Multilayer Feed-forward Neural Network

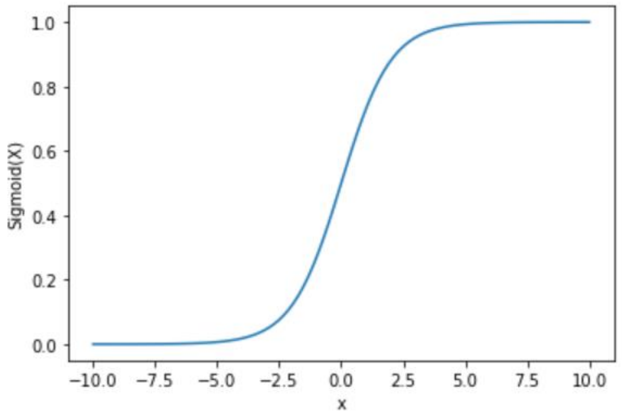
Dr R Bhargavi
Vellore Institute of Technology

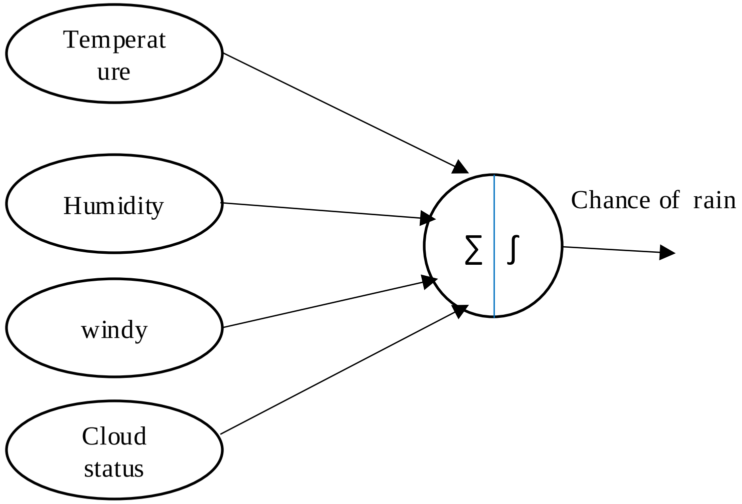
Threshold Activation Function - Limitations

- Threshold/step function is not differentiable
  - Not suitable for use in gradient-based optimization algorithms like backpropagation, which rely on computing gradients for updating weights during training.
  - The lack of derivatives makes it challenging to apply efficient optimization techniques.
- The threshold function has a fixed output range (0 or 1), and there's no notion of the strength of activation. This can limit the expressiveness of the model, especially when dealing with tasks that require a continuous range of output values.
  - What if we were interested in predicting the chance of raining instead of whether it rains or not?

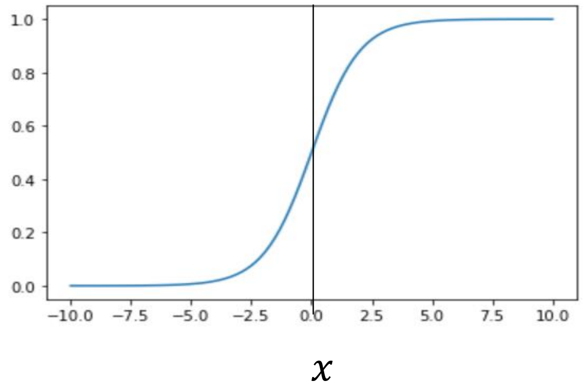
Sigmoid Perceptron

- Sigmoid function: $Sigmoid(x) = \frac{1}{1+e^{-x}}$
- Squashes the output between 0 and 1.
- When threshold function is replaced with Sigmoid function, the output of a neuron/node becomes as $y = \frac{1}{1+ e^{-\sum_{i=1}^{m} w_i x_i + b}}$
- When used in output layer, it can be interpreted as the probability that y (output variable) belongs to a particular category.
- For binary classification, a threshold can be used on y to result in either 0 or 1.

