Artificial Intelligence

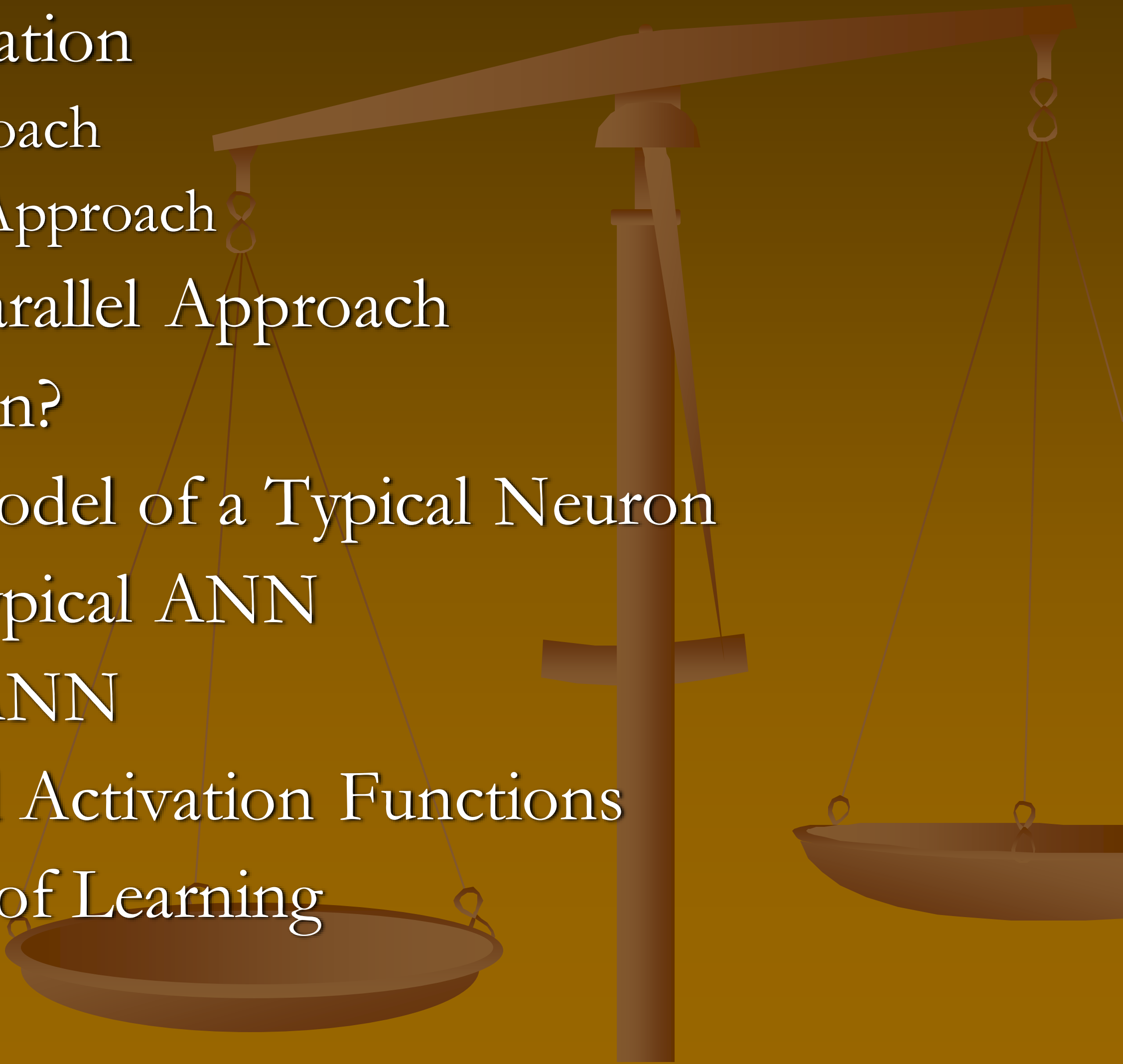
Classification

Artificial Neural Network - I Lecture

Department of Computer Science
National University of Computer & Emerging Sciences
Lahore.

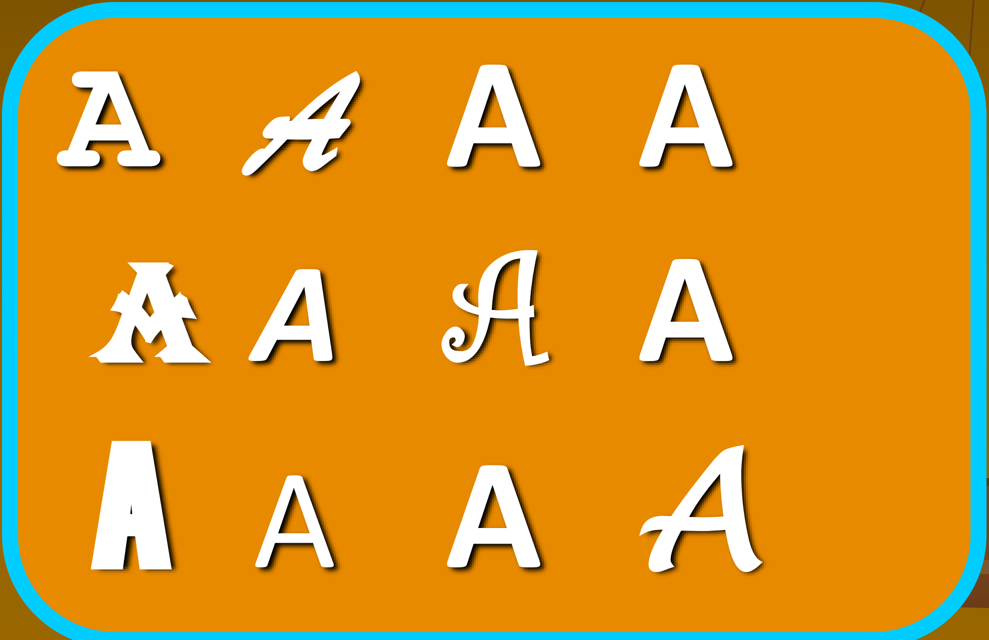
Today's Topics Are …

- Artificial Neural Networks (ANNs)
  - Introduction
  - Pattern Classification
    - Database Approach
    - An Intelligent Approach
  - Sequential vs. Parallel Approach
  - What is a Neuron?
  - Mathematical Model of a Typical Neuron
  - Working of a Typical ANN
  - Training of an ANN
  - Commonly used Activation Functions
  - The Hebb Rule of Learning



Artificial Neural Networks

- A neural network is a computational paradigm inspired by the parallelism of the brain.
- There are certain things that humans do well (e.g. vision and language), while there are certain things which present day computers do well (e.g. numeric computations).
- Take a look at the letters in the following figure:



- We can recognize all of these variations of the letter "A".
- Can we write down a computer program to recognize all of these, and all other possible variations of the letter 'A'?

Artificial Neural Networks

Pattern Classification – Database Approach

- One possible approach to writing the program is to store all of the possible patterns in a database (A Database Management Person's Approach!), and to do a bit by bit comparison of the letter in question with all of the stored templates.
- If the pattern matches one of the templates to within a certain degree of error, we can conclude the letter is in fact an "A".

Problems

- The number of templates that you would have to store would be large, especially if you wanted to recognize all letters, both upper and lower case, and all numerals.
- There could be variations in size and in angle of rotation, as well in the style of the letter.
- How about handwritten characters (the number of possible templates could explode astronomically!)?

