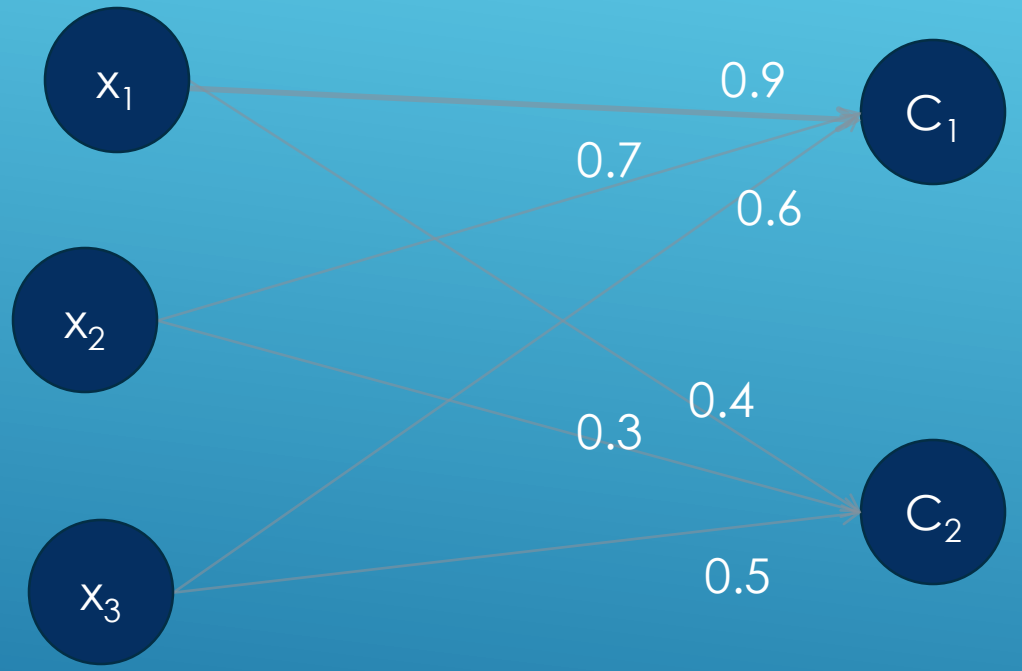
KOHONEN SOM (SELF ORGANIZING MAPS)

The weight vector for the cluster unit are (0.9,0.7,0.6) and (0.4,0.2,0.1). Find the winning cluster for the input vector (0.4,0.2,0.1). Use learning rate of 0.2. Find the new weight for the winning unit.



$$D(j) = \sum (w_{ij} - x_i)^2$$

$$D(1) = (0.9 - 0.4)^2 + (0.7 - 0.2)^2 + (0.6 - 0.1)^2$$

$$D(2) = (0.4 - 0.4)^2 + (0.3 - 0.2)^2 + (0.5 - 0.1)^2$$

$$w_{ij}(new) = w_{ij}(old) + \alpha * (x_i - w_{ij}(old))$$

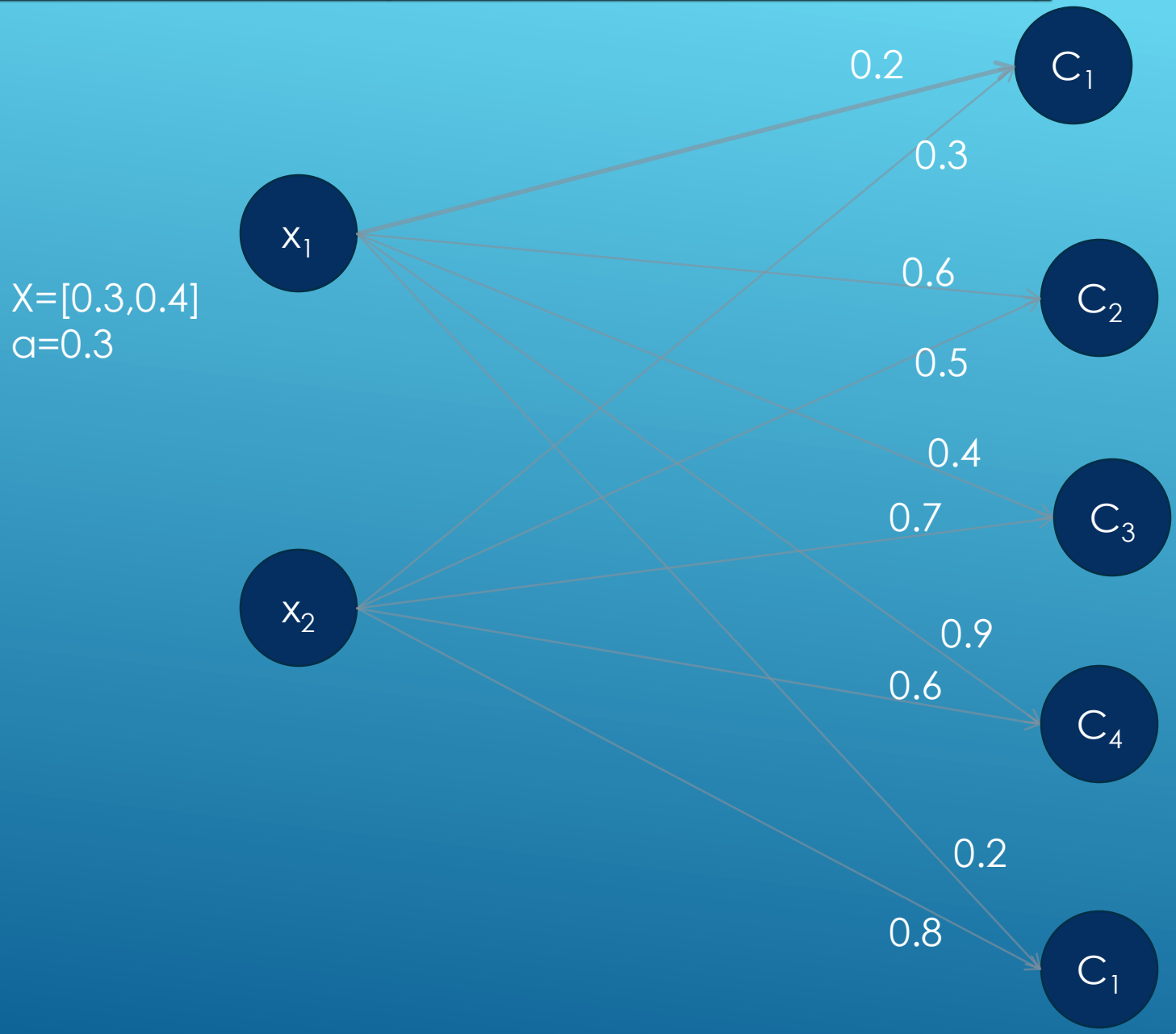
$$w_{12}(new) = 0.4 + 0.2 * (0.4 - 0.4)$$

