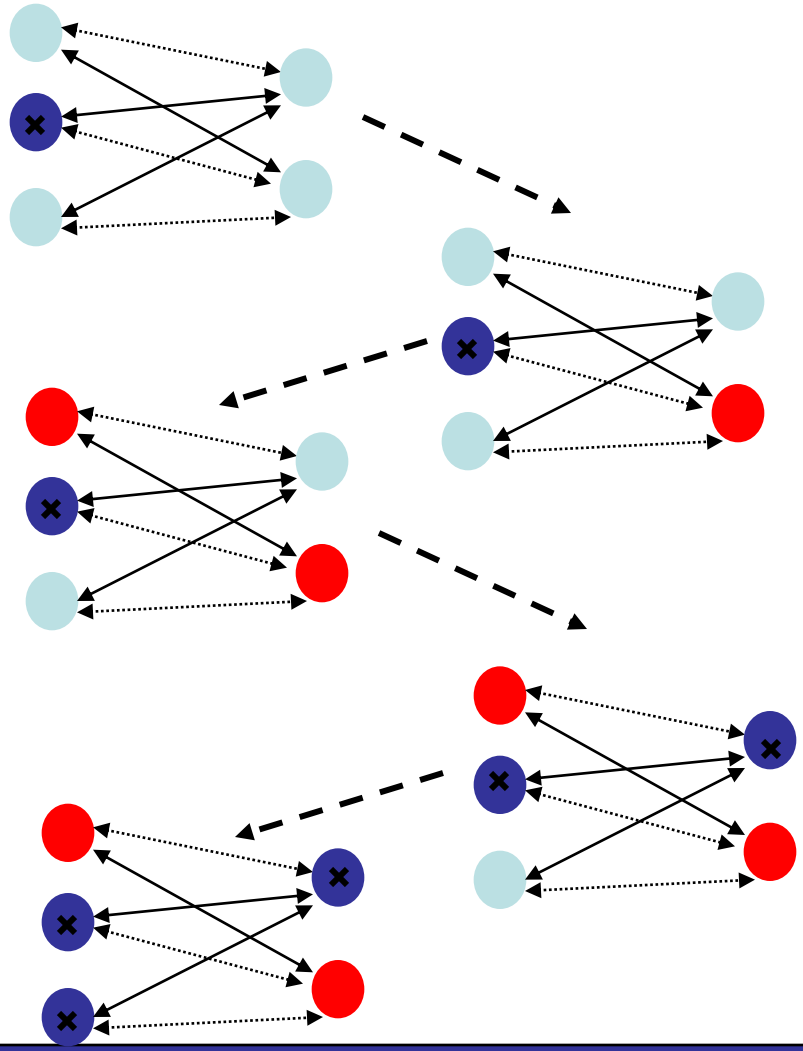


2. Distributed Storage of All Patterns:



- 1 node per pattern unit for X & Y
- Full inter-layer connection
- Weights = avg correlations across all patterns of the corresponding units

3. Retrieval

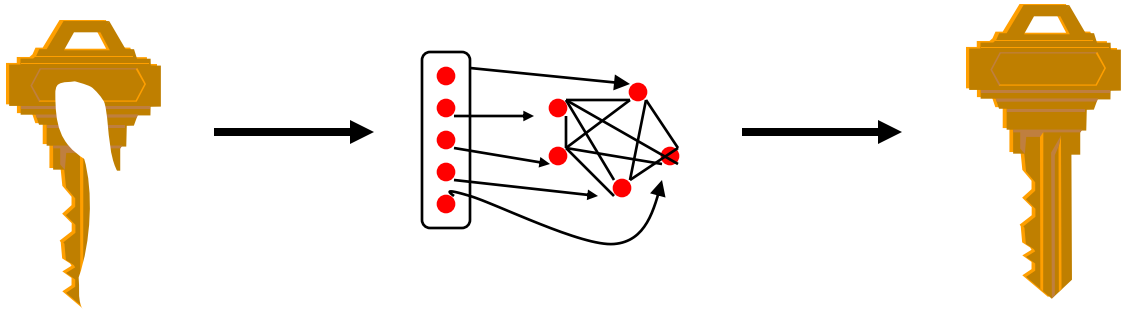


Hopfield Networks

- Auto-Association Network
- Fully-connected (clique) with symmetric weights
- State of node = f(inputs)
- Weight values based on Hebbian principle
- Performance: Must iterate a bit to converge on a pattern, but generally

  much less computation than in back-propagation networks.

Input Output (after many iterations)

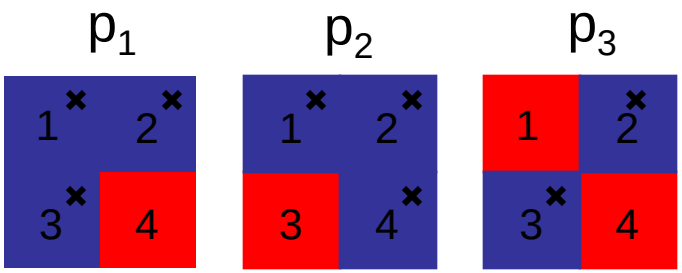


Discrete node update rule: $x_{pk}(t+1) = \mathrm{sgn}(\sum_{j=1}^{n} w_{kj} x_{pj}(t) + I_{pk})$

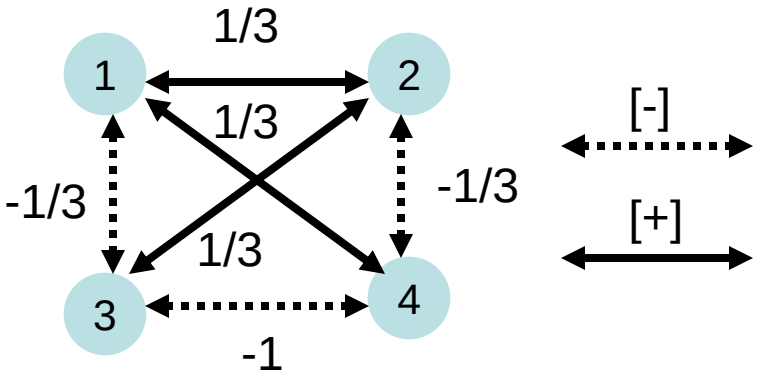
Input value

Hopfield Network Example

1. Patterns to Remember



3. Build Network

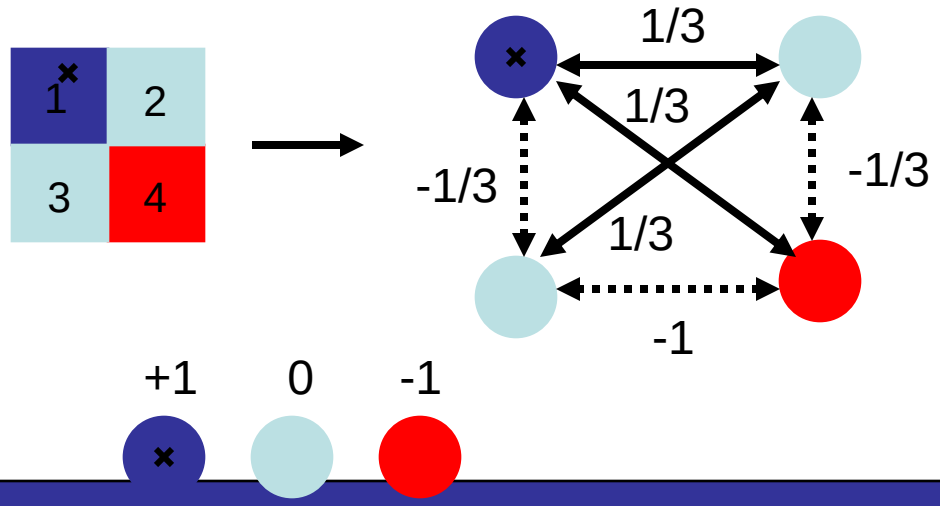


2. Hebbian Weight Init:

Avg Correlations across 3 patterns

	p_1	p_2	p_3	Avg
W_{12}	1	1	-1	1/3
W_{13}	1	-1	-1	-1/3
W_{14}	-1	1	1	1/3
W_{23}	1	-1	1	1/3
W_{24}	-1	1	-1	-1/3
W_{34}	-1	-1	-1	-1

4. Enter Test Pattern



Storage Capacity of Hopfield Networks

Capacity = Relationship between # patterns that can be stored & retrieved

without error to the size of the network.