Artificial Neural Networks

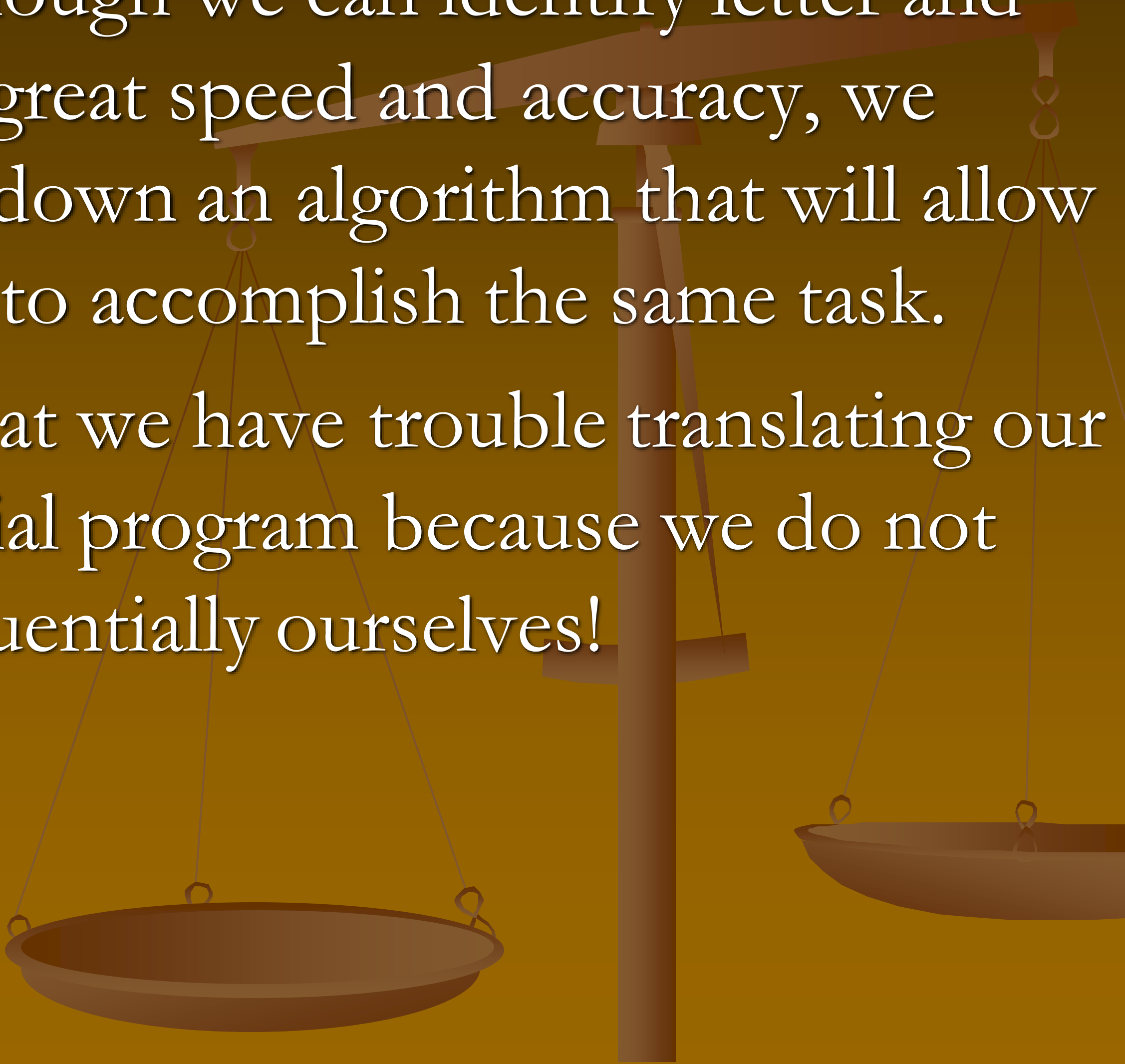
Pattern Classification – An Intelligent Solution

- The situation may not be hopeless, however, since we can add a bit of intelligence to our program.
- For example, we can study all of the ways that the letter "A" is formed, and attempt to find out commonalities – features, as they are called – that distinguish one letter from another.
- We can then program an intelligent system to search for those features and match them against the known list for the various letters.
- While this approach might work for some letters, but variations in writing style and typography would still necessitate a huge information base and would likely not account for all possibilities.

Artificial Neural Networks

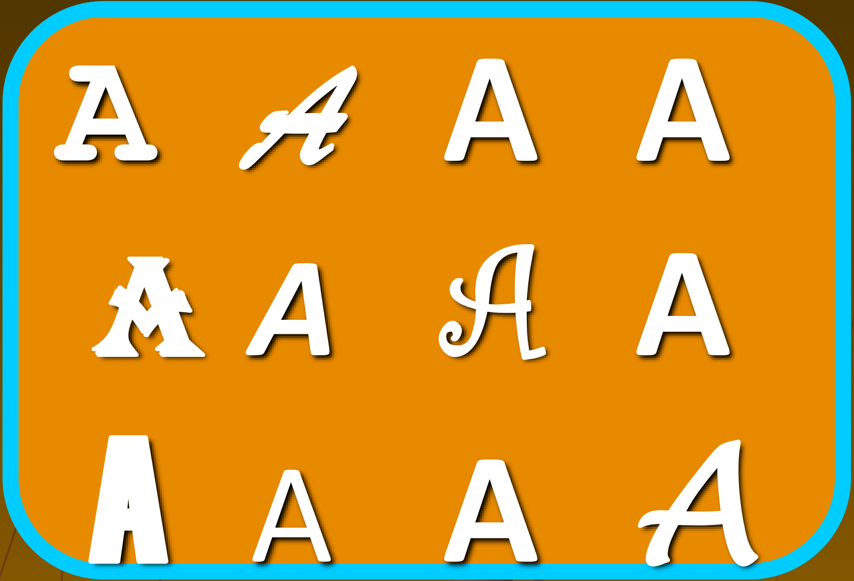
Sequential vs. Parallel

- It is curious that, although we can identify letter and objects visually with great speed and accuracy, we have trouble writing down an algorithm that will allow a computer program to accomplish the same task.
- Perhaps, it may be that we have trouble translating our ability into a sequential program because we do not perform the task sequentially ourselves!



Artificial Neural Networks

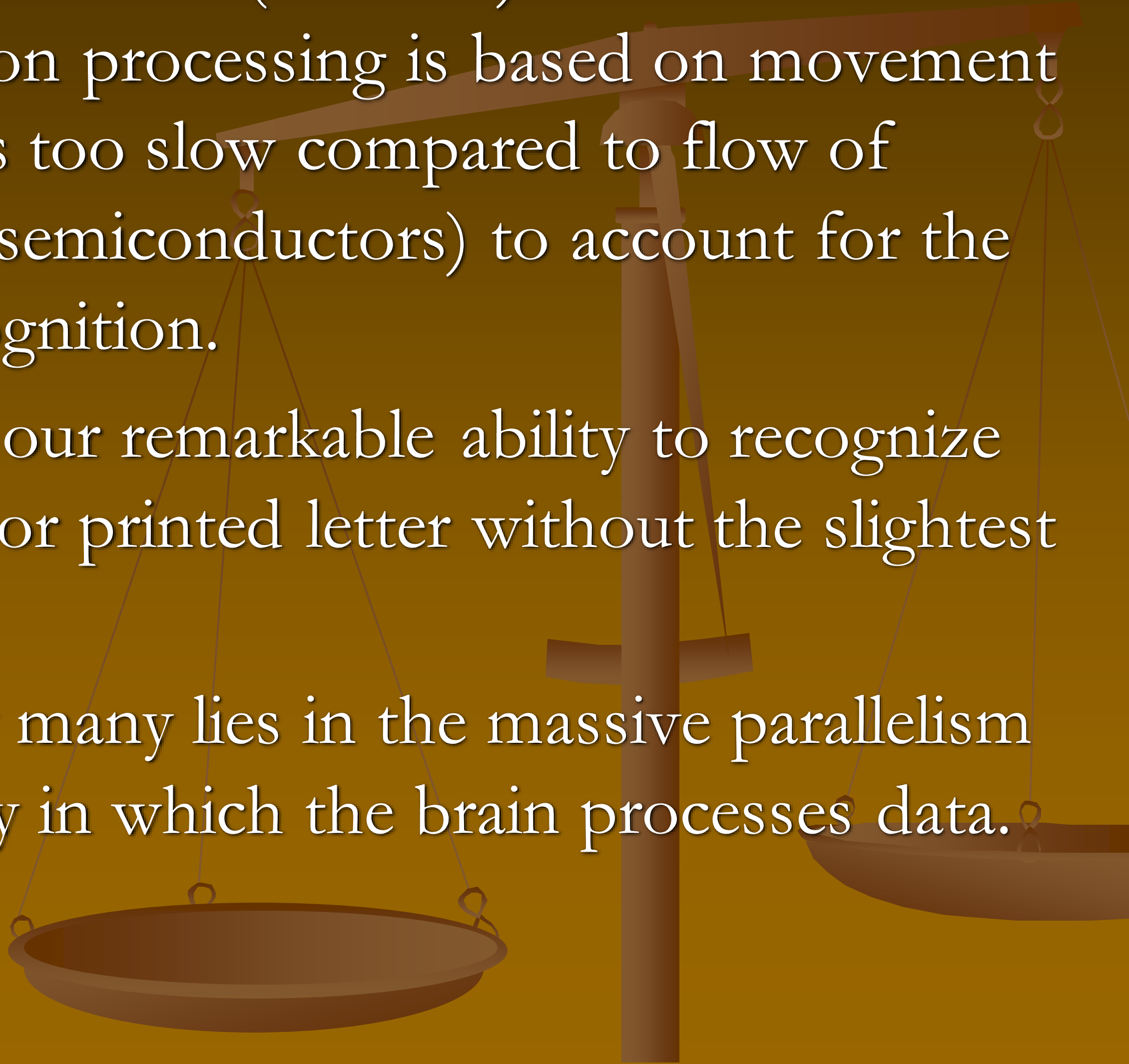
Sequential vs. Parallel (con…)



- Lets look at the letters again.
- Think about the process you went through to identify the letters.
- Did you scan the letter from left to right and top to bottom?
- Did you assemble information about the angles of lines and the intersection between line segments?
- Did you pick apart the letter's features and "logically" conclude that, on the basis of the information, the letter must be an "A"?
- Probably not, at least not consciously, and certainly not in a long sequence of individual steps that treat each feature or pixel sequentially