$$w_{22}(new) = 0.3 + 0.2 * (0.2 - 0.3)$$

$$w_{32}(new) = 0.5 + 0.2 * (0.1 - 0.5)$$

$$w = \begin{bmatrix} 0.9 & 0.4 \\ 0.7 & 0.28 \\ 0.6 & 0.42 \end{bmatrix}$$

KOHONEN SOM (SELF ORGANIZING MAPS)



$$D(j) = \sum (w_{ij} - x_i)^2$$

$$D(1) = (0.2 - 0.3)^2 + (0.3 - 0.4)^2$$

$$D(2) = (0.6 - 0.3)^2 + (0.5 - 0.4)^2$$

$$D(3) = (0.4 - 0.3)^2 + (0.7 - 0.4)^2$$

$$D(4) = (0.9 - 0.3)^2 + (0.6 - 0.4)^2$$

$$D(5) = (0.2 - 0.3)^2 + (0.8 - 0.4)^2$$

$$w_{ij}(new) = w_{ij}(old) + \alpha * (x_i - w_{ij}(old))$$

$$w_{11}(new) = 0.2 + 0.3 * (0.3 - 0.2)$$

$$w_{21}(new) = 0.3 + 0.3 * (0.4 - 0.3)$$

$$w = \begin{bmatrix} 0.23 & 0.6 & 0.4 & 0.9 & 0.2 \\ 0.33 & 0.5 & 0.7 & 0.6 & 0.8 \end{bmatrix}$$

If R=1,
Weight updation=

$$w_{ij}(new) = w_{ij}(old) + \alpha * (x_i - w_{ij}(old))$$

$$w_{12}(new) = 0.6 + 0.3 * (0.3 - 0.6)$$

$$w_{22}(new) = 0.5 + 0.3 * (0.4 - 0.5)$$

$$w = \begin{bmatrix} 0.23 & 0.51 & 0.4 & 0.9 & 0.2 \\ 0.33 & 0.47 & 0.7 & 0.6 & 0.8 \end{bmatrix}$$

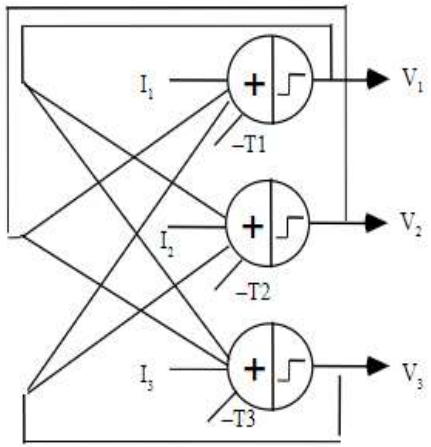
HOP FIELD NETWORK ALGORITHM

- Proposed by J.J. Hopfield. A fully Connected, feed-back, fixed weight network.
- Each neuron accepts its input from the outputs of all other neurons and the its own input:

Net function $u_i = \sum_{i=j} w_{ij} v_j + I_i$

$I_i = u_i / R_i$

Output: $v_i = \begin{cases} 1 & u_i \geq 0 \\ -1 & u_i < 0. \end{cases}$



Step 1: Initialize all weights to store pattern.
Step 2: For each input vector do step 3-7
Step 3: For each bipolar training vector and target output pair (s, t) perform steps 4-7
Step 4: Set activations for input units with input vector
Step 5: Compute net input to the output units

$$Y_{ini} = x_i + \sum_i Y_i * w_{ij}$$

$$Y_j = \begin{cases} 1\_if\_Y_{ini} > \theta \\ Y_i\_if\_Y_{ini} = \theta \\ 0\_if\_Z_{ini} < \theta \end{cases}$$

Step 6: Broadcat the value to all other units
Step 7: Test for convergence

Energy Function:

$$E = -0.5 \sum_{i \neq j} \sum_{j} Y_i Y_j w_{ij}$$

$$E = -0.5 \sum_{i \neq j} \sum_{j} x_i x_i^T w_{ij}$$

Hallucinations is one of the main problems in the **Discrete Hopfield Network**.
Sometimes network output can be something that we hasn't taught it.

Example 3: Hopfield neural network is trained by Hebb outer product rule for input row vector S=(1 1 -1 -1). Find the weight matrix.
S1=(1 1 -1 -1) t1=(1 1 -1 -1)

Step 1: Initialize all weights to zero
Step 2: Find the output for each input
S1=(1 1 -1 -1) t1=(1 1 -1 -1)

$$w_1 = S_1^t * t_1$$

$$= \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

Test the weight using different input set:

$$y_{inj} = \sum_{i=1}^{4} x_i * w_{ij}$$

$$y_{in1} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41}$$

$$y_{in2} = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42}$$

$$y_{in3} = x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43}$$

$$y_{in4} = x_1 w_{14} + x_2 w_{24} + x_3 w_{34} + x_4 w_{44}$$

S1=(1 1 0 0) t1=(1 0)

$$= \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix}$$

Design a Hopfield network for 4 bit bipolar patterns. The training patterns are:
S1=[1 1 -1 -1]
S2=[-1 1 -1 1]
S3=[-1 -1 -1 1]
Find the weight matrix and the energy for the inputs. Determine the pattern to which the sample T=[-1 1 -1 -1] associates.

$$w_1 = S_1^t * t_1$$

$$= \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

$$w_2 = S_2^t * t_2$$

$$= \begin{bmatrix} 1 & -1 & -1 \\ 1 & 1 & -1 \\ -1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$E = -0.5 \sum_{i \neq j} \sum_{j} Y_i Y_j w_{ij}$$

$$E = -0.5 \sum_{i \neq j} \sum_{j} x_i x_i^T w_{ij}$$

Design a Hopfield network for 4 bit bipolar patterns. The training patterns are:
S1=[1 1 1 1 1]
S2=[1 -1 -1 1 -1]
S3=[-1 1 -1 -1 -1]
Find the weight matrix and the energy for the inputs. Determine the pattern to which the sample T1=[1 1 1 -1 -1] , T2=[1 -1 -1 -1 -1] associates.