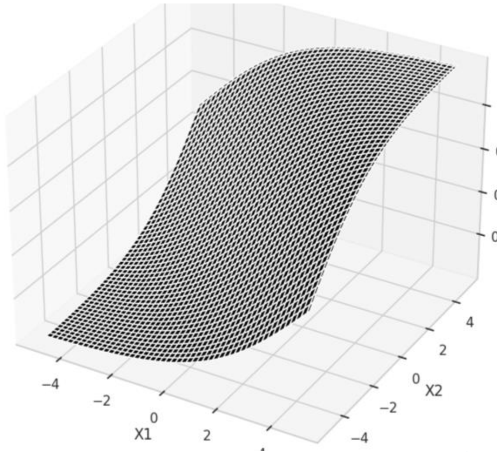




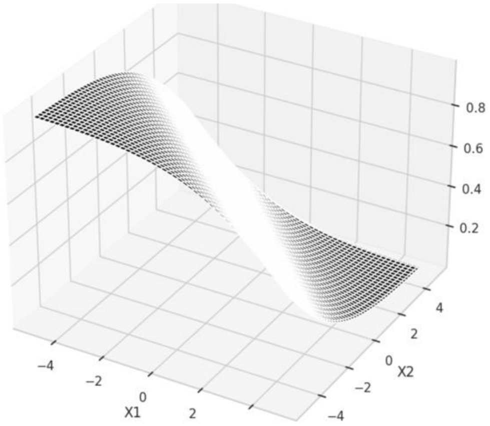
Sigmoid Function



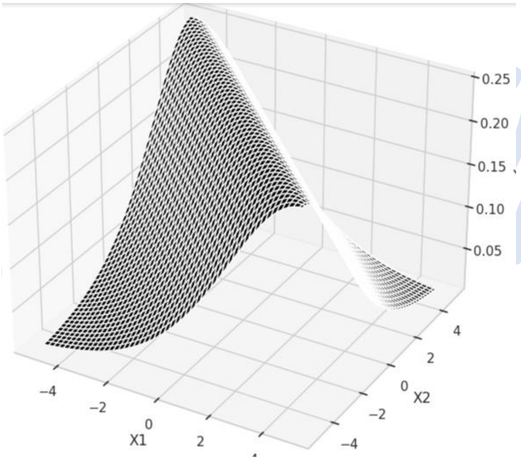
Sigmoid in 2-dim



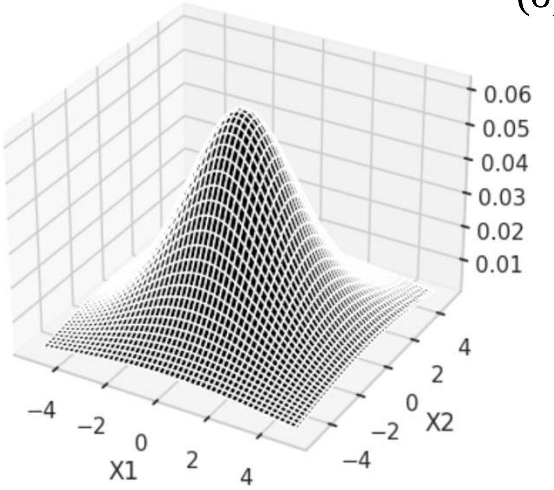
Sigmoid in 3-dim



Sigmoid in 3-dim
(opposite face)