Artificial Neural Networks

Sequential vs. Parallel (con…)



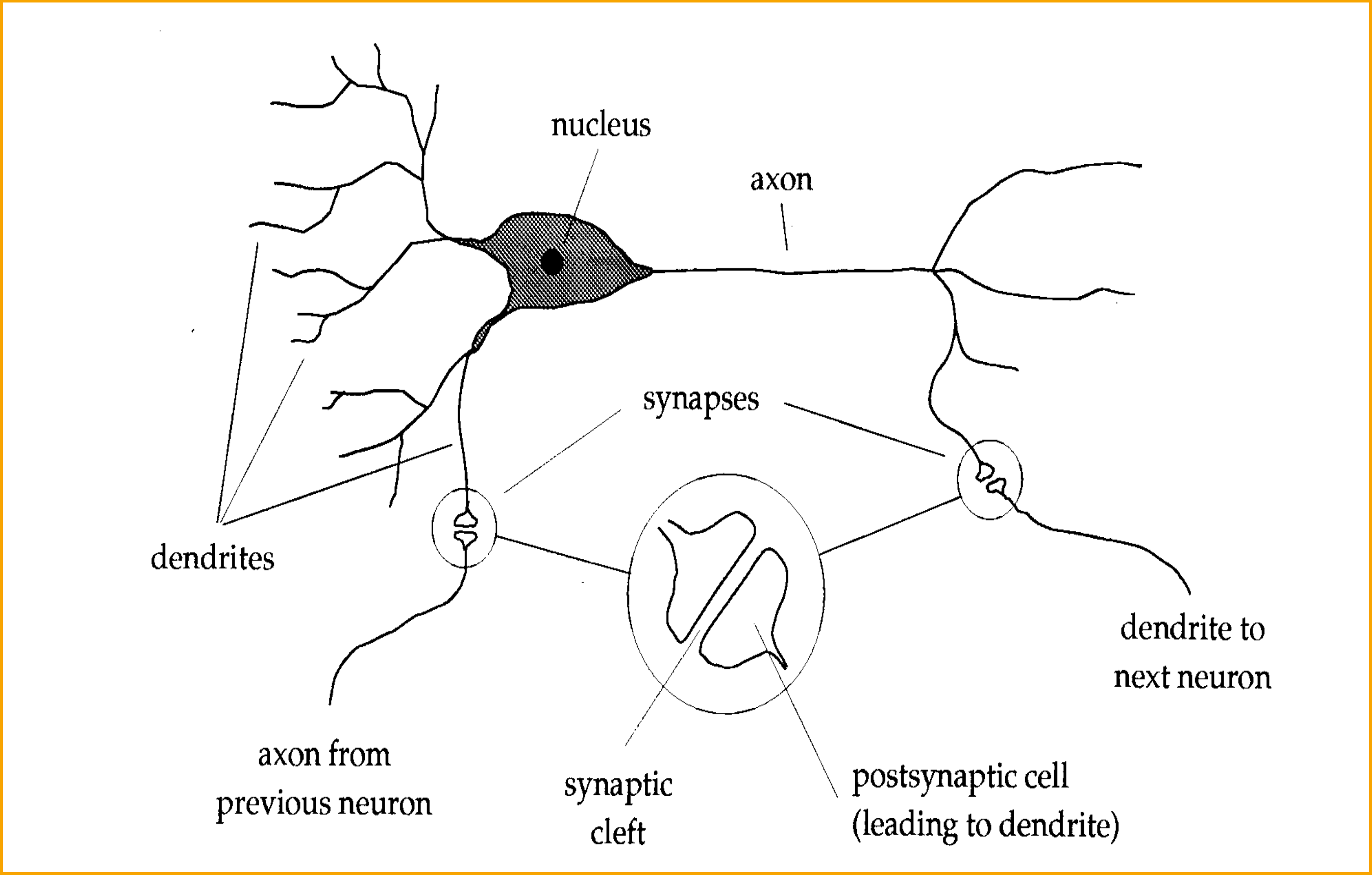
- The brain's processing elements (neurons) are much too slow (because their information processing is based on movement of ions in fluids which is too slow compared to flow of electrons or photons in semiconductors) to account for the most instantaneous recognition.
- What then accounts for our remarkable ability to recognize almost any handwritten or printed letter without the slightest effort?
- The answer, believed by many lies in the massive parallelism of the brain, and the way in which the brain processes data.

Artificial Neural Networks

Sequential vs. Parallel (con…)

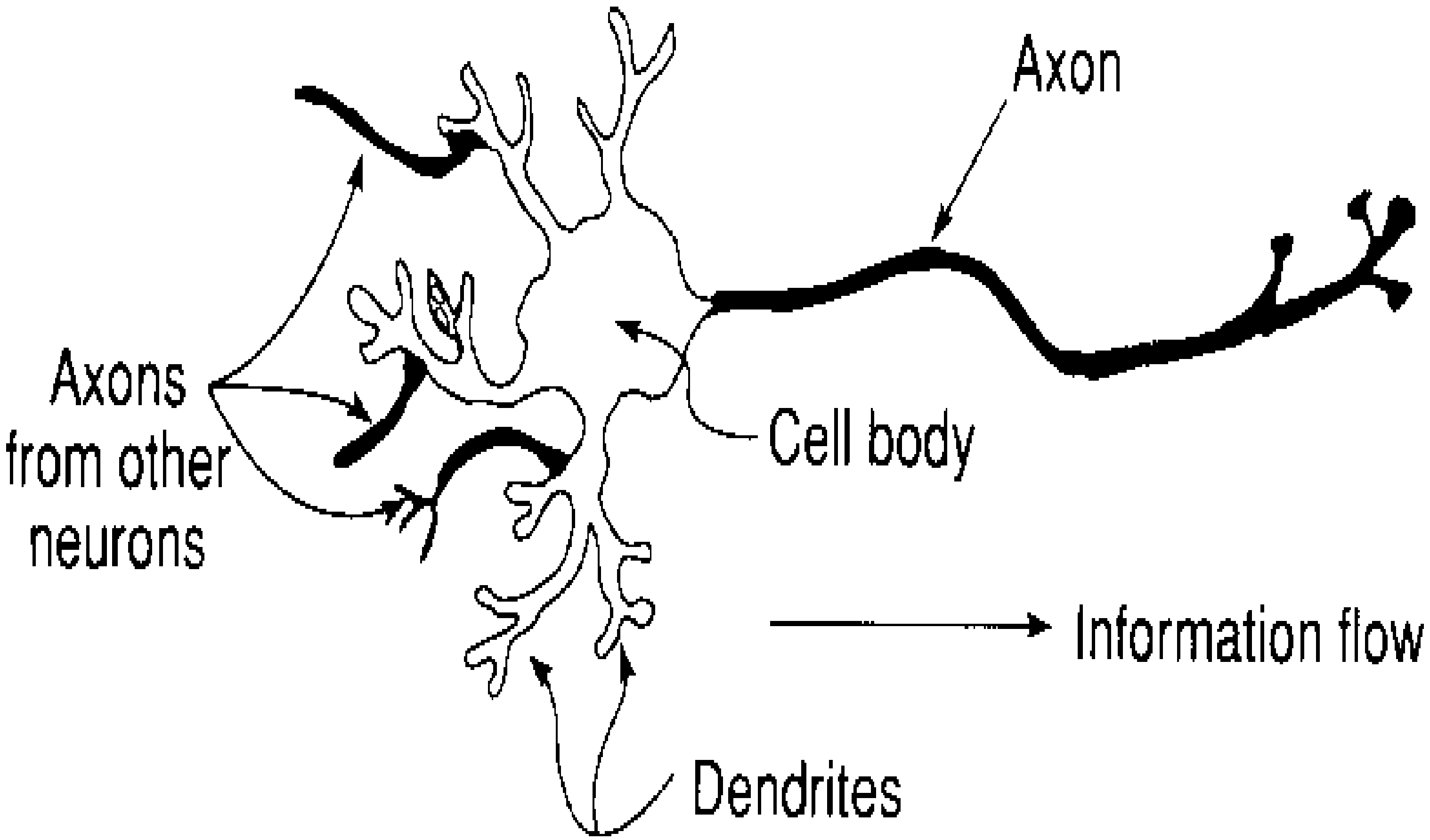
- When we look at one of the letters, information from every point reaches our retina simultaneously.
- That information is transformed into electrical impulses that travel along nerves to locations deep within our brain.
- Features are extracted from various parts of the image and information from one part of the image is combined with information from other parts, allowing the brain to rapidly from a hypothesis about the identity of the letter.
- It is likely that some of this processing is sequential.
- However, much of this work is done in parallel, with the brain operating on the whole image at once, sharing information among neurons about features in all parts of the image.
- This massive parallelism gives the brain its ability to process data so quickly and efficiently.

What Is Neuron?