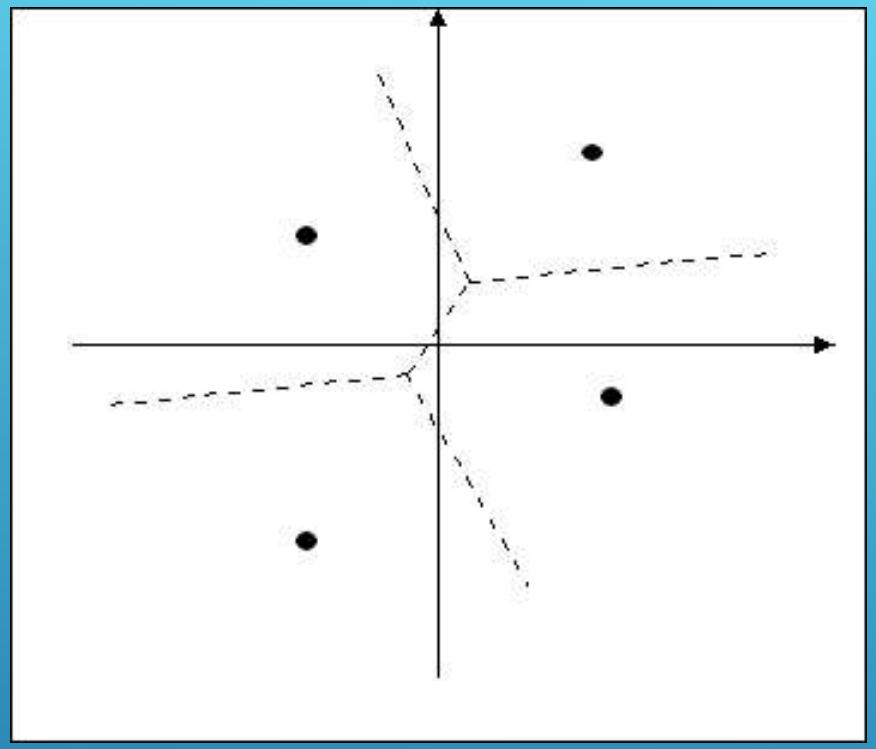
LVQ

- *Vector Quantisation* is a technique that exploits the underlying structure of input vectors for the purpose of data compression.
- An input space is divided in a number of distinct regions and for each region a reconstruction (representative) is defined.
- When the quantizer is presented with a new input vector, the region in which the vector lies is first determined, and is then represented by the reproduction vector for this region.
- The collection of all possible reproduction vectors is called the *code book* of the quantizer and its members are called *code words*.

LVQ-1

- A vector quantizer with minimum encoding *distortion* is called *Voronoi* or *nearest-neighbour quantizer*, since the Voronoi cells about a set of points in an input space correspond to a partition of that space according to the nearest-neighbour rule based on the Euclidean metric.
- An example with an input space divided to four cells and their associated Voronoi vectors is shown below:

LVQ-2

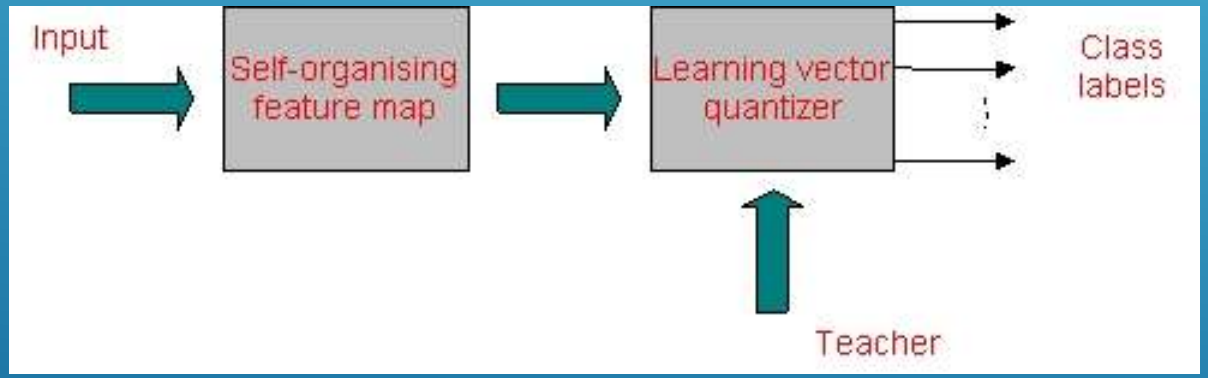


- The SOM algorithm provides an approximate method for computing the Voronoi vectors in an unsupervised manner, with the approximation being specified by the

LVQ-3

weight vectors of the neurons in the feature map.

- Computation of the feature map can be viewed as the first of two stages for adaptively solving a pattern classification problem as shown below. The second stage is provided by the learning vector quantization, which provides a method for fine tuning of a feature map.



LVQ-4

- *Learning vector quantization* (LVQ) is a supervised learning technique that uses class information to move the Voronoi vectors slightly, so as to improve the quality of the classifier decision regions.
- An input vector **x** is picked at random from the input space. If the class labels of the input vector and a Voronoi vector **w** agree, the Voronoi vector is moved in the direction of the input vector **x**. If, on the other hand, the class labels of the input vector and the Voronoi vector disagree, the Voronoi vector **w** is moved away from the input vector **x**.
- Let us denote $\{\mathbf{w}_j\}_{j=1}^{l}$ the set of Voronoi vectors, and let $\{\mathbf{x}_i\}_{i=1}^{N}$ be the set of input vectors. We assume that

N >> l.

- The LVQ algorithm proceeds as follows:

  i. Suppose that the Voronoi vector $\mathbf{w}_c$ is the closest to the input vector $\mathbf{x}_i$. Let C_{wc} and C_{xi} denote the class labels associated with $\mathbf{w}_c$ and $\mathbf{x}_i$ respectively. Then the Voronoi vector $\mathbf{w}_c$ is adjusted as follows:

  - If $C_{wc} = C_{xi}$ then

  $$\mathbf{W}_c(n+1)= \mathbf{w}_c(n)+a_n[\mathbf{x}_i- \mathbf{w}_c(n)]$$

  Where $0< a_n <1$

- If $C_{wc} \neq C_{xi}$ then

$$\mathbf{W}_c(n+1)= \mathbf{w}_c(n)-a_n[\mathbf{x}_i- \mathbf{w}_c(n)]$$

ii. The other Voronoi vectors are not modified.

- It is desirable for the learning constant a_n to decrease monotonically with time n. For example a_n could be initially 0.1 and decrease linearly with n.
- After several passes through the input data the Voronoi vectors typically converge at which point the training is complete.

Step 1: Initialize weights (reference) vectors.
Initialize learning rate.

Step 2 : While stopping is false, do Steps 3–7

Step 3: For each training input vector x, do Steps 4–5

Step 4 : Compute J using squared Euclidean distance.

$$D(j) = \sum (w_{ij} - x_i)^2$$

Find j when $D(j)$ is minimum

Step 5 : Update w_J as follows:

If $t = c_J$, then

$$w_{J(new)} = w_{J(old)} + \alpha[x - w_J\,(old)]$$

If $t \neq c_J$, then

$$w_{J(new)} = w_{J(old)} - \alpha[x - w_J\,(old)]$$

Step 6: Reduce the learning rate.

Step 7: Test for the stopping condition.