Capacity = # patterns / # nodes or # patterns / # weights

- If we use the following definition of 100% correct retrieval:

  When any of the stored patterns is entered completely (no noise), then that same pattern is returned by the network; i.e. The pattern is a stable attractor.

- A detailed proof shows that a Hopfield network of N nodes can achieve 100% correct retrieval on P patterns if: $P < N/(4*\ln(N))$

In general, as more patterns are added to a network, the avg correlations will be less likely to match the correlations in any particular pattern. Hence, the likelihood of retrieval error will increase.

=> The key to perfect recall is selective ignorance!!

N	Max P
10	1
100	5
1000	36
10000	271
10^{11}	10^{9}

Stochastic Hopfield Networks

Node state is stochastically determined by sum of inputs:

Node fires with probability:

$$p \equiv \frac{1}{1+e^{-2\beta sum_k}}$$

For these networks, effective retrieval is obtained when P < 0.138N, which is an improvement over standard Hopfield nets.

<u>Boltzmann Machines:</u>

Similar to Hopfield nets but with hidden layers.

State changes occur either:

a. Deterministically when $\Delta E < 0$

b. Stochastically with probability = $\frac{1}{1+e^{-\Delta E/\tau}}$

Where t is a decreasing temperature variable and ΔE is the expected change in energy if the change is made.

The non-deterministic allows the system to ”jiggle” out of local minima

Unit -3

Fuzzy logic

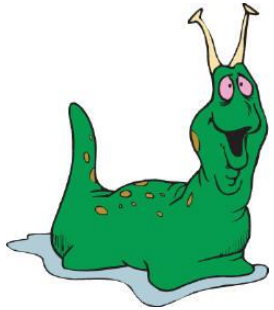
Overview

- What is Fuzzy Logic?
- Where did it begin?
- Fuzzy Logic vs. Neural Networks
- Fuzzy Logic in Control Systems
- Fuzzy Logic in Other Fields
- Future

WHAT IS FUZZY LOGIC?

- Definition of fuzzy
  - Fuzzy – “not clear, distinct, or precise; blurred”
- Definition of fuzzy logic
  - A form of knowledge representation suitable for notions that cannot be defined precisely, but which depend upon their contexts.

TRADITIONAL REPRESENTATION OF LOGIC



Slow

Speed = 0

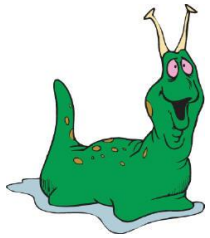
Fast

Speed = 1

```
bool speed;
get the speed
if ( speed == 0) {
//  speed is slow
}
else {
//  speed is fast
}
```

FUZZY LOGIC REPRESENTATION

- For every problem must represent in terms of fuzzy sets.
- What are fuzzy sets?



Slowest

[0.0 – 0.25]

Slow

[0.25 – 0.50]

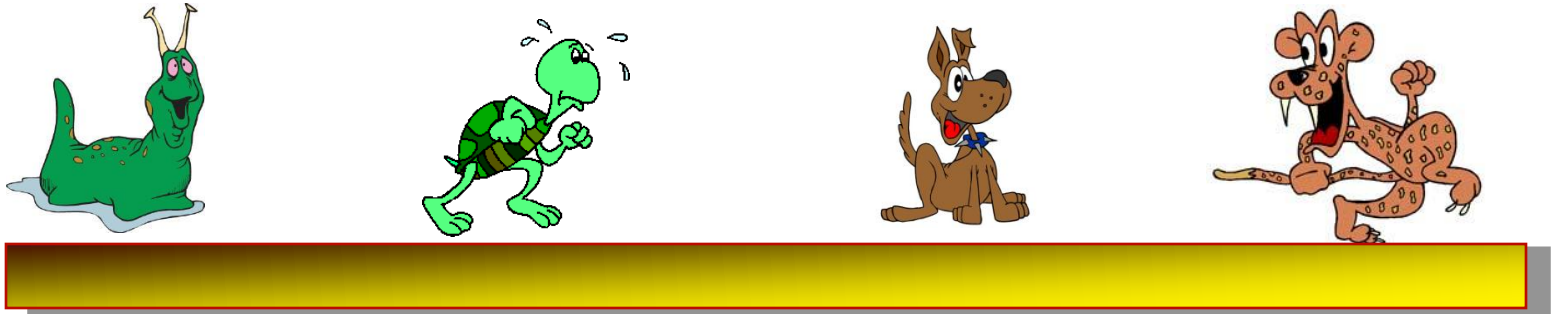
Fast

[0.50 – 0.75]

Fastest

[0.75 – 1.00]

FUZZY LOGIC REPRESENTATION CONT.



Slowest Slow Fast Fastest

```
float speed;
get the speed
if ((speed >= 0.0)&&(speed < 0.25)) {
//  speed is slowest
}
else if ((speed >= 0.25)&&(speed < 0.5))
{
//  speed is slow
}
else if ((speed >= 0.5)&&(speed < 0.75))
{
//  speed is fast
}
else // speed >= 0.75 && speed < 1.0
{
//  speed is fastest
}
```

ORIGINS OF FUZZY LOGIC

- Traces back to Ancient Greece
- Lotfi Asker Zadeh (1965)
  - First to publish ideas of fuzzy logic.
- Professor Toshire Terano (1972)
  - Organized the world's first working group on fuzzy systems.
- F.L. Smidth & Co. (1980)
  - First to market fuzzy expert systems.

FUZZY LOGIC VS. NEURAL NETWORKS

- How does a Neural Network work?
- Both model the human brain.
  - Fuzzy Logic
  - Neural Networks
- Both used to create behavioral systems.

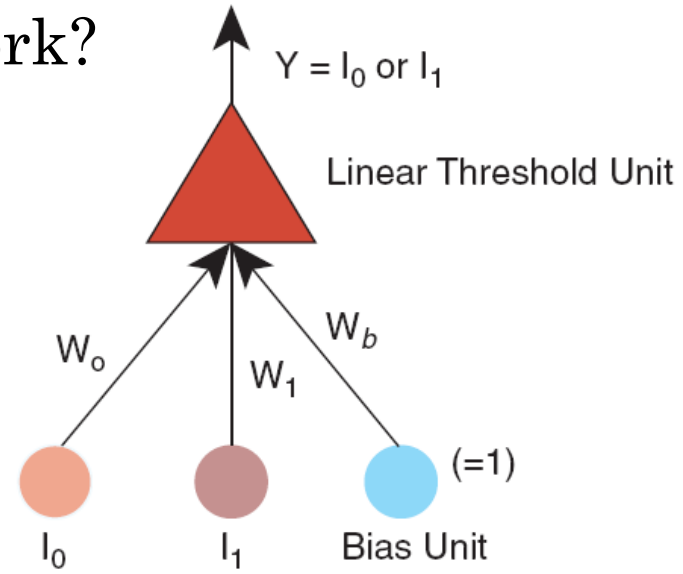


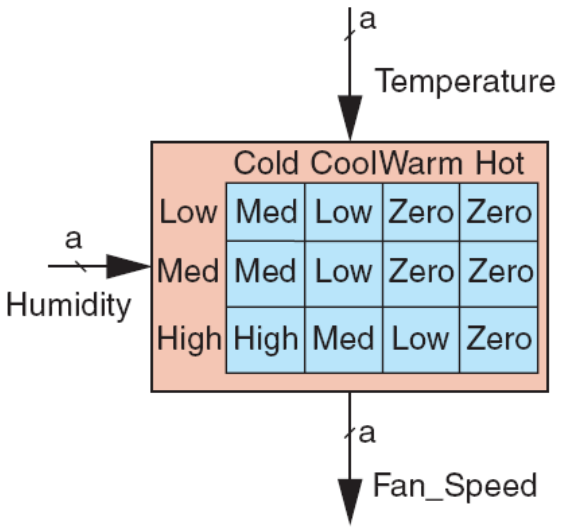
Fig. 2 A simple, single-unit adaptive network