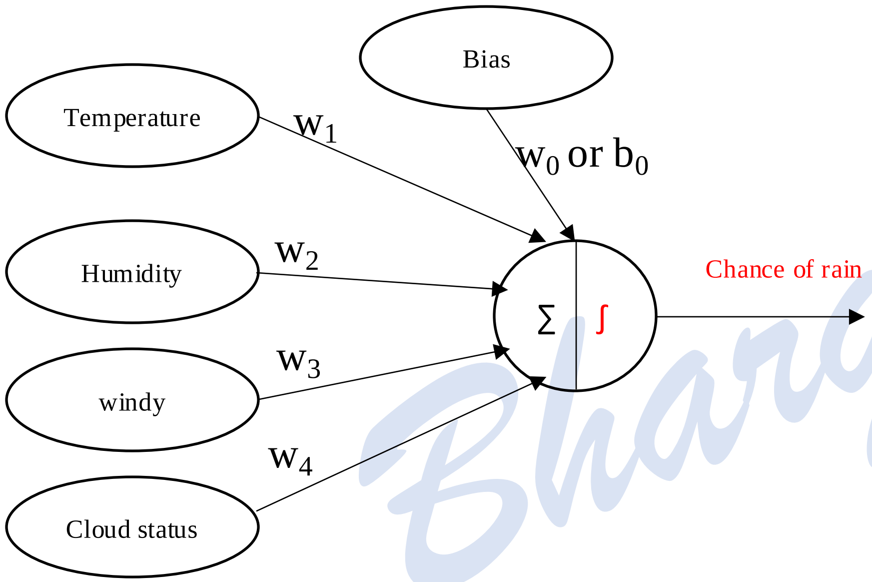
Combined opposite facing
Sigmoid functions - Ridge