Construct and test LVQ with four vectors assigned to two classes.
Assume alpha =0.1, Perform interaction upto alpha=0.05

Vector	Class
1010	1
0011	2
1100	1
1001	2

1) Initialize weight =1010 and w2=0011
2) Begin training
3) X=1100 with T=1
4) Calculate J

$$D(j) = \sum (w_{ij} - x_i)^2$$

$$D(1) = (1-1)^2 + (0-1)^2 + (1-0)^2 + (0-0)^2$$

$$D(2) = (0-1)^2 + (0-1)^2 + (1-0)^2 + (1-0)^2$$

$$w_{ij}(new) = w_{ij}(old) + \alpha * (x_i - w_{ij}(old))$$

$$w_1(new) = [1010] + 0.1[[1100] - [1010]]$$

$$w = \begin{bmatrix} 1 & 0.11 & 0.99 & -0.1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$W =$$

- Randomly choose the initial weights
- While error is too large
  - For each training pattern (presented in random order)
    - Apply the inputs to the network
    - Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer
    - Calculate the error at the outputs
    - Use the output error to compute error signals for pre-output layers
    - Use the error signals to compute weight adjustments
    - Apply the weight adjustments
  - Periodically evaluate the network performance

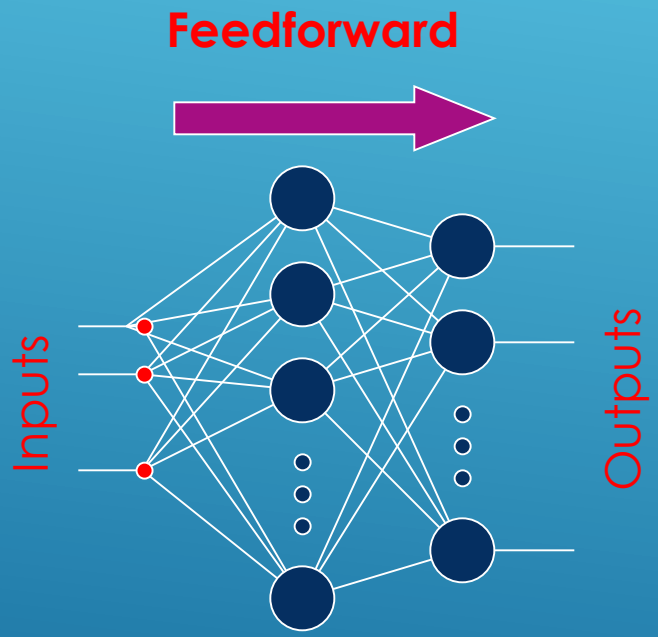
A PSEUDO-CODE ALGORITHM

- Two-dimensional arrays
  - Weights (at least for input-to-hidden layer and hidden-to-output layer connections)
  - Weight changes (Δ_{ij})
- One-dimensional arrays
  - Neuron layers
    - Cumulative current input
    - Current output
    - Error signal for each neuron
  - Bias weights

POSSIBLE DATA STRUCTURES

APPLY INPUTS FROM A PATTERN

- Apply the value of each input parameter to each input node
- Input nodes compute only the identity function



CALCULATE OUTPUTS FOR EACH NEURON BASED ON THE PATTERN

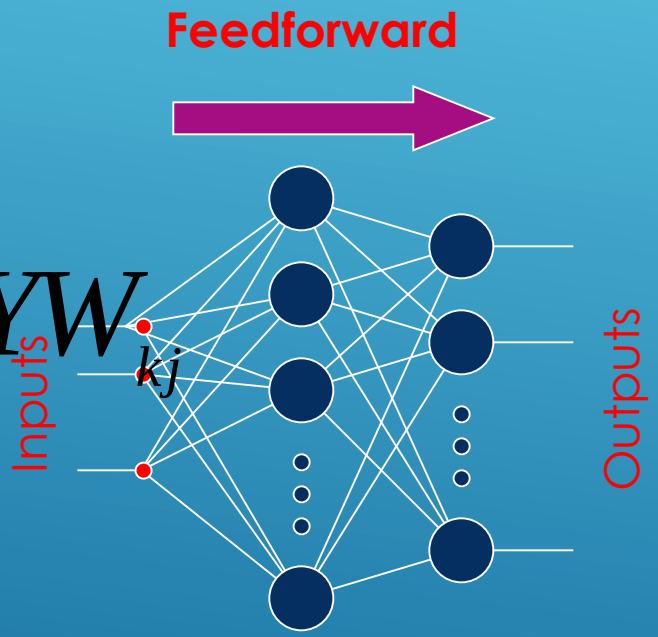
- The output from neuron j for pattern p is O_{pj} where

$$Y(Y_{inj}) = \frac{1}{1 + e^{-\alpha Y_{inj}}}$$

and

$$net_j = bias * W_{bias} + \sum_k YW_{kj}$$

k ranges over the input indices and W_{jk} is the weight on the connection from input k to neuron j



- The output neuron error signal δ_{pj} is given by $\delta_{pj}=(T_{pj}-O_{pj})\ O_{pj}\ (1-O_{pj})$
- T_{pj} is the target value of output neuron j for pattern p
- O_{pj} is the actual output value of output neuron j for pattern p

CALCULATE THE ERROR SIGNAL FOR EACH OUTPUT NEURON

CALCULATE THE ERROR SIGNAL FOR EACH HIDDEN NEURON

- The hidden neuron error signal δ_{pj} is given by

$$\delta_{pj} = O_{pj}(1-O_{pj})\sum_{k}\delta_{pk}W_{kj}$$

where δ_{pk} is the error signal of a post-synaptic neuron k and W_{kj} is the weight of the connection from hidden neuron j to the post-synaptic neuron k