FUZZY LOGIC IN CONTROL SYSTEMS

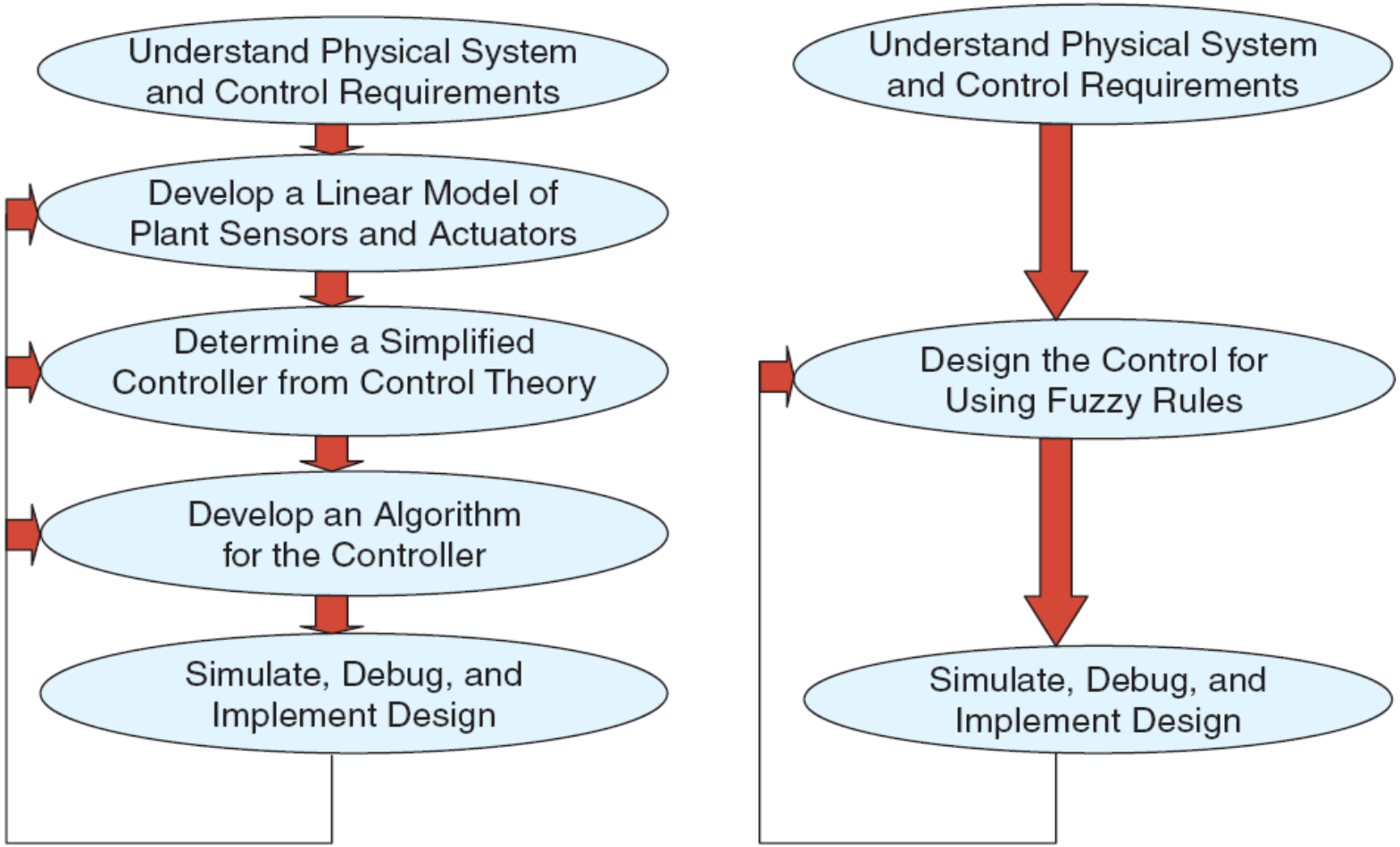
- Fuzzy Logic provides a more efficient and resourceful way to solve Control Systems.
- Some Examples
  - Temperature Controller
  - Anti – Lock Break System (ABS)

TEMPERATURE CONTROLLER

- The problem
  - Change the speed of a heater fan, based off the room temperature and humidity.
- A temperature control system has four settings
  - Cold, Cool, Warm, and Hot
- Humidity can be defined by:
  - Low, Medium, and High
- Using this we can define the fuzzy set.



BENEFITS OF USING FUZZY LOGIC



FUZZY LOGIC IN OTHER FIELDS

- Business
- Hybrid Modeling
- Expert Systems

Fuzzy Logic Example

Automotive Speed Controller

3 inputs:

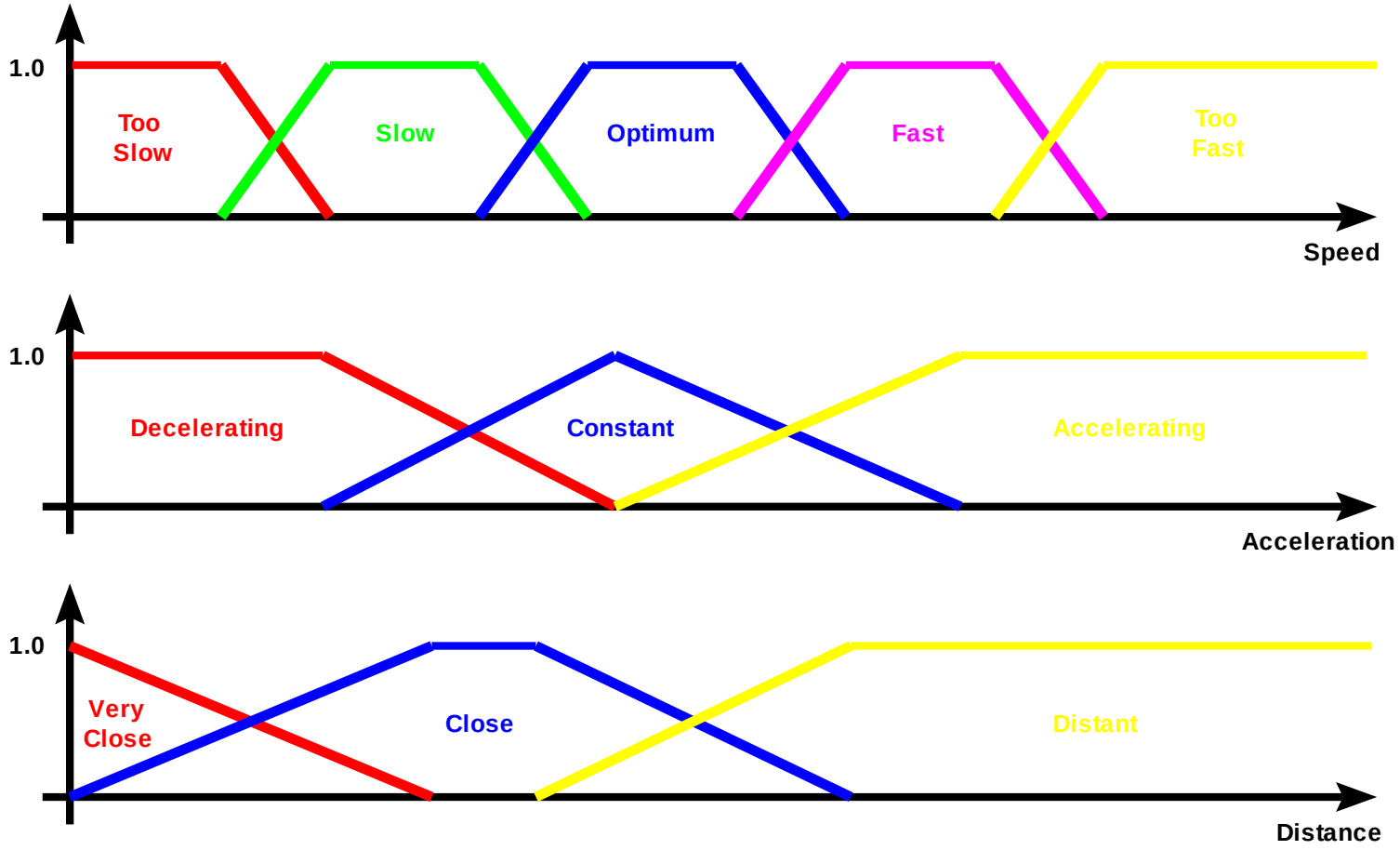
- speed (5 levels)
- acceleration (3 levels)
- distance to destination (3 levels)

1 output:

- power (fuel flow to engine)

Set of rules to determine output based on input values

Fuzzy Logic Example



Fuzzy Logic Example

Example Rules

IF speed is TOO SLOW and acceleration is DECELERATING,
THEN INCREASE POWER GREATLY

IF speed is SLOW and acceleration is DECREASING,
THEN INCREASE POWER SLIGHTLY

IF distance is CLOSE,
THEN DECREASE POWER SLIGHTLY

. . .

Fuzzy Logic Example

Note there would be a total of 95 different rules for all combinations of inputs of 1, 2, or 3 at a time.

(5x3x3 + 5x3 + 5x3 + 3x3 + 5 + 3 + 3)

In practice, a system won't require all the rules.

System tweaked by adding or changing rules and by adjusting set boundaries.

System performance can be very good but not usually optimized by traditional metrics (minimize RMS error).

Fuzzy Logic Summary

Doesn't require an understanding of process but any knowledge will help formulate rules.

Complicated systems may require several iterations to find a set of rules resulting in a stable system.

Combining Neural Networks with fuzzy logic reduces time to establish rules by analyzing clusters of data.

Possible applications: Master Production Schedule, Material Requirements Planning, Inventory Capacity Planning

CONCLUSION

- Fuzzy logic provides an alternative way to represent linguistic and subjective attributes of the real world in computing.
- It is able to be applied to control systems and other applications in order to improve the efficiency and simplicity of the design process.

UNIT-4

FUZZY AIRTHEMATIC

Definition

- Fuzzy Number
  - Convex and normal fuzzy set defined on R
  - Equivalently it satisfies
    - Normal fuzzy set on R
    - Every alpha-cut must be a closed interval
    - Support must be bounded
- Applications of fuzzy number
  - Fuzzy Control, Decision Making, Optimizations

Arithmetic Operations

- Interval Operations $A = [a_1, a_3]$, $B = [b_1, b_3]$

Addition

$$[a_1, a_3](+)[b_1, b_3] = [a_1 + b_1, a_3 + b_3]$$

Subtraction

$$[a_1, a_3](-)[b_1, b_3] = [a_1 - b_3, a_3 - b_1]$$

Multiplication

$$[a_1, a_3](\bullet)[b_1, b_3] = [a_1 \bullet b_1 \wedge a_1 \bullet b_3 \wedge a_3 \bullet b_1 \wedge a_3 \bullet b_3,$$
$$a_1 \bullet b_1 \vee a_1 \bullet b_3 \vee a_3 \bullet b_1 \vee a_3 \bullet b_3]$$

Division

$$[a_1, a_3](/)[b_1, b_3] = [a_1 / b_1 \wedge a_1 / b_3 \wedge a_3 / b_1 \wedge a_3 / b_3,$$
$$a_1 / b_1 \vee a_1 / b_3 \vee a_3 / b_1 \vee a_3 / b_3]$$
$$\text{execpt } b_1 = b_3 = 0$$

Examples

- Addition

  [2,5]+[1,3]=[3,8] [0,1]+[-6,5]=[-6,6]

- Subtraction

  [2,5]-[1,3]=[-1,4] [0,1]-[-6,5]=[-5,7]

- Multiplication

  [-1,1]*[-2,-0.5]=[-2,2] [3,4]*[2,2]=[6,8]

- Division

  [-1,1]/[-2,-0.5]=[-2,2] [4,10]*[1,2]=[2,10]

Properties of Interval Operations

Commutative

$A + B = B + A \qquad A \cdot B = B \cdot A$

Assocoative

$(A + B) + C = A + (B + C) \quad (A \cdot B) \cdot C = A \cdot (B \cdot C)$

Identity $\quad 0 = [0,0] \quad 1 = [1,1]$

$A = A + 0 = 0 + A \qquad A = A \cdot 1 = 1 \cdot A$

Subdistributive

$A \cdot (B + C) \subseteq A \cdot B + A \cdot C$

Inverse

$0 \in A\text{-}A \quad 1 \in A/A$

Monotonicity for any operations *

If $A \subseteq E$ and $B \subseteq F$ then $A*B \subseteq E*F$

Arithmetic Operation on Fuzzy Numbers

- Interval operations of alpha-level sets

$$^{\alpha}(A * B) = {}^{\alpha}A *^{\alpha} B \text{ for any } \alpha \in (0,1].$$

When $* = /$, $0 \notin {}^{\alpha}B$ for all $\alpha \in (0,1]$.

$$A * B = \bigcup_{\alpha \in (0,1]} {}_{\alpha}(A * B)$$

- Note: The Result is a fuzzy number.
- Example: See Text pp. 105 and Fig. 4.5

Example

- $A+B = \{1/5, 0.8/6, 0.5/7\}$

$i) z < 5$ No such case.