Combined functions
of two ridges

Single Layer Neural Network with Sigmoid

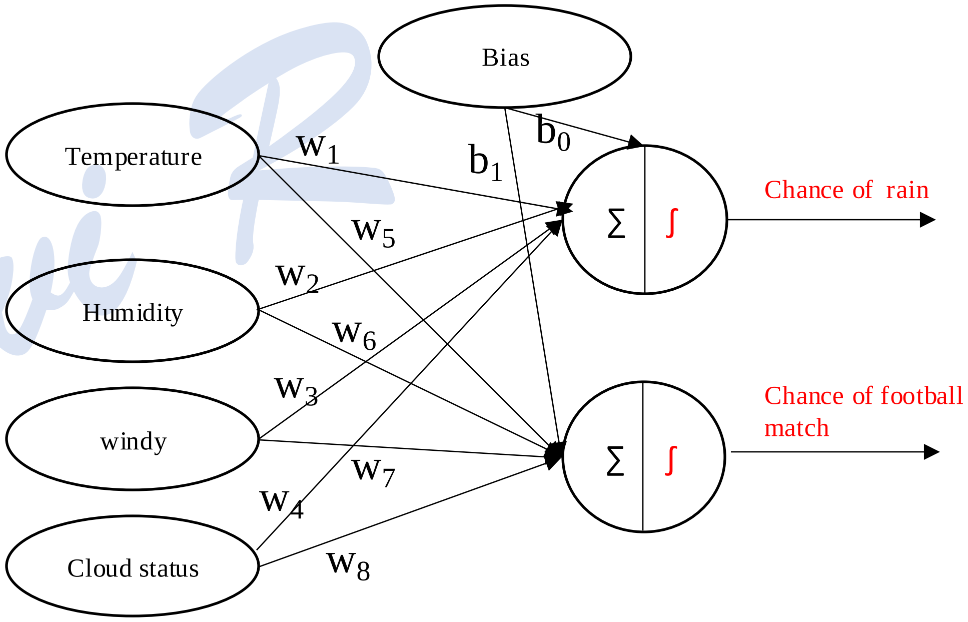
One output



Input Layer

Output Layer

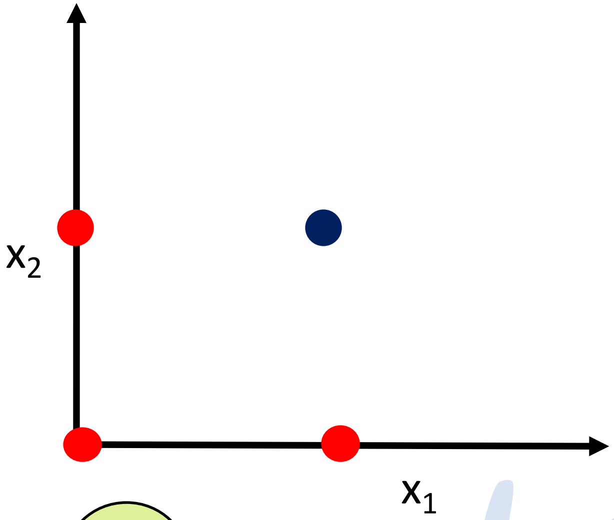
Multiple outputs

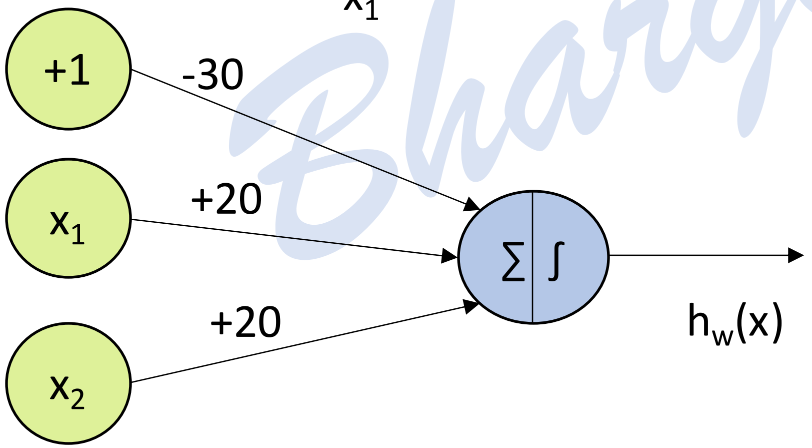


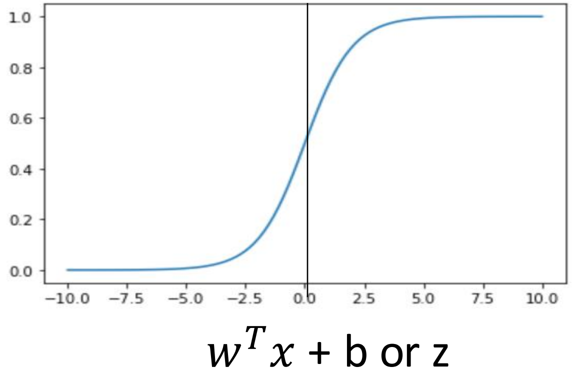
Input Layer

Output Layer

Example – Linear Function (AND)