CALCULATE AND APPLY WEIGHT ADJUSTMENTS

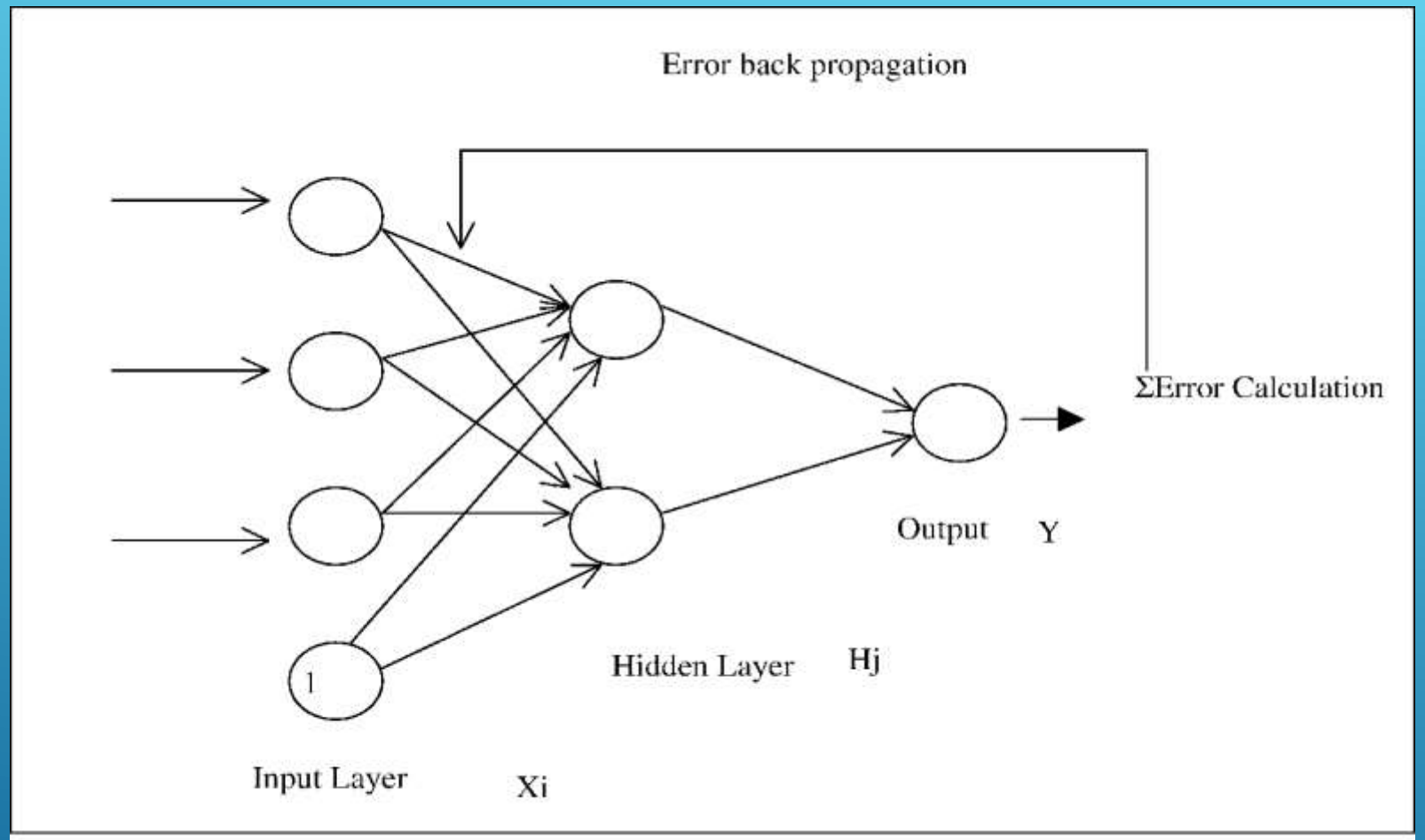
- Compute weight adjustments ΔW_{ji} at time t by

  $\Delta W_{ji}(t) = \eta\ \delta_{pj}\ O_{pi}$

- Apply weight adjustments according to

  $W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji}(t)$

- *Some add a momentum term* $\alpha * \Delta W_{ji}(t-1)$



Notes: The weight connecting node *i* in the input layer to node *j* in the hidden layer is denoted by *Wji*, and the weight connecting node *j* to the output node is represented by *Vj*

FUZZY LOGIC

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

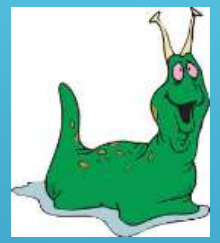
Speed = 0

Fast

Speed = 1

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

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.

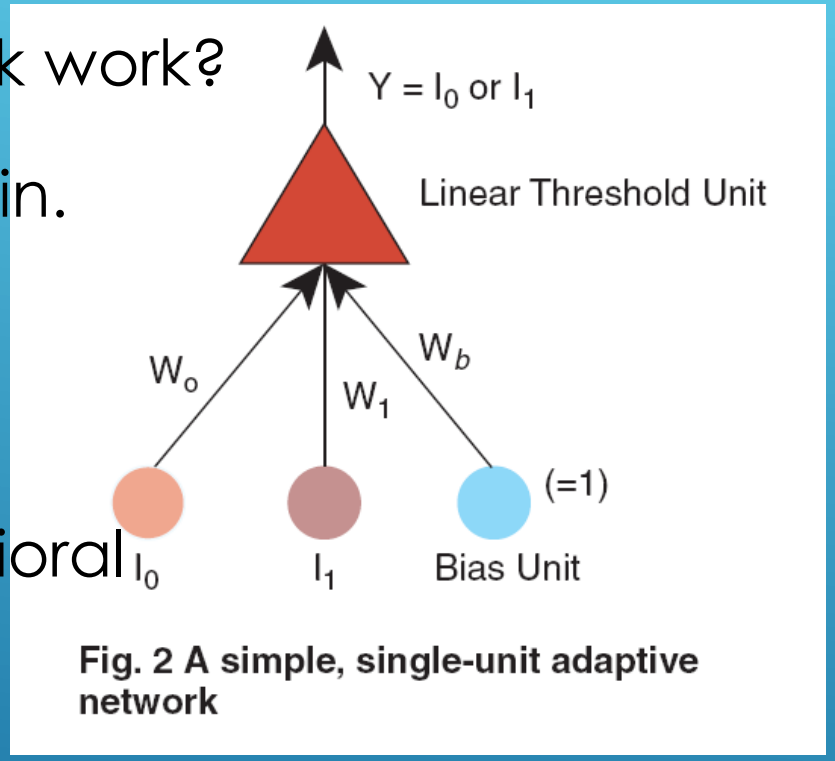
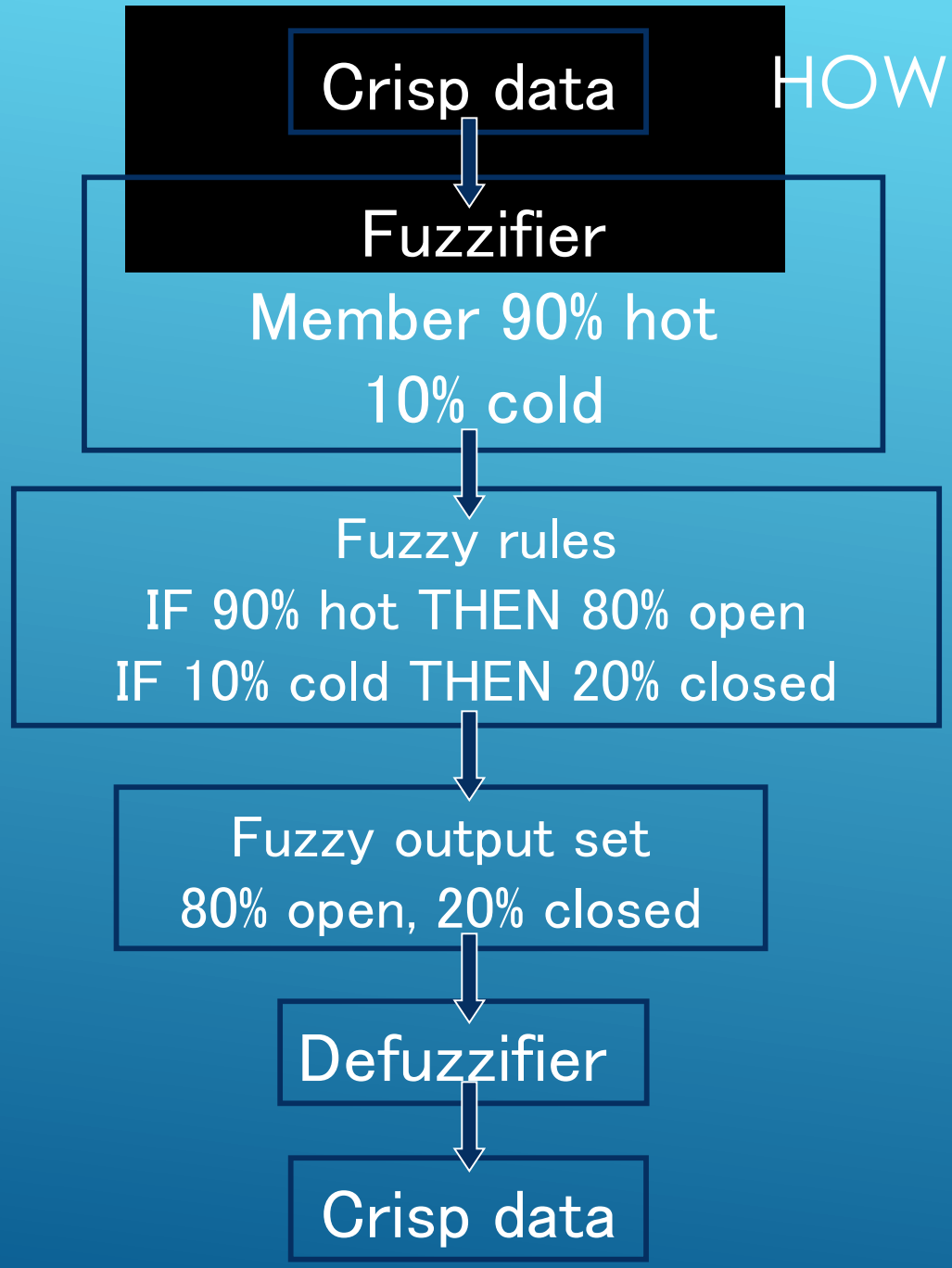