$\mu_{A(+)B}(z) = 0$

$ii) z = 5$

$x + y = 2 + 3$

$\mu_A(2) \wedge \mu_B(3) = 1$

$iii) z = 6$

$x + y = 3 + 3$ or $x + y = 2 + 4$

$\mu_A(3) \wedge \mu_B(3) = 0.5$

$\mu_A(2) \wedge \mu_B(4) = 0.8$ $\mu_{A(+)B}(6) = \underset{\substack{6=3+3 \\ 6=2+4}}{\vee} (0.5,\ 0.8) = 0.8$

$iv) z = 7$

$\mu_A(3) \wedge \mu_B(4) = \min(0.5,\ 0.8) = 0.5$

Example

- Max (A,B) = { (3 , 1) , (4 , 0.5) }

$i) z \le 2$ No such case.

$$\mu_{A(\vee)B}(z) = 0$$

$ii) z = 3$

$$x \vee y = 2 \vee 3 \quad \text{or} \quad x \vee y = 3 \vee 3$$

$$\mu_A(2) \wedge \mu_B(3) = 1 \wedge 1 = 1 \qquad \mu_A(3) \wedge \mu_B(3) = 0.5 \wedge 1 = 0.5$$

$$\mu_{A(\vee)B}(3) = \underset{\substack{3=2\vee 3 \\ 3=3\vee 3}}{\vee}(1, 0.5) = 1$$

$iii) z = 4$

$$x \vee y = 2 \vee 4 \quad \text{or} \quad x \vee y = 3 \vee 4$$

$$\mu_A(2) \wedge \mu_B(4) = 1 \wedge 0.5 = 0.5 \qquad \mu_A(3) \wedge \mu_B(4) = 0.5 \wedge 0.5 = 0.5$$

$$\mu_{A(\vee)B}(4) = \underset{\substack{4=2\vee 4 \\ 4=3\vee 4}}{\vee}(0.5, 0.5) = 0.5$$

$iv) z \ge 5$ No such case.

$$\mu_{A(\vee)B}(z) = 0$$

Typical Fuzzy Numbers

- **Triangular Fuzzy Number**
  - Fig. 4.5
- **Trapezoidal Fuzzy Numbers: Fig. 4.4**
  - Linguistic variable: ”Performance”
  - Linguistic values (terms):

    “very small” … “very large”
  - Semantic Rules:

    Terms maps on trapezoidal fuzzy numbers
  - Syntactic rules (grammar):

    Rules for other terms such as “not small”

Lattice of Fuzzy Numbers

- Lattice
  - Partially ordered set with ordering relation
  - Meet(g.l.b) and Join(l.u.b) operations
  - Example:

    Real number and “less than or equal to”
- Lattice of fuzzy numbers

$$MIN(A,B)(z) = \sup_{z=\min(x,y)} \min[A(x), B(y)] = MEET(A,B)$$

$$MAX(A,B)(z) = \sup_{z=\max(x,y)} \min[A(x), B(y)] = JOIN(A,B)$$

Fuzzy Equations

- Addition $A + X = B$
  - X = B-A is not a solution because A+(B-A) is not B.
  - Conditions to have a solution

For any $\alpha \in (0,1]$, let ${}^{\alpha}A = [{}^{\alpha}a_1, {}^{\alpha}a_2]$, ${}^{\alpha}B = [{}^{\alpha}b_1, {}^{\alpha}b_2]$ and ${}^{\alpha}X = [{}^{\alpha}x_1, {}^{\alpha}x_2]$

(i) ${}^{\alpha}b_1 - {}^{\alpha}a_1 \le {}^{\alpha}b_2 - {}^{\alpha}a_2$ for any $\alpha \in (0,1]$

(ii) $\alpha \le \beta$ implies ${}^{\alpha}b_1 - {}^{\alpha}a_1 \le {}^{\beta}b_1 - {}^{\beta}a_1 \le {}^{\beta}b_2 - {}^{\beta}a_2 \le {}^{\alpha}b_2 - {}^{\alpha}a_2$

  - Solution

Suppose ${}^{\alpha}X = [{}^{\alpha}b_1 - {}^{\alpha}a_1, {}^{\alpha}b_2 - {}^{\alpha}a_2]$ is a solution of ${}^{\alpha}A + {}^{\alpha}X = {}^{\alpha}B$ for any $\alpha \in (0,1]$. Then

$$X = \bigcup_{\alpha \in (0,1]} {}_{\alpha}X$$

Fuzzy Equations

- Multiplication $A \cdot X = B$
  - X = B/A is not a solution.
  - Conditions to have a solution

For any $\alpha \in (0,1]$, let ${}^{\alpha}A = [{}^{\alpha}a_1, {}^{\alpha}a_2]$, ${}^{\alpha}B = [{}^{\alpha}b_1, {}^{\alpha}b_2]$ and ${}^{\alpha}X = [{}^{\alpha}x_1, {}^{\alpha}x_2]$

(i) ${}^{\alpha}b_1 / {}^{\alpha}a_1 \leq {}^{\alpha}b_2 / {}^{\alpha}a_2$ for any $\alpha \in (0,1]$

(ii) $\alpha \leq \beta$ implies ${}^{\alpha}b_1 / {}^{\alpha}a_1 \leq {}^{\beta}b_1 / {}^{\beta}a_1 \leq {}^{\beta}b_2 / {}^{\beta}a_2 \leq {}^{\alpha}b_2 / {}^{\alpha}a_2$

  - Solution

Suppose ${}^{\alpha}X = [{}^{\alpha}b_1 / {}^{\alpha}a_1, {}^{\alpha}b_2 / {}^{\alpha}a_2]$ is a solution of ${}^{\alpha}A \cdot {}^{\alpha}X = {}^{\alpha}B$ for any $\alpha \in (0,1]$. Then

$$X = \bigcup_{\alpha \in (0,1]} {}_{\alpha}X$$

Fuzzification

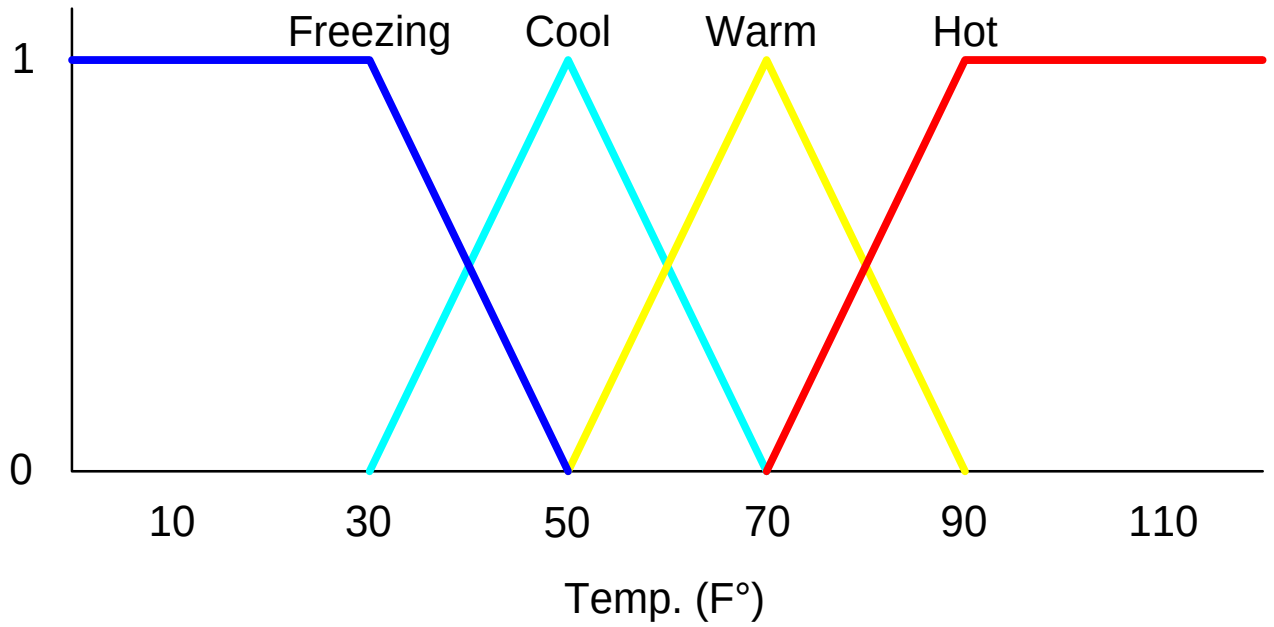
Fuzzification is the process of changing a real scalar value into a fuzzy value.