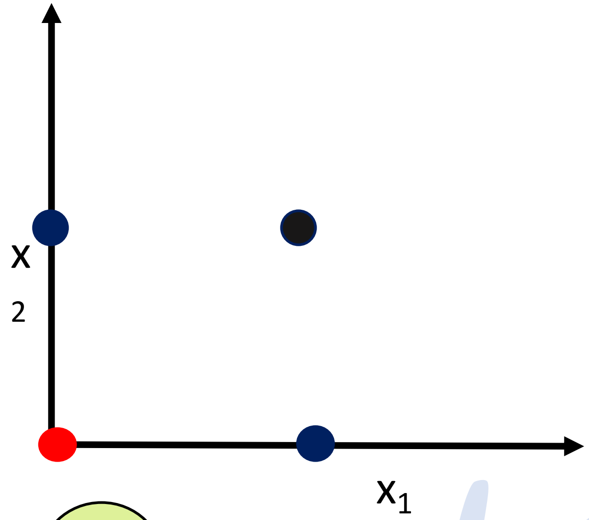


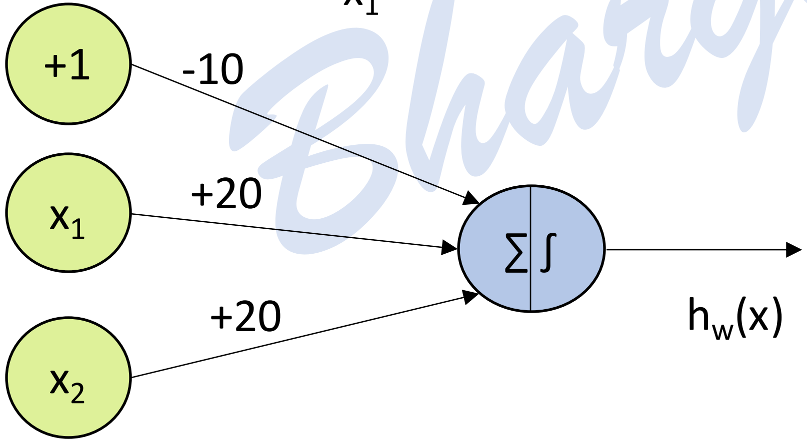


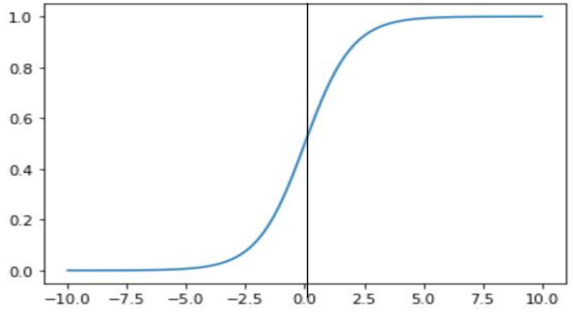


x_1	x_2	f(g(x))
0	0	f(-30) approx. 0
0	1	f(-10) approx. 0
1	0	f(-10) approx. 0
1	1	f(10) approx. 1

Example – Linear Function (OR)





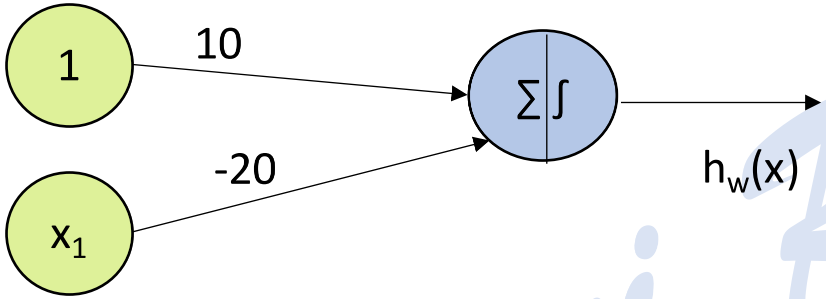


$w^T x$ + b or z

x_1	x_2	f(g(x))
0	0	f(-10) approx. 0
0	1	f(10) approx. 1
1	0	f(10) approx. 1
1	1	f(30) approx. 1

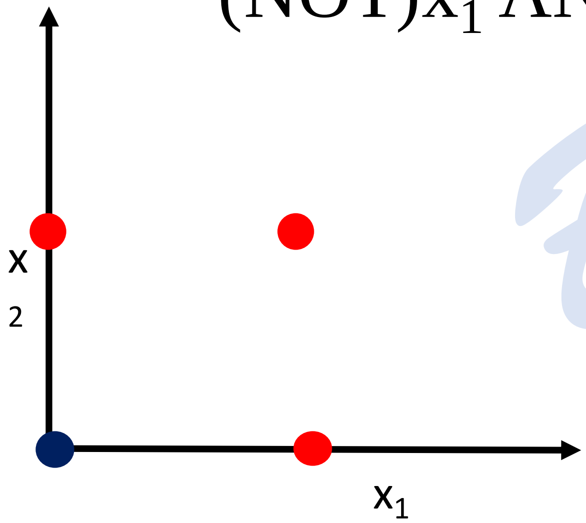
Examples -Linear Functions (cont..)