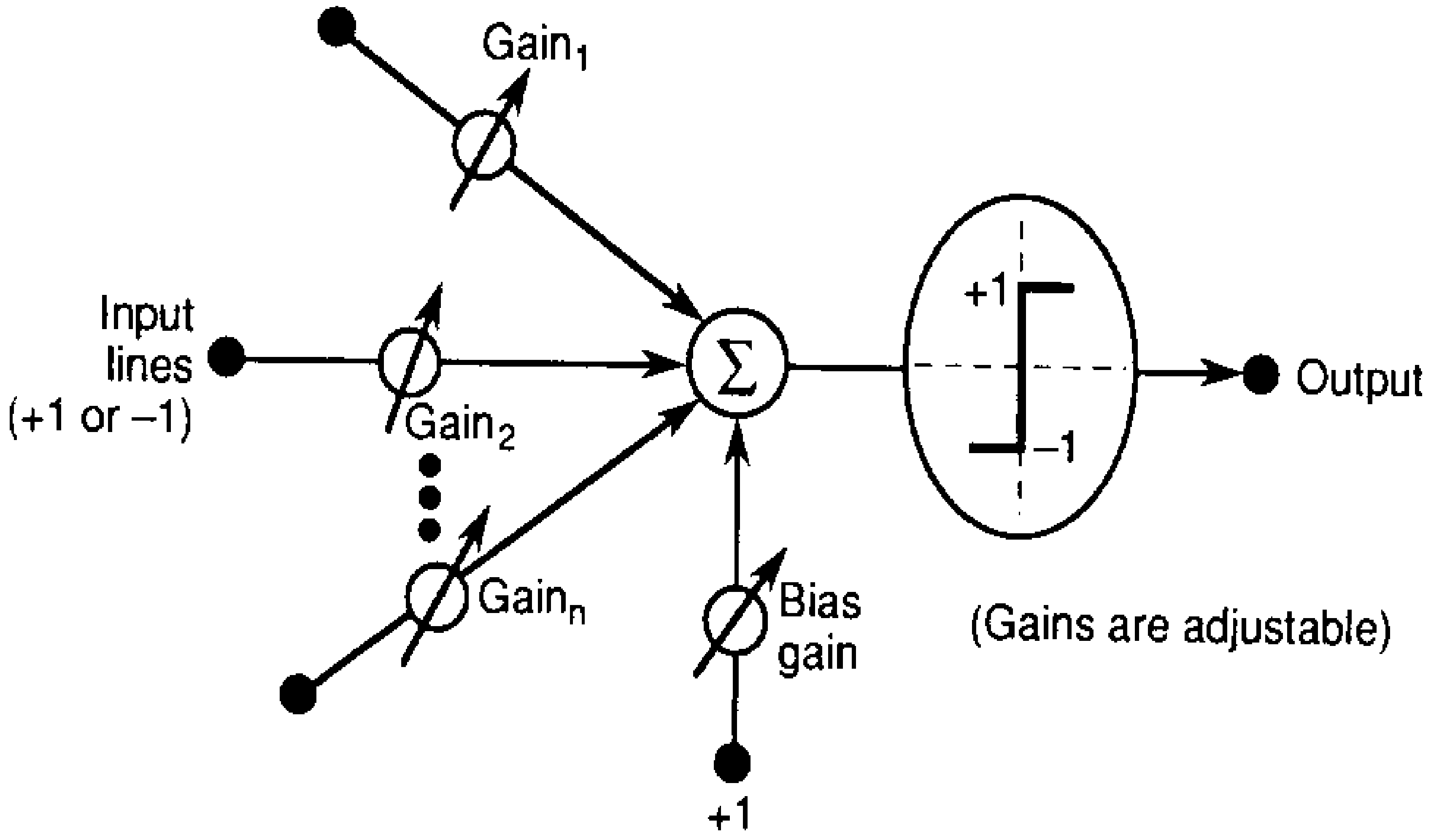


spinal medulla
Example image of VOXEL-MAN/brain
© 1994 IMDM University Hamburg, Germany

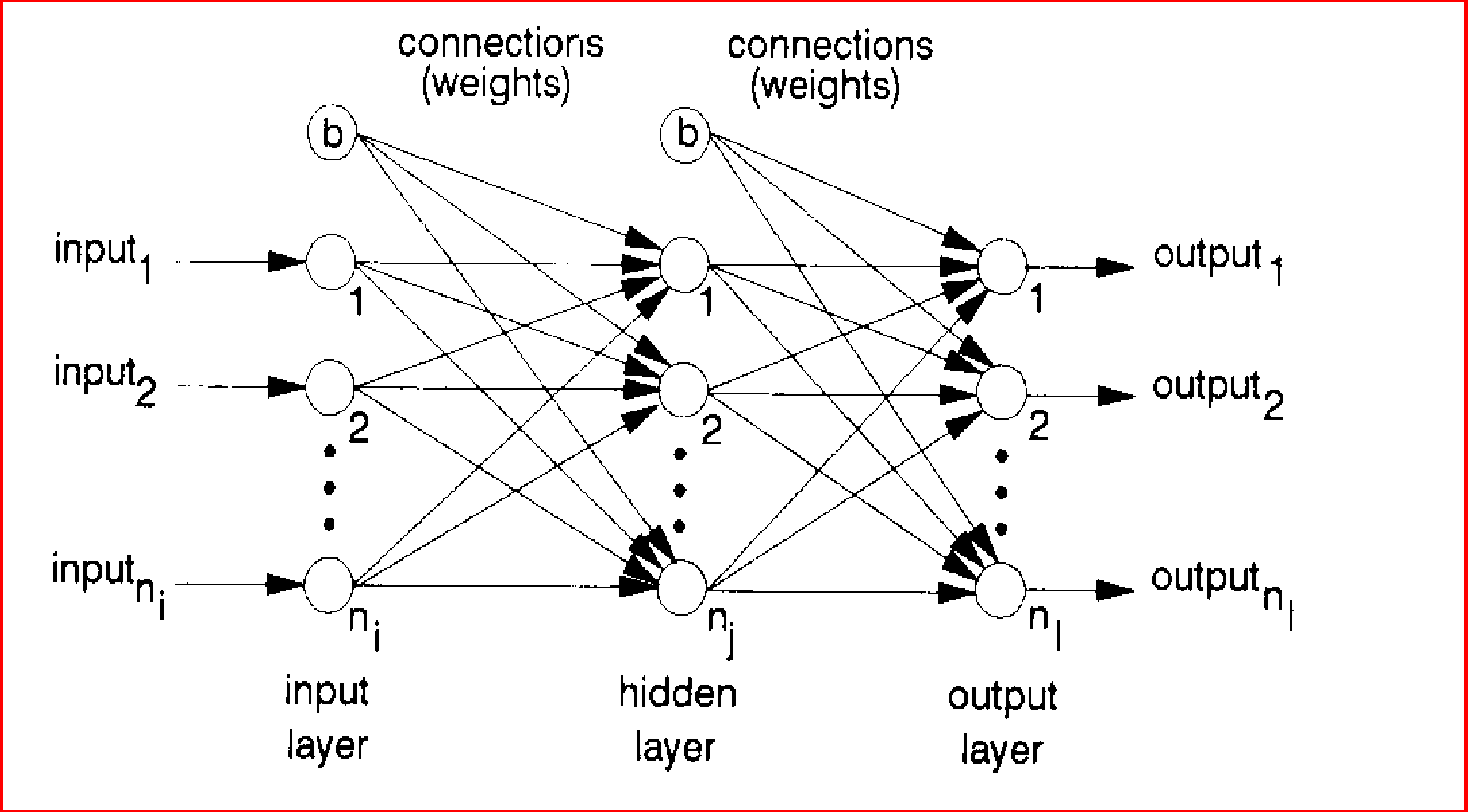
Mathematical Model of A Typical Neuron



Mathematical Model of A Typical Neuron



A Typical Two Layer Neural Network



Example: Working of a Typical ANN

Logic Gates / Truth Tables

- A typical AND gate truth table is shown below:

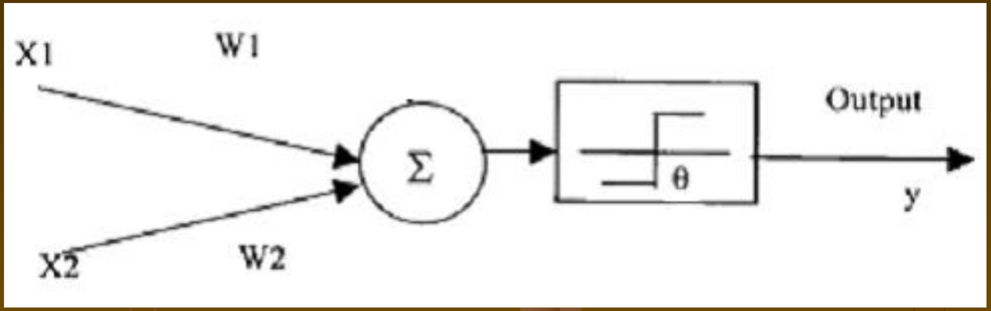
Inputs		Outputs
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Inputs		Outputs
x_1	x_2	y
–1	–1	–1
–1	1	–1
1	–1	–1
1	1	1

- Lets design an artificial neural network based on the above truth table.