This is achieved with the different types of **fuzzifiers (membership functions).**

Fuzzification

Temp: {Freezing, Cool, Warm, Hot}

Degree of Truth or "Membership"

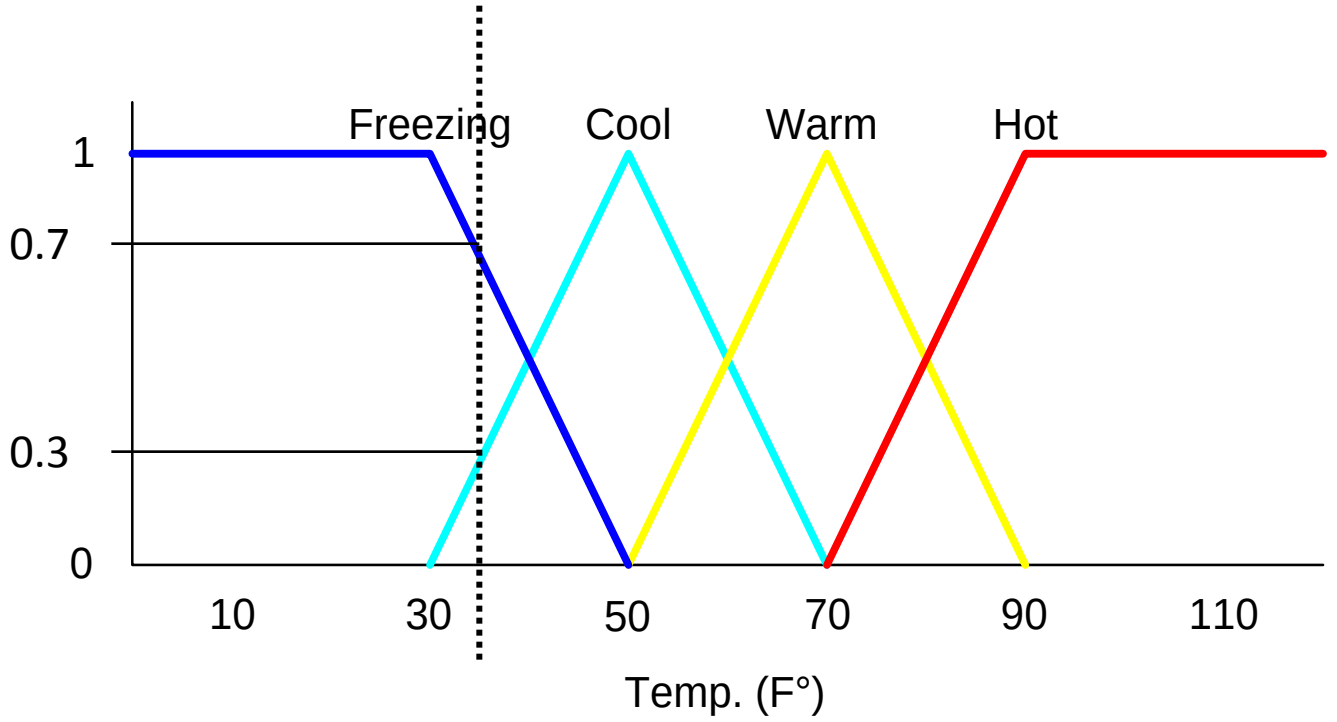


Membership Functions

Fuzzification

How cool is 36 F° ?

It is 30% Cool and 70% Freezing



Fuzzification

Membership Functions

The MATLAB toolbox includes 11 built-in membership function types. These 11 functions are, in turn, built from several basic functions:

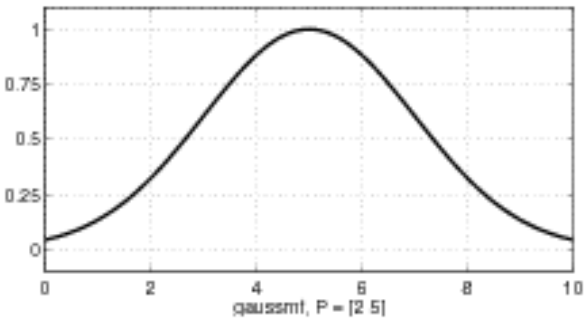
- piecewise linear functions
- the Gaussian distribution function
- the sigmoid curve
- quadratic and cubic polynomial curves

Fuzzification

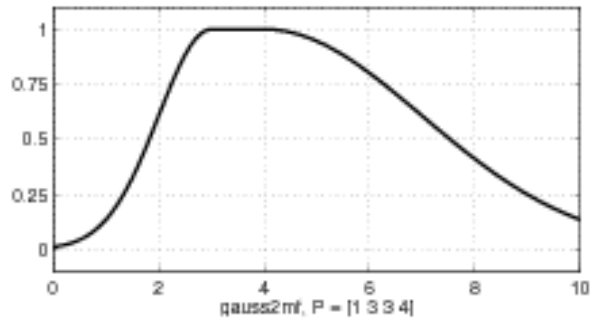
Membership Functions

Two membership functions are built on the Gaussian distribution curve: a simple Gaussian curve and a two-sided composite of two different Gaussian curves. The two functions are **gaussmf** and **gauss2mf**. The generalized bell membership has the function name **gbellmf**.

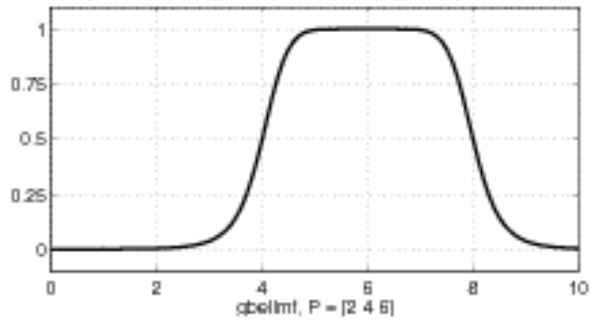
Because of their smoothness and concise notation, Gaussian and bell membership functions are popular methods for specifying fuzzy sets. Both of these curves have the advantage of being smooth and nonzero at all points.



gaussmf



gauss2mf



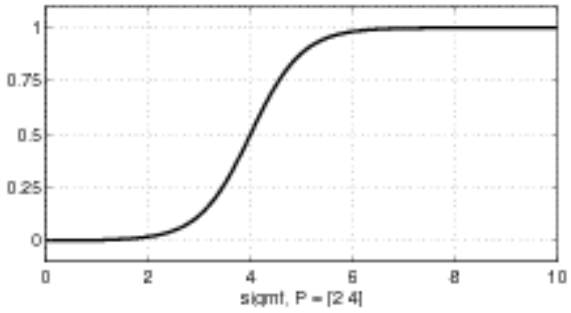
gbellmf

Fuzzification

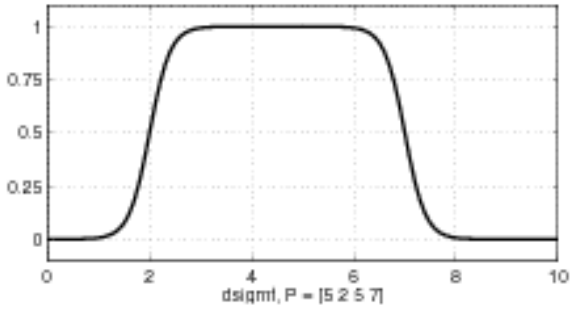
Membership Functions

Although the Gaussian membership functions and bell membership functions achieve smoothness, they are unable to specify asymmetric membership functions, which are important in certain applications.