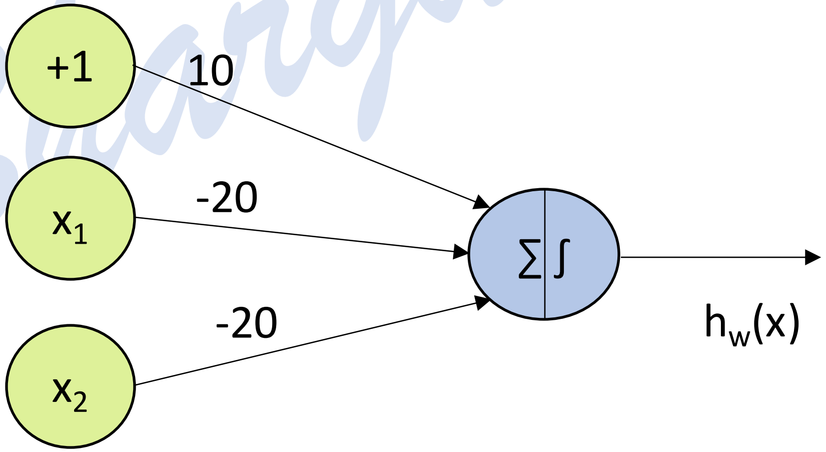
NOT



x_1	f(g(x))
0	1
1	0

(NOT)x_1 AND (NOT)x_2

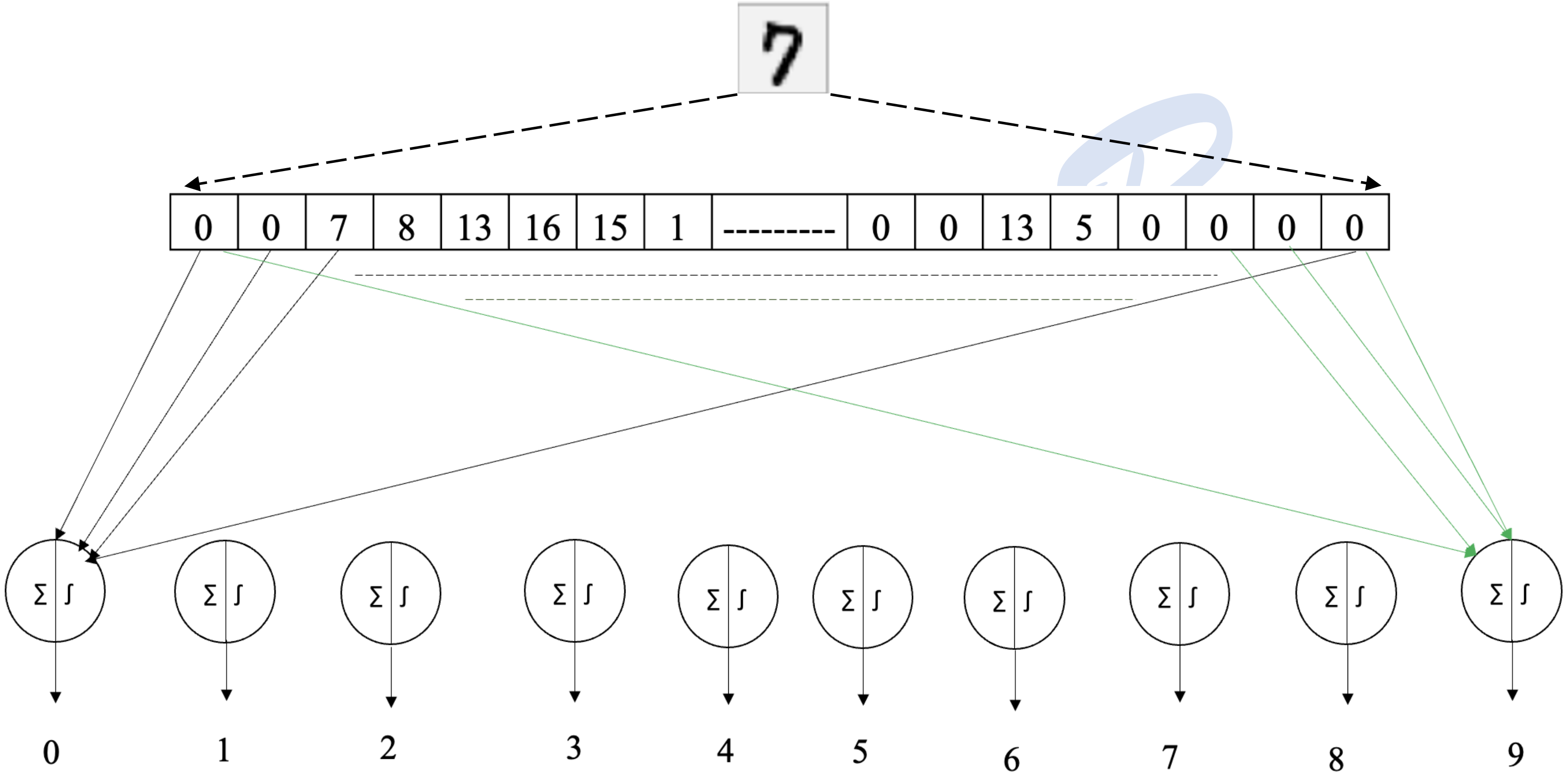




x_1	x_2	F(g(x))
0	0	1
0	1	0
1	0	0
1	1	0

Handwritten Digit Classification

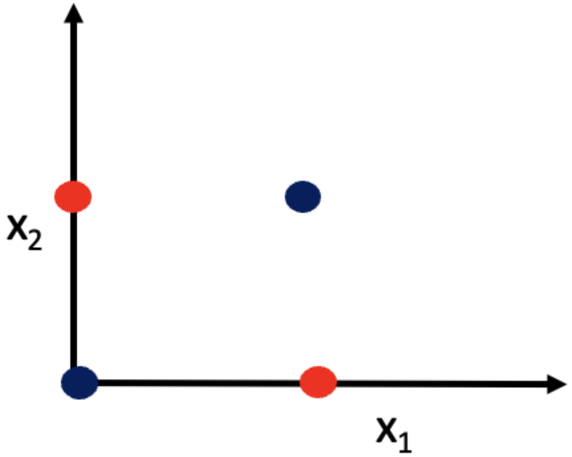
28 x 28 image (pixels representing the intensity of gray scale)



Single Layer Feed Forward Network - Limitations

- Single layer feed-forward network can be used to classify only the linearly separable data.
- Most of the real world applications have very complex non-linear relations between input and output. Hence can not be solved with single layer feed forward networks.
- Limited in their ability to represent complex functions.
- Overfit on simple problems or underfit on more complex problems.