Example: Working of a Typical ANN

Artificial Neuron Model



Implementation of AND function

Let $W_1 = W_2 = 1$

X_1	X_2	$X_1W_1 + X_2W_2$	Y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

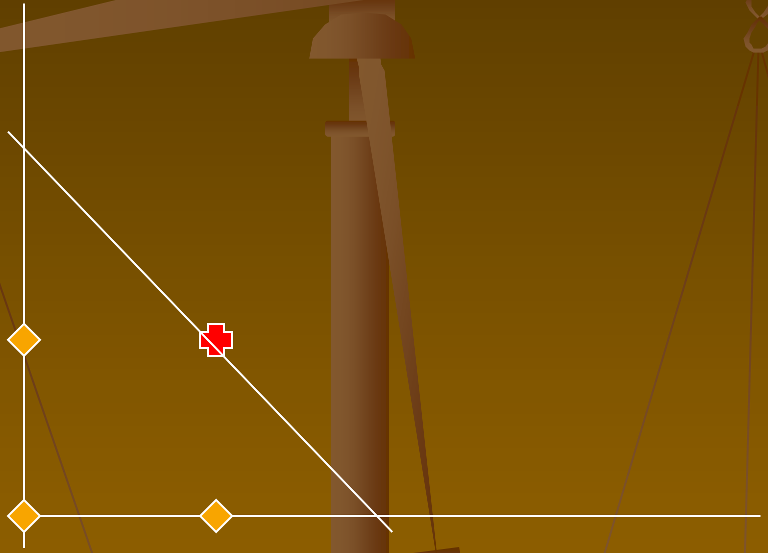
If we make $\theta = 2$ (or any value >1 but <=2), we will get correct results with a unit step activation function

Example: Working of a Typical ANN

Artificial Neuron Model

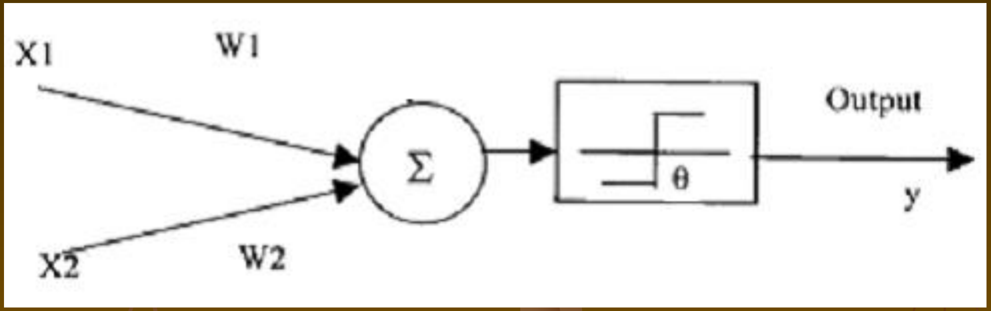
If we place the 4 points in a two coordinate system (X1 and X2), we have drawn a line from (2, 0) to (0, 2) in the resulting plane

Any new data falling on the left side of the line will give an output of zero and the data on the right side of the line will be classified as one



Example: Working of a Typical ANN

Artificial Neuron Model



Implementation of OR function

Let $W_1 = W_2 = 1$

X_1	X_2	$X_1W_1 + X_2W_2$	Y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

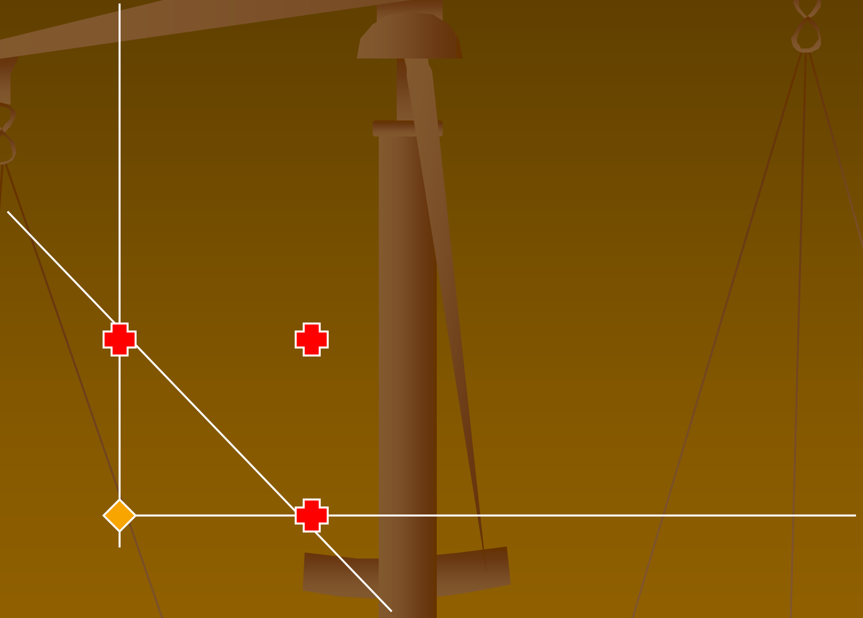
If we make $\theta = 1$ (or any value >0 but <=1), we will get correct results with a unit step activation function

Example: Working of a Typical ANN

Artificial Neuron Model

If we place the 4 points in a two coordinate system (X1 and X2), we have drawn a line from (1, 0) to (0, 1) in the resulting plane

Any new data falling on the left side of the line will give an output of zero and the data on the right side of the line will be classified as one



Class Activity

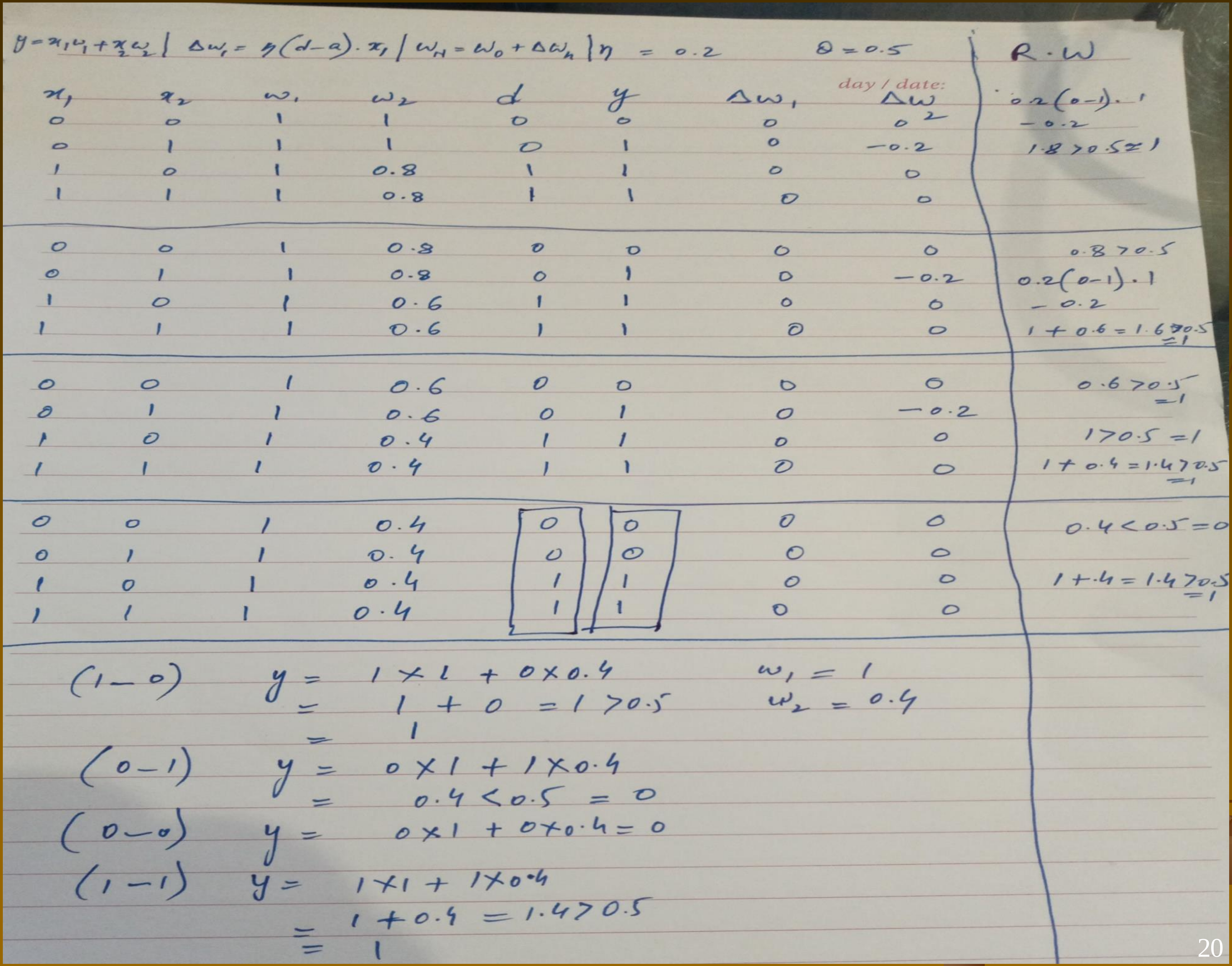
Show the mathematical working of Artificial Neural Network by taking the case in figure below. First two columns are the input values for X1 and X2 and the third column is the desired output.