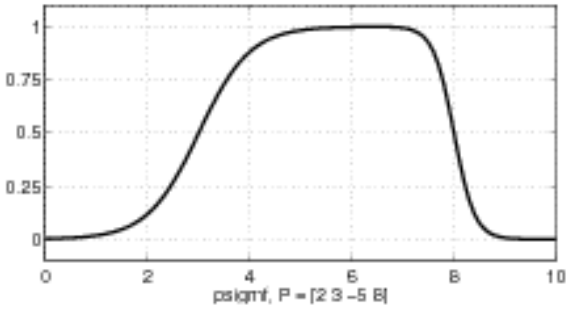
the **sigmoidal membership function** is defined, which is either open left or right. Asymmetric and closed (i.e. not open to the left or right) membership functions can be synthesized using two sigmoidal functions, so in addition to the basic **sigmf**, you also have the difference between two sigmoidal functions, **dsigmf**, and the product of two sigmoidal functions **psigmf**.



sigmf



dsigmf

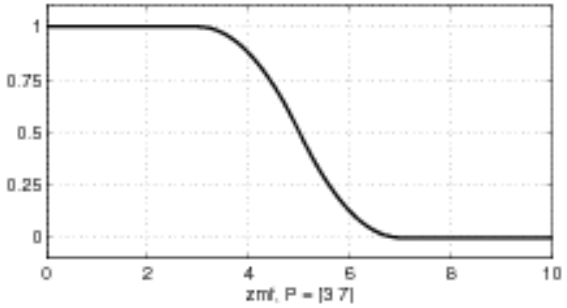


psigmf

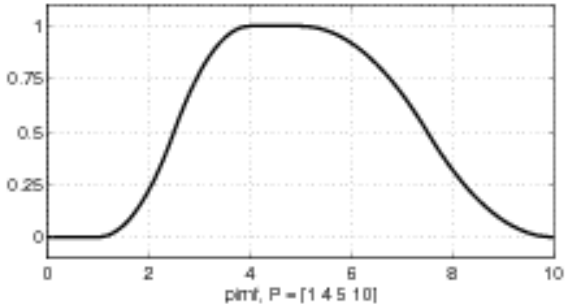
Fuzzification

Membership Functions

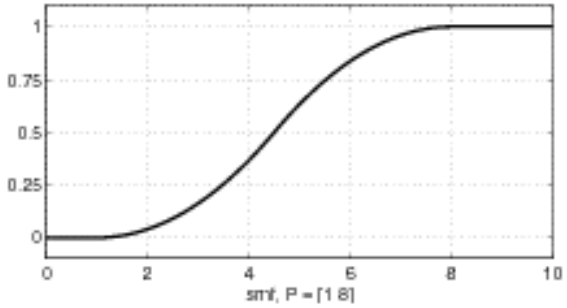
Polynomial based curves account for several of the membership functions in the toolbox.
Three related membership functions are the **Z, S, and Pi curves**, all named because of their shape. The function **zmf** is the asymmetrical polynomial curve open to the left, **smf** is the mirror-image function that opens to the right, and **pimf** is zero on both extremes with a rise in the middle.



zmf



pimf



smf

Unit -5

GENETIC ALGORITHRMS

Introduction

- After scientists became disillusioned with classical and neo-classical attempts at modeling intelligence, they looked in other directions.
- Two prominent fields arose, connectionism (neural networking, parallel processing) and evolutionary computing.
- It is the latter that this essay deals with - genetic algorithms and genetic programming.

What is GA

- A **genetic algorithm** (or **GA**) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- Genetic algorithms are categorized as global search heuristics.
- Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

What is GA

- Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions.

- Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

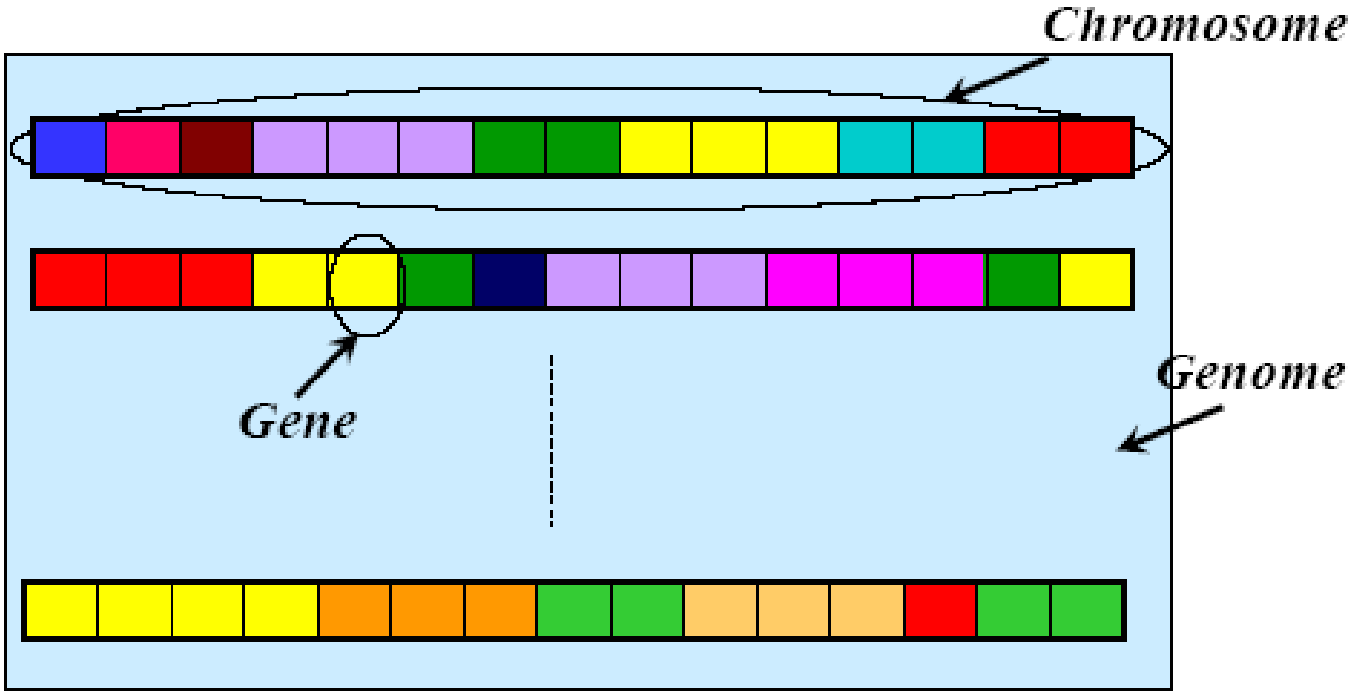
What is GA

- The new population is then used in the next iteration of the algorithm.
- Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.
- If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Key terms

- **Individual** - Any possible solution
- **Population** - Group of all *individuals*
- **Search Space** - All possible solutions to the problem
- **Chromosome** - Blueprint for an *individual*
- **Trait** - Possible aspect (*features)* of an *individual*
- **Allele** - Possible settings of trait (black, blond, etc.)
- **Locus** - The position of a *gene* on the *chromosome*
- **Genome** - Collection of all *chromosomes* for an *individual*