Fig. 2 A simple, single-unit adaptive network

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)

HOW THE MODELS WORK



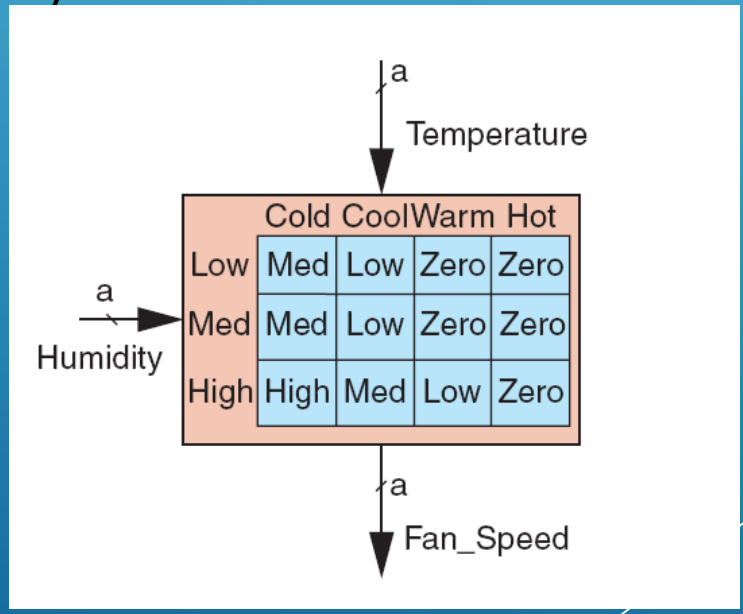
Inputs converted to degrees of membership of fuzzy sets.

Fuzzy rules applied to get new sets of members.

These sets are then converted back to real numbers.

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.



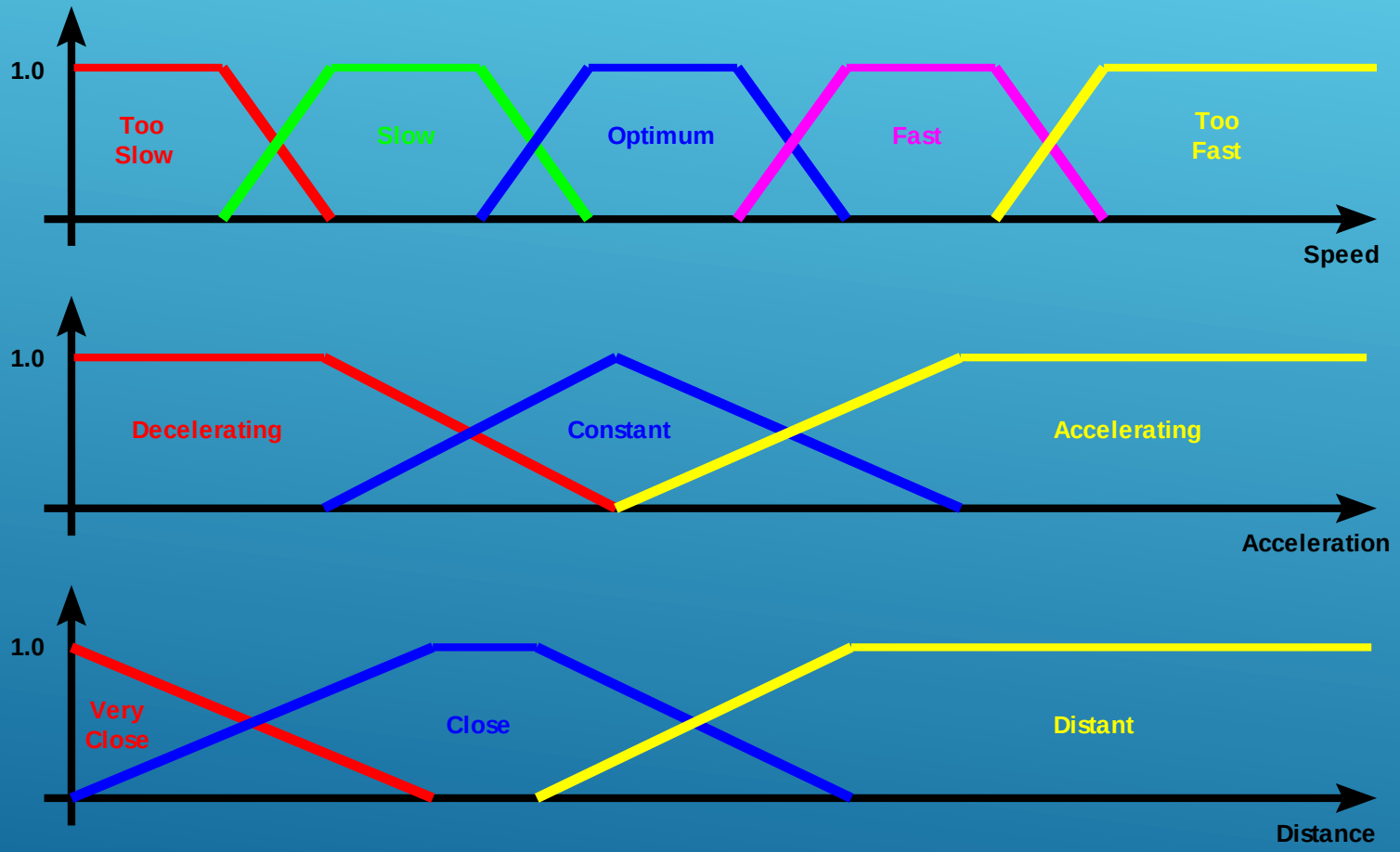
Steps

Fuzzification: determines an input's % membership in overlapping sets.