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

- Learning rate = 0.2
- Threshold = 0.5
- Actual output = W1X1+W2X2
- Next weight adjustment = Wn+∆Wn
- Change in weight = ∆Wn = learning rate * (desired output- actual output) * Xn
- In Figure. First two columns are input vectors x1 and x2 and last column is the desired output y.

- Show the complete iterations for acquiring the desired output?

x1	**x2**	**w1**	**w2**	**d**	**y**	**∆ w1**	**∆ w2**



$y = x_1 w_1 + x_2 w_2$ | $\Delta w_1 = \eta (d - a) \cdot x_1$ | $w_{n+1} = w_0 + \Delta w_n$ | $\eta = 0.2$ $\theta = 0.5$ | R · W

day / date:

x_1	x_2	w_1	w_2	d	y	Δw_1	Δw_2	
0	0	1	1	0	0	0	0	$0.2(0-1) \cdot 1$
0	1	1	1	0	1	0	-0.2	-0.2
1	0	1	0.8	1	1	0	0	$1.8 > 0.5 = 1$
1	1	1	0.8	1	1	0	0	
0	0	1	0.8	0	0	0	0	$0.8 > 0.5$
0	1	1	0.8	0	1	0	-0.2	$0.2(0-1) \cdot 1$
1	0	1	0.6	1	1	0	0	-0.2
1	1	1	0.6	1	1	0	0	$1 + 0.6 = 1.6 > 0.5 = 1$
0	0	1	0.6	0	0	0	0	$0.6 > 0.5 = 1$
0	1	1	0.6	0	1	0	-0.2	
1	0	1	0.4	1	1	0	0	$1 > 0.5 = 1$
1	1	1	0.4	1	1	0	0	$1 + 0.4 = 1.4 > 0.5 = 1$
0	0	1	0.4	0	0	0	0	$0.4 < 0.5 = 0$
0	1	1	0.4	0	0	0	0	
1	0	1	0.4	1	1	0	0	$1 + .4 = 1.4 > 0.5 = 1$
1	1	1	0.4	1	1	0	0	

$(1-0)$ $y = 1 \times 1 + 0 \times 0.4$ $= 1 + 0 = 1 > 0.5$ $= 1$ $w_1 = 1$ $w_2 = 0.4$

$(0-1)$ $y = 0 \times 1 + 1 \times 0.4$ $= 0.4 < 0.5 = 0$

$(0-0)$ $y = 0 \times 1 + 0 \times 0.4 = 0$

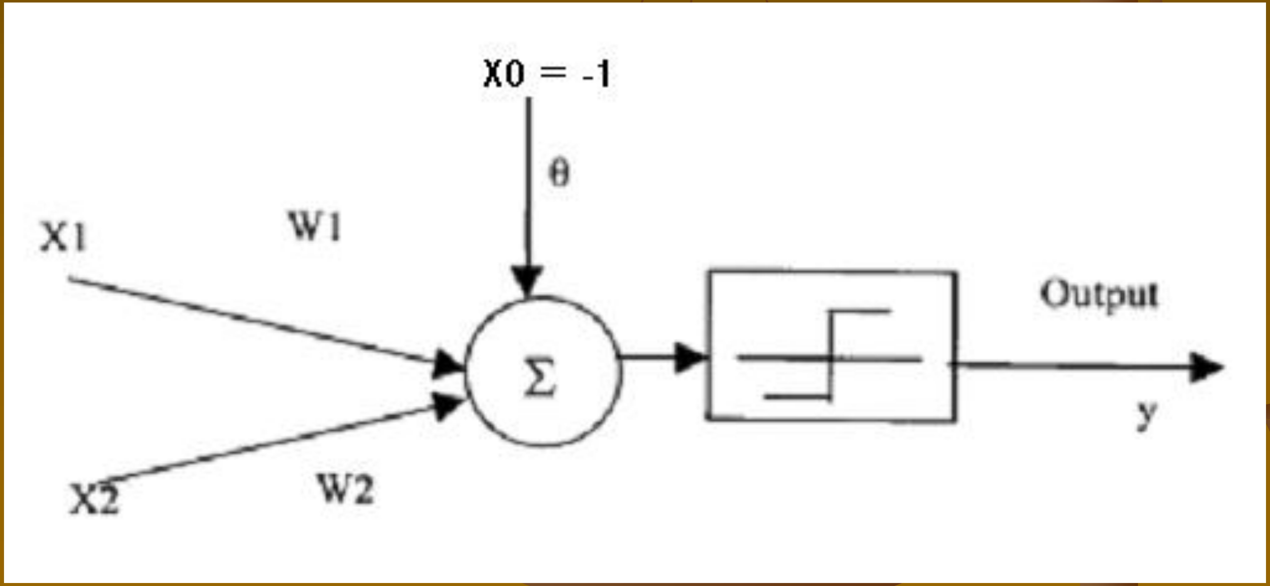
$(1-1)$ $y = 1 \times 1 + 1 \times 0.4$ $= 1 + 0.4 = 1.4 > 0.5$ $= 1$

Example: Working of a Typical ANN

Artificial Neuron Model

If we want to utilize a unit step function centered at zero for both AND and OR neurons, we can incorporate another input X0 constantly set at –1

The weight W0 corresponding to this input would be the θ, calculated previously



Example: Working of a Typical ANN

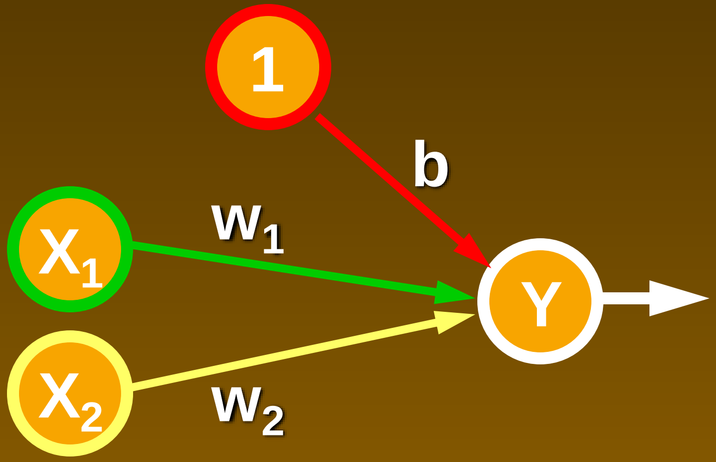
Setting of weights (Training)

X_1	X_2	Y
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

Example: Working of a Typical ANN

Logic Gates / Truth Tables

- A simple neural net architecture can be developed as shown in the figure below:



$$y = b + w_1x_1 + w_2x_2$$

$$f(x) = \begin{cases} 1 \text{ if sum of inputs} > 0 \\ -1 \text{ otherwise} \end{cases}$$

- The three unknowns in this problem are: b, w_1, w_2.
- These can be found very easily by substituting the values from the truth table, and then solving a set of linear equations.
- There is however, one problem that we have more constraints than the number of unknowns. Thus there could be more than one possible solutions.
- Can we work out these unknowns? How?
- One possible solution set is $b=-2$, $w_1=2$, $w_2=2$. Are there other possibilities?

Training of ANN

Need for Training

- In the simple example considered here, the number of unknowns (weights and biases) is small.
- In a more general neural architecture to solve realistic problems of interest, the number of unknowns becomes very large (think about a multilayered neural architecture). Also in such cases we may not have sufficient information available to us.
- Thus general methods of training are required to find out the unknown weights and biases. The two broad categories being:

  1. Supervised Training
  2. Unsupervised Training

Training of ANNs

Supervised Training

- Training is accomplished by presenting a sequence of training vectors or patterns, each with an associated target output vector.
- The weights are then adjusted according to a learning algorithm.
- During training, the network develops an associative memory. It can then recall a stored pattern when it is given an input vector that is sufficiently similar to a vector it has learned.