Chromosome, Genes and Genomes

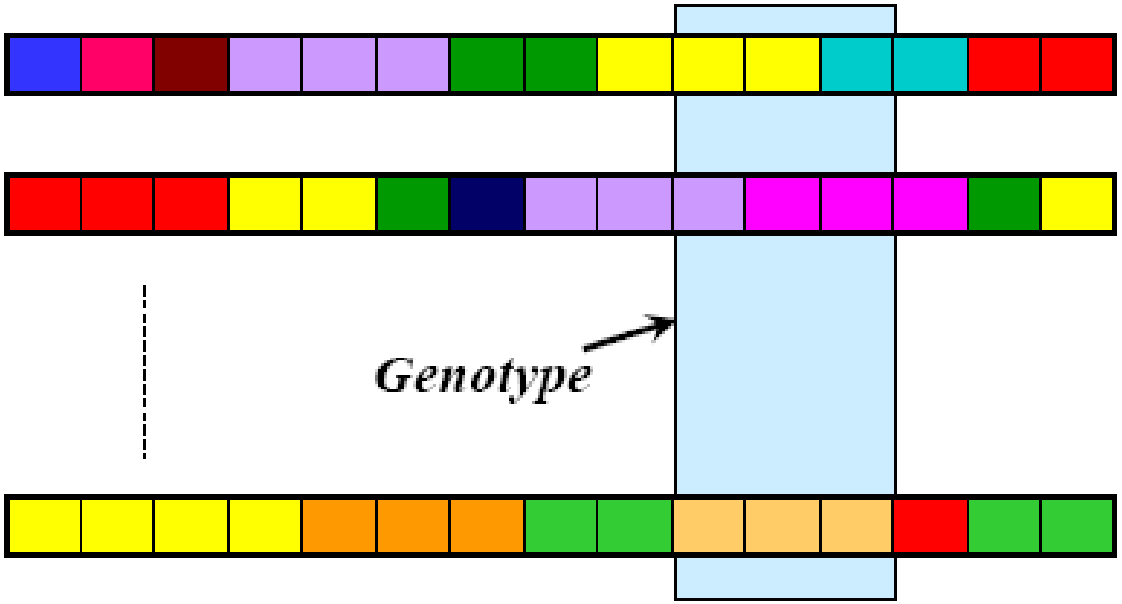


Genotype and Phenotype

- ***Genotype:***
  – Particular set of genes in a genome

- ***Phenotype:***
  – Physical characteristic of the genotype (smart, beautiful, healthy, etc.)

Genotype and Phenotype



GA Requirements

- A typical genetic algorithm requires two things to be defined:
- a genetic representation of the solution domain, and
- a fitness function to evaluate the solution domain.

- A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way.
- The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, that facilitates simple crossover operation.
- Variable length representations may also be used, but crossover implementation is more complex in this case.
- Tree-like representations are explored in Genetic programming.

Representation

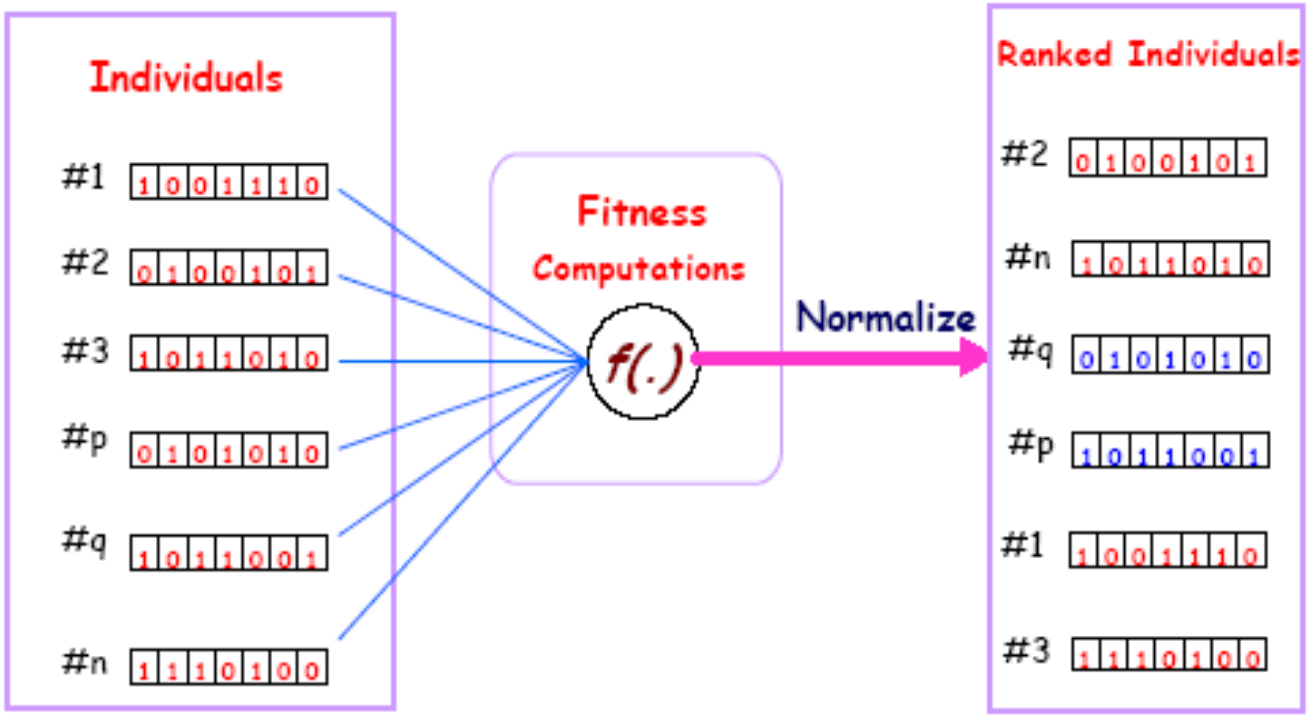
Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)

GA Requirements

- The fitness function is defined over the genetic representation and measures the *quality* of the represented solution.
- The fitness function is always problem dependent.
- For instance, in the knapsack problem we want to maximize the total value of objects that we can put in a knapsack of some fixed capacity.
- A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack.
- Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack.
- The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these

A fitness function



General Algorithm for GA

- **Initialization**
- Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions.
- Traditionally, the population is generated randomly, covering the entire range of possible solutions (the *search space*).
- Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

General Algorithm for GA

- **Selection**
- During each successive generation, a proportion of the existing population is selected to breed a new generation.
- Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected.
- Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.
- Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection

General Algorithm for GA

- In roulette wheel selection, individuals are given a probability of being selected that is directly proportionate to their fitness.

- Two individuals are then chosen randomly based on these probabilities and produce offspring.

General Algorithm for GA

- These processes ultimately result in the next generation population of chromosomes that is different from the initial generation.

- Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above

Evolving Neural Networks

- Evolving the architecture of neural network is slightly more complicated, and there have been several ways of doing it. For small nets, a simple matrix represents which neuron connects which, and then this matrix is, in turn, converted into the necessary 'genes', and various combinations of these are evolved.

Evolving Neural Networks

- Many would think that a learning function could be evolved via genetic programming. Unfortunately, genetic programming combined with neural networks could be *incredibly* slow, thus impractical.
- As with many problems, you have to constrain what you are attempting to create.
- For example, in 1990, David Chalmers attempted to evolve a function as good as the delta rule.
- He did this by creating a general equation based upon the delta rule with 8 unknowns, which the genetic algorithm then evolved.

Example

- $f(x) = \{MAX(x^2): 0 <= x <= 32 \}$
- Encode Solution: Just use 5 bits (1 or 0).
- Generate initial population.

A	0	1	1	0	1
B	1	1	0	0	0
C	0	1	0	0	0
D	1	0	0	1	1

- Evaluate each solution against objective.

Sol.	String	Fitness	% of Total
A	01101	169	14.4
B	11000	576	49.2
C	01000	64	5.5
D	10011	361	30.9

Example Cont'd

- Create next generation of solutions
  - Probability of "being a parent" depends on the fitness.
- Ways for parents to create next generation
  - Reproduction
    - Use a string again unmodified.
  - Crossover
    - Cut and paste portions of one string to another.
  - Mutation
    - Randomly flip a bit.
  - COMBINATION of all of the above.

THANKYOU