Rules: determine outputs based on inputs and rules.

Combination/Defuzzification: combine all fuzzy actions into a single fuzzy action and transform the single fuzzy action into a crisp, executable system output. May use centroid of weighted sets.

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

. . .

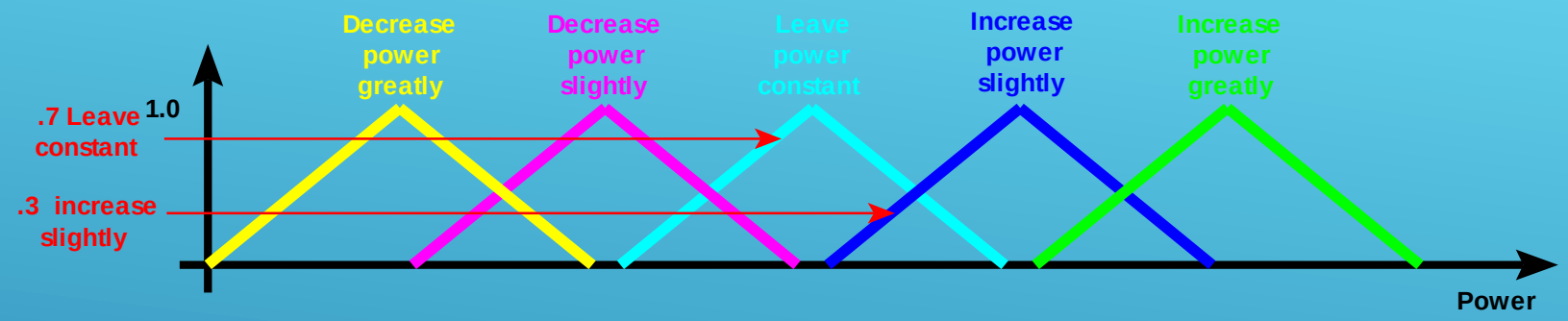
Output Determination

Degree of membership in an output fuzzy set now represents each fuzzy action.

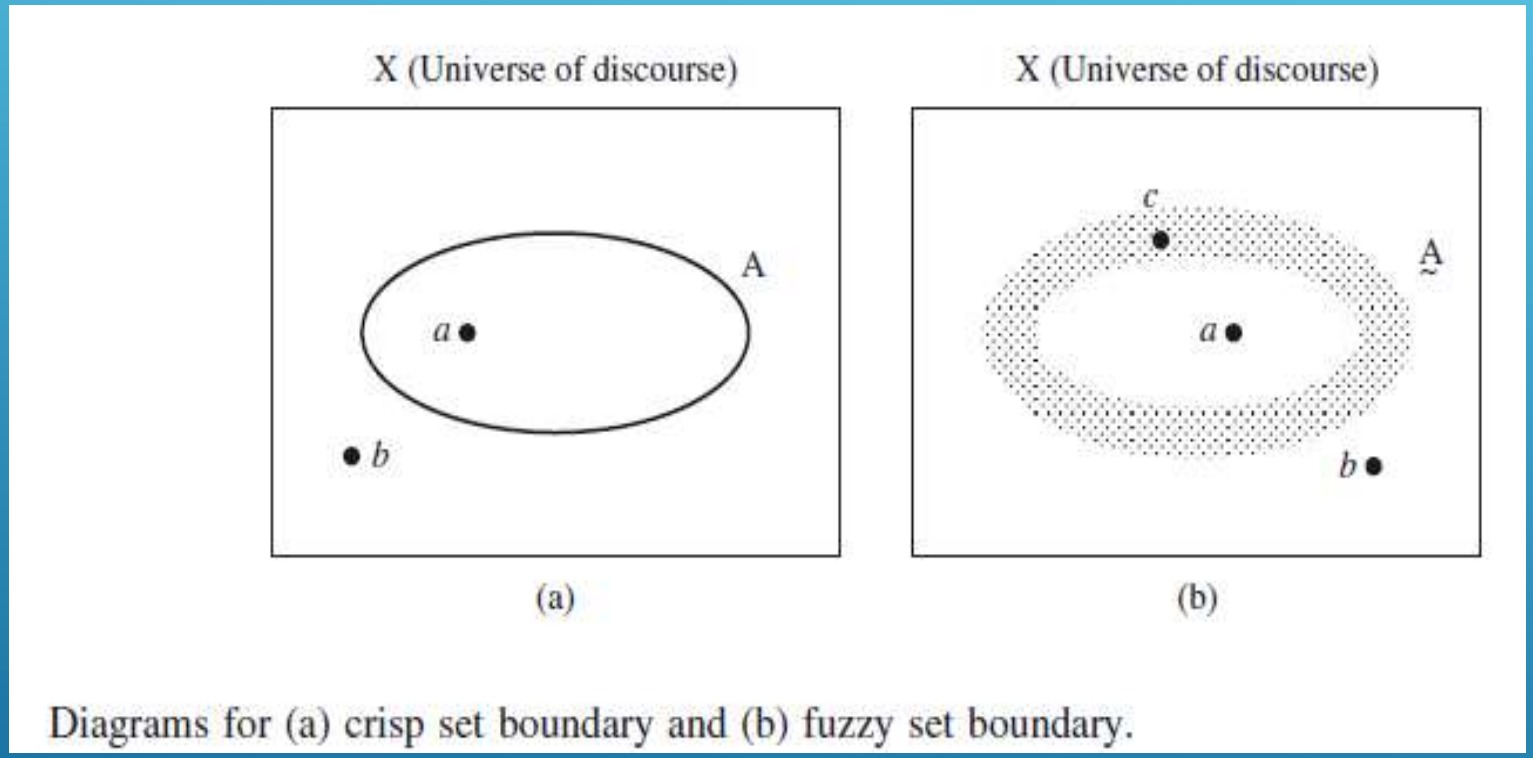
Fuzzy actions are combined to form a system output.

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)


Decrease power greatly
Decrease power slightly
Leave power constant
Increase power slightly
Increase power greatly
1.0
.7 Leave constant
.3 increase slightly
Power

Crisp Set and Fuzzy Set



Diagrams for (a) crisp set boundary and (b) fuzzy set boundary.