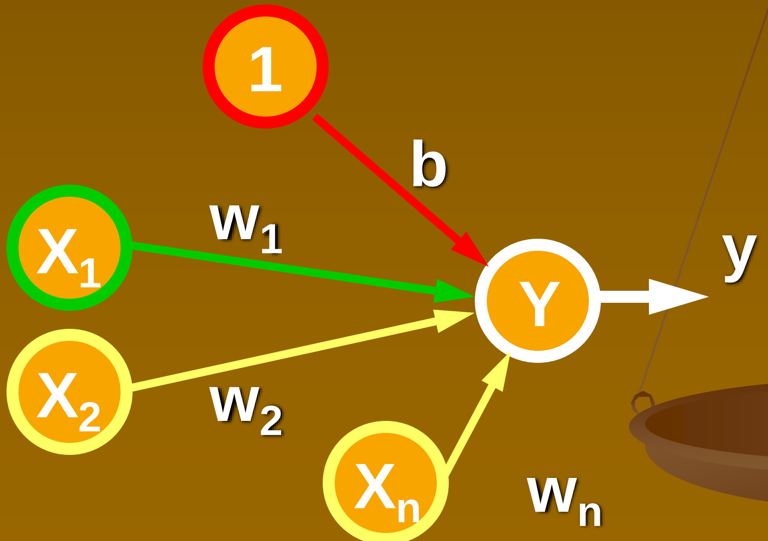
Unsupervised Training

- A sequence of input vectors is provided, but no traget vectors are specified in this case.
- The net modifies its weights and biases, so that the most similar input vectors are assigned to the same output unit.

Training a Single Layer Neural Net for Pattern Classification

The Hebb Rule

- The basic idea behind the Hebb rule is to modify the synapse strengths (weights) in a manner such that if two interconnected neurons are both "on" at the same time, then the weights between those neurons should be increased.
- A stronger form of learning occurs if we also increase the weights if both the neurons are "off" at the same time. This extended version is now referred to as the Hebb rule.
- A typical Hebb Net is shown in the figure:



$$y = b + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

$$f(x) = \begin{cases} 1 \text{ if sum of inputs} > 0 \\ -1 \text{ otherwise} \end{cases}$$

Training a Single Layer Neural Net for Pattern Classification

Algorithm for Hebb Net

1. Initialize all weights ($w_i = 0$ for $i = 1$ to n)

2. For each input training vector and target output pair <u>s</u>:t, do the following steps

    3. Set activations for input units:

        $x_i = s_i$ for $i = 1$ to n

    3. Set activations for output units:

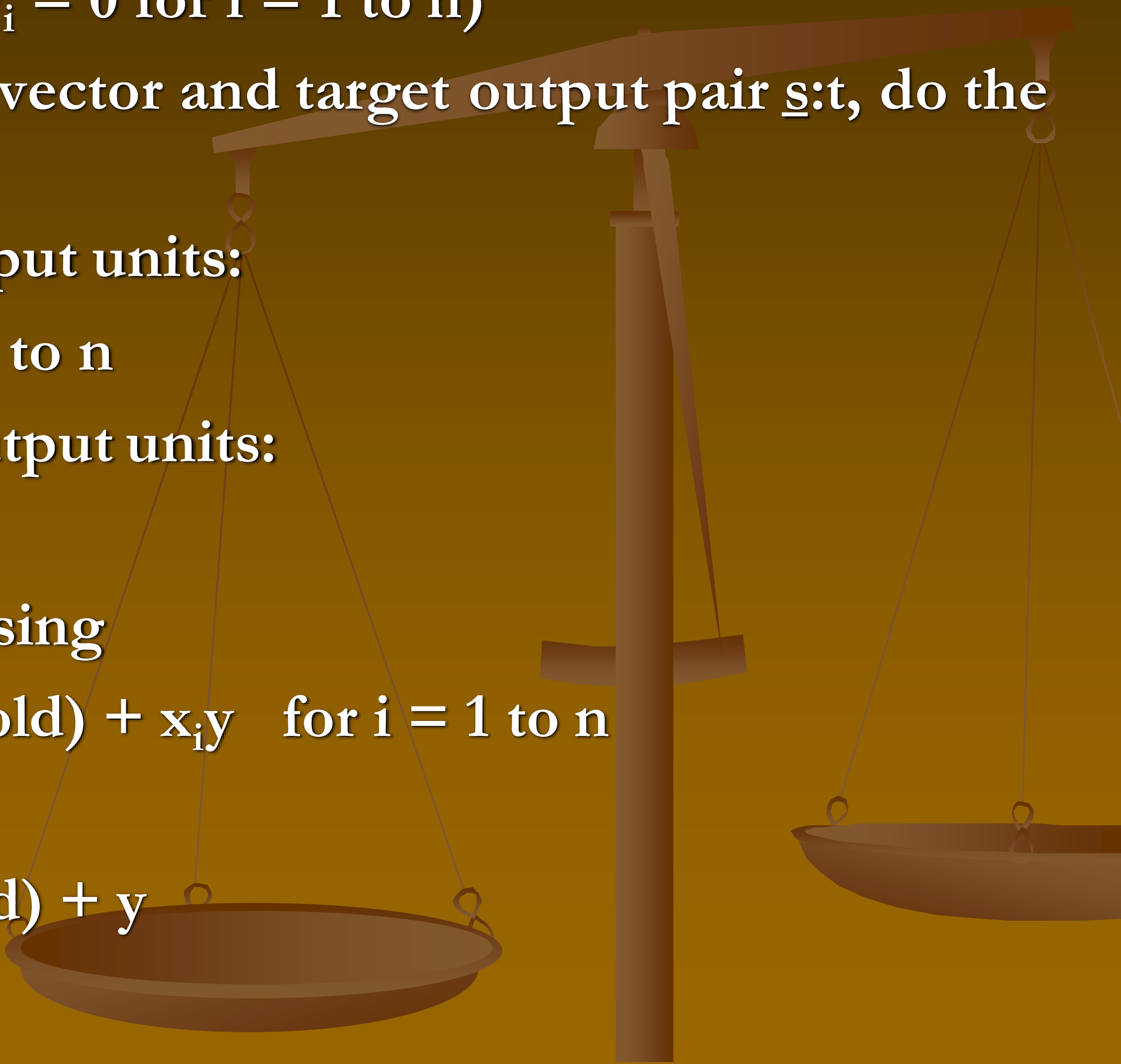
        $y = t$

    3. Adjust the weights using

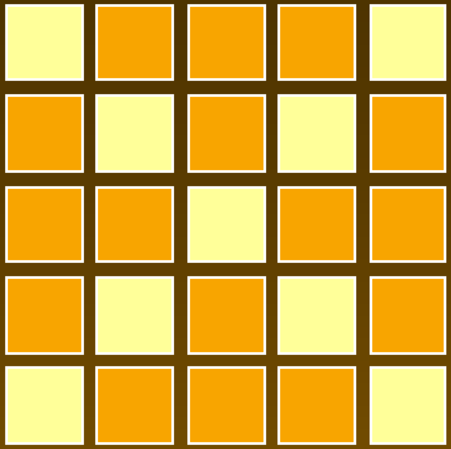
        $w_i(\text{new}) = w_i(\text{old}) + x_i y$ for $i = 1$ to n

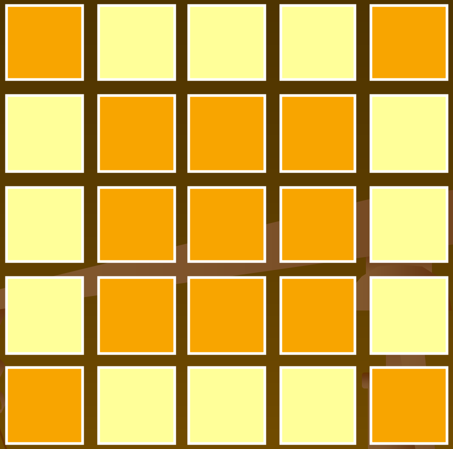
    Adjust the bias

        $b(\text{new}) = b(\text{old}) + y$



A Hebb net to classify two-dimensional input patterns (representing letters)