Information World

Crisp set has a unique membership function

$$\mu_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$$

$$\mu_A(x) \in \{0, 1\}$$

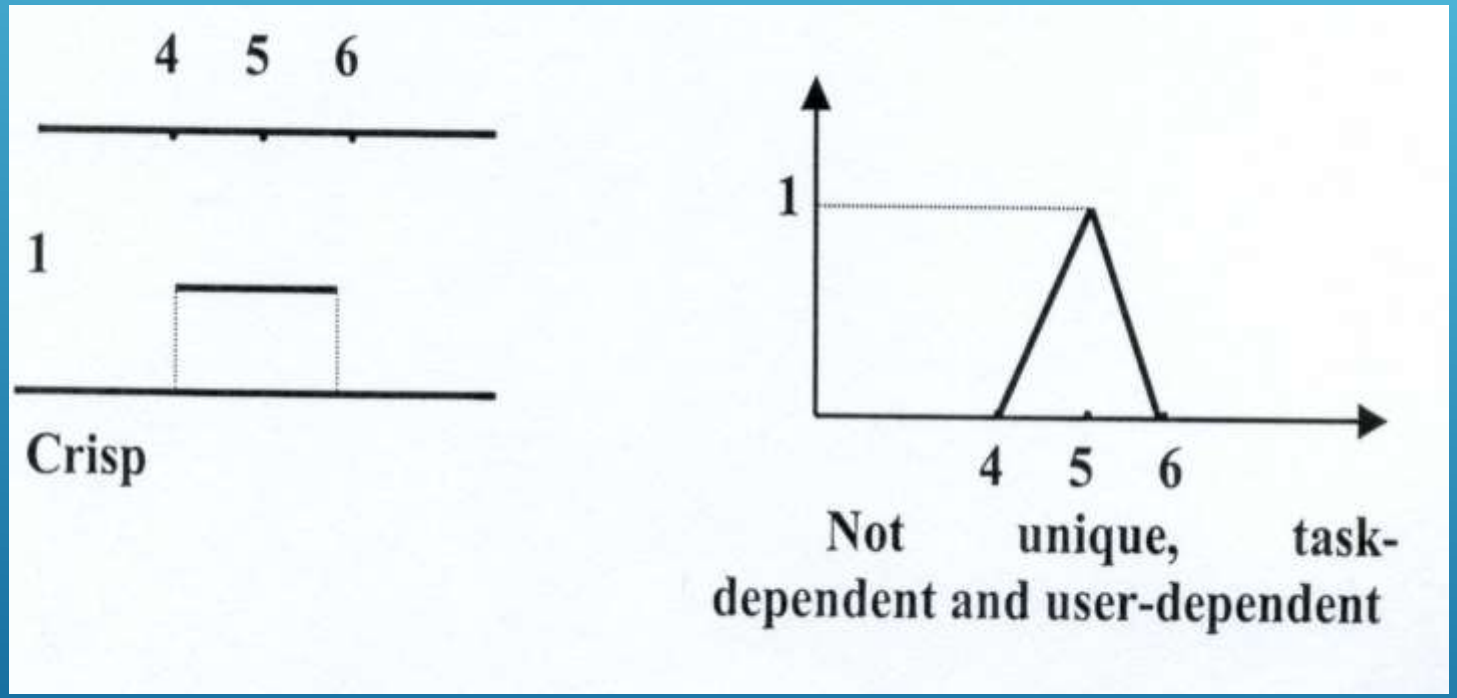
Fuzzy Set can have an infinite number of membership functions

$$\mu_A \in [0,1]$$

Fuzziness

Examples:

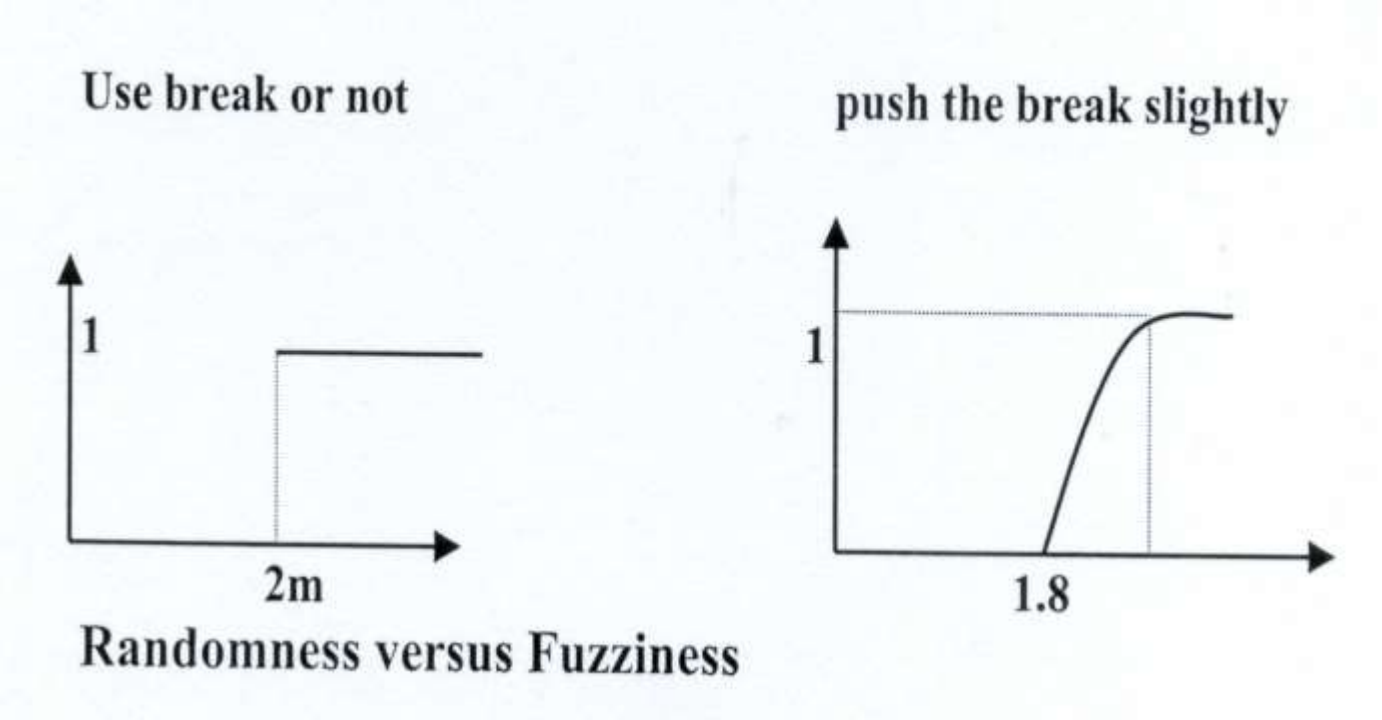
A number is close to 5



Fuzziness

Examples:

He/she is tall



Randomness versus Fuzziness

Classical Sets

CLASSICAL SETS

Define a universe of discourse, X, as a collection of objects all having the same characteristics. The individual elements in the universe X will be denoted as *x*. The features of the elements in X can be discrete, or continuous valued quantities on the real line. Examples of elements of various universes might be as follows:

- the clock speeds of computer CPUs;
- the operating currents of an electronic motor;
- the operating temperature of a heat pump;
- the integers 1 to 10.

Operations on Classical Sets

Union:

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

Intersection:

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$$

Complement:

$$A' = \{x \mid x \notin A, x \in X\}$$

X – Universal Set

Set Difference:

$$A \mid B = \{x \mid x \in A \text{ and } x \notin B\}$$

Set difference is also denoted by A - B