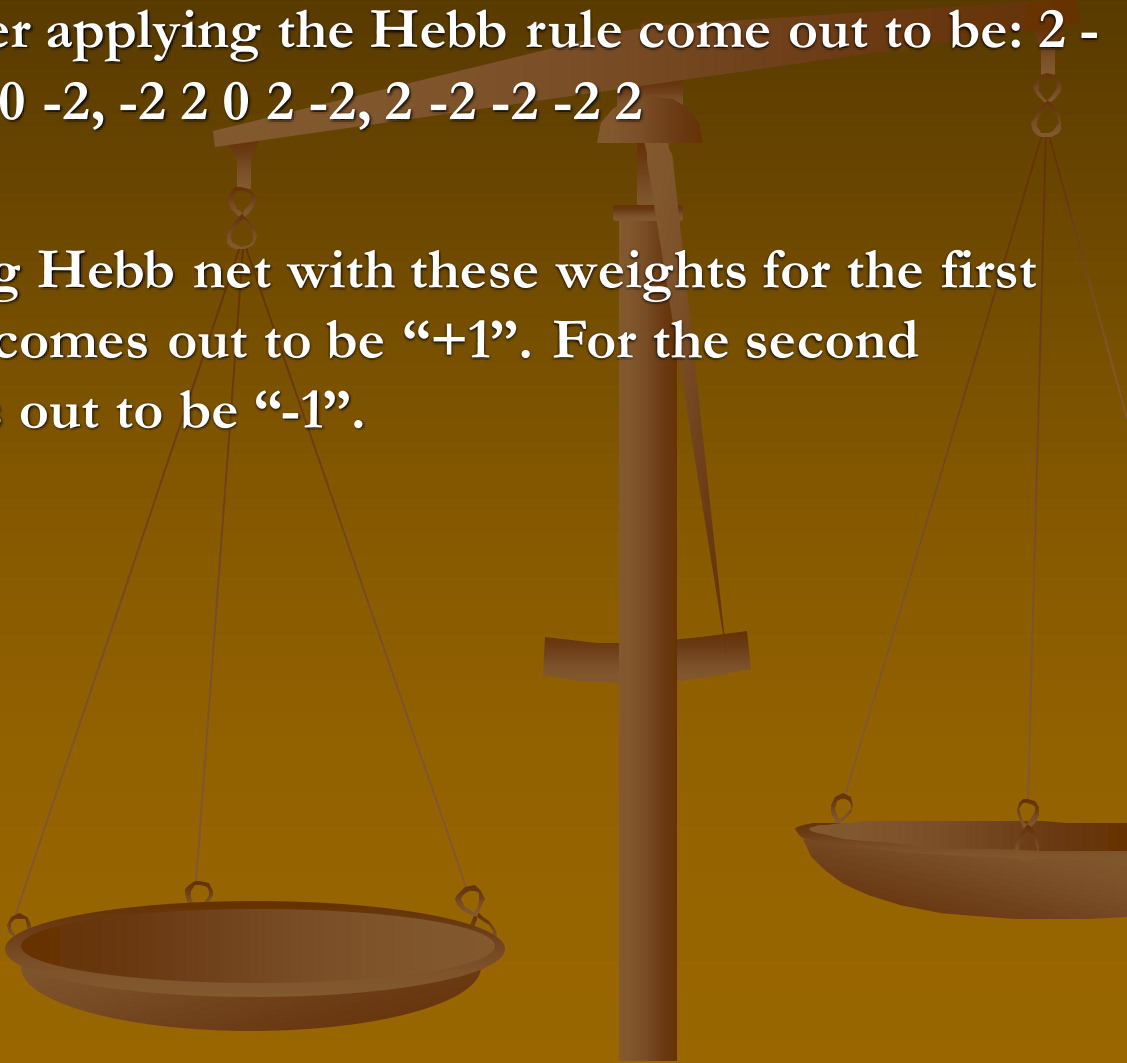

PATTERN 1


PATTERN 2

- We treat the above as a pattern classification problem with one output class (i.e. class "X").
- We will take pattern "X" to be of class "X" and "O" of not "X".
- The two patterns can be written in the form of bipolar vectors:
- Pattern 1: (1 -1 -1 -1 1, -1 1 -1 1 -1, -1 -1 1 -1 -1, -1 1 -1 1 -1, 1 -1 -1 -1 1)
- Pattern 2:(-1 1 1 1 -1, 1 -1 -1 -1 1, 1 -1 -1 -1 1, 1 -1 -1 -1 1, -1 1 1 1 -1)

A Hebb net to classify two-dimensional input patterns (representing letters)

- The correct response for the first pattern is “+1” and for the second pattern is “-1”.
- The weights obtained after applying the Hebb rule come out to be: 2 -2 -2 -2 2, -2 2 0 2 -2, -2 0 2 0 -2, -2 2 0 2 -2, 2 -2 -2 -2 2
- The bias weight is 0.
- When we test the resulting Hebb net with these weights for the first input pattern, the output comes out to be “+1”. For the second pattern, the output comes out to be “-1”.



PERCEPTRON

- The perceptron learning rule is a more powerful learning rule than the Hebb rule.
- Under suitable assumptions, its iterative learning procedure can be proved to converge to the correct weights i.e. the weights that allow the net to produce the correct output value for each of the training input pattern.

Algorithm

1. Initialize weights and bias (set to zero). Also set the learning rate a such that ($0 < a <= 1$).
2. While stopping condition is false, do the following steps.
3. For each training pair s:t, do steps 4 to 6
4. Set the activations of input units. $x_i = s_i$
5. Compute the response of the output unit.

PERCEPTRON

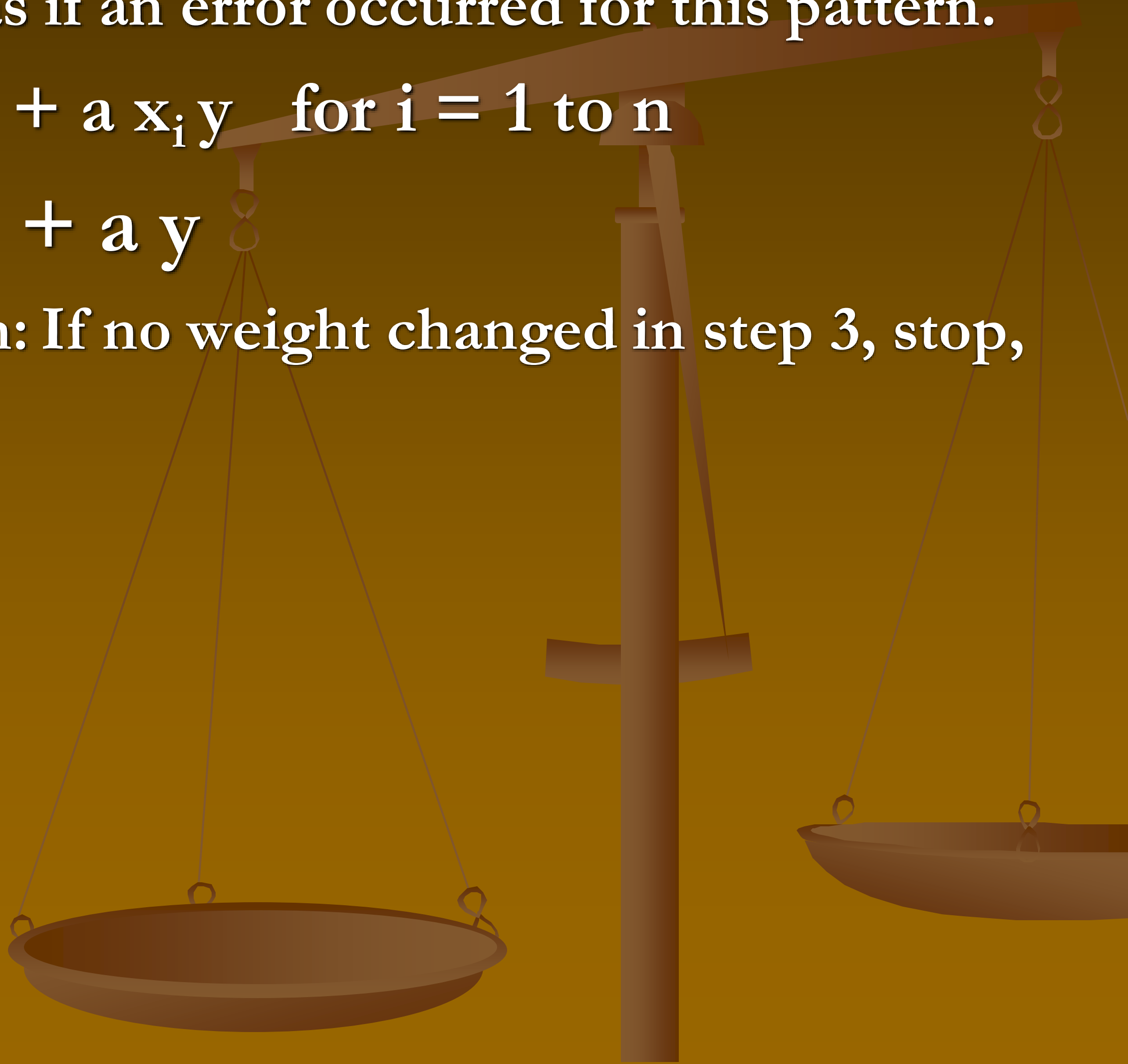
Algorithm

5. Update weights and bias if an error occurred for this pattern.

   $w_i(\text{new}) = w_i(\text{old}) + a\, x_i\, y \quad \text{for } i = 1 \text{ to } n$

   $b(\text{new}) = b(\text{old}) + a\, y$

6. Test stopping condition: If no weight changed in step 3, stop, else continue.



Common Activation Functions

1. Binary Step Function

(a "threshold" or "Heaviside" function)

$$f(x) = \begin{cases} 0 & if\ \ x < 0 \\ 1 & if\ \ x \geq 0 \end{cases}$$

2. Bipolar Step Function

$$f(x) = \begin{cases} -1 & if\ \ x < 0 \\ 1 & if\ \ x \geq 0 \end{cases}$$

3. Binary Sigmoid Function

(Logistic Sigmoid)

$$f(x) = \frac{1}{1+\exp(-\sigma x)};\ f'(x) = \sigma f(x)[1 - f(x)]$$

3. Bipolar Sigmoid

$$g(x) = \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)};\ g'(x) = \frac{\sigma}{2}[1+g(x)][1-g(x)]$$

4. Hyperbolic - Tangent

$$h(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)};\ h'(x) = [1+h(x)][1-h(x)]$$