XNOR Function

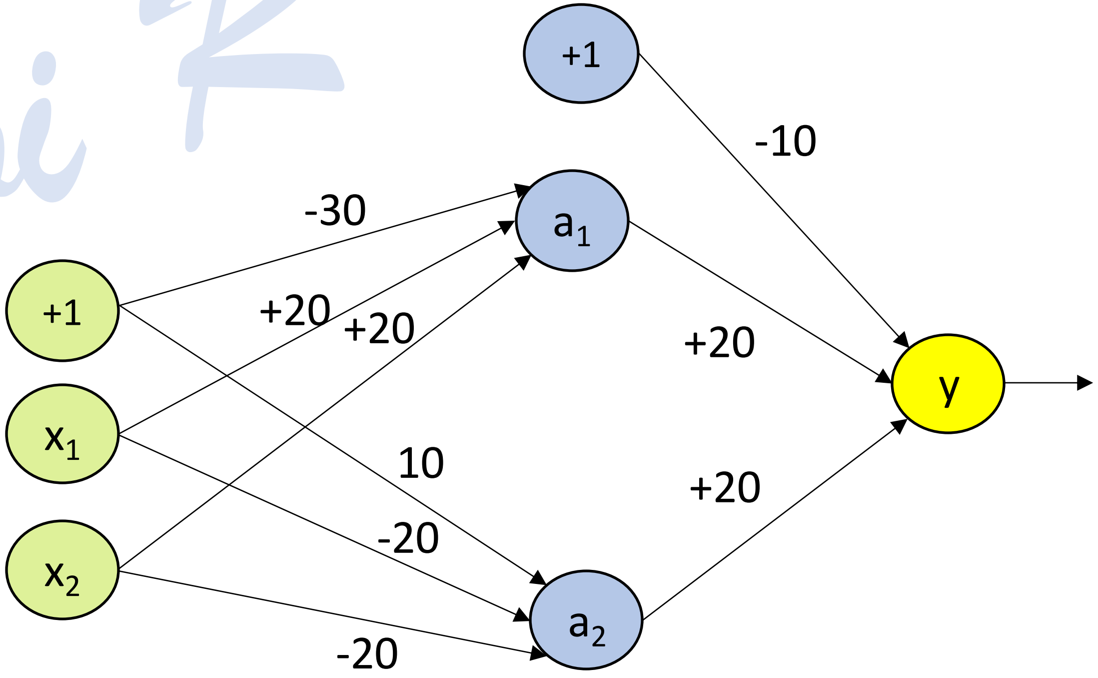


x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

XNOR With Multiple Layers of Nodes

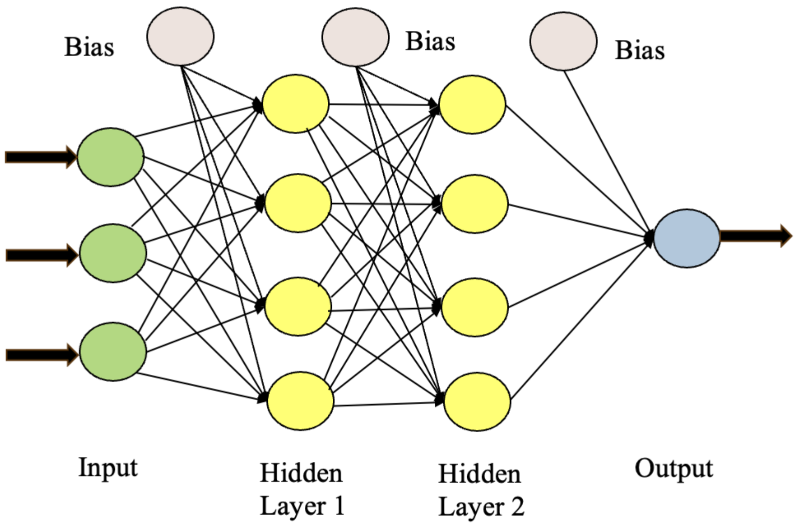
- Let us assume Threshold activation function g for each of the nodes and compute the output.

x_1	x_2	a_1	a_2	y = g(h(x))
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

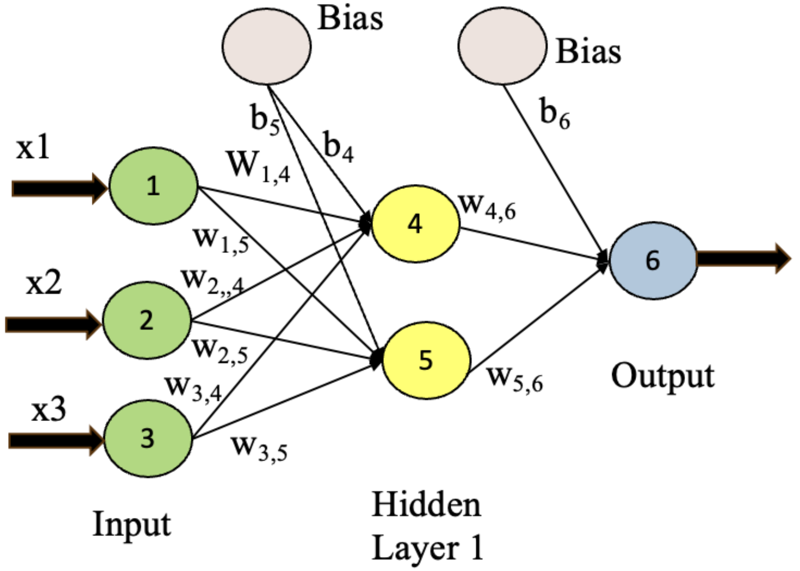


Multilayer Feed-forward Neural Network - Architecture

- Multilayer Perceptrons (MLP).
- One **input layer** consisting of input neurons.
- Input neurons don't apply any activation/computation.
- One **output layer** of one or more neurons.
- One or more **hidden layers**, each with a set of neurons.
- Output from nodes in a layer is used as input to the nodes in next layer hence the name **feed forward** neural networks.
- Neural networks with feedback loops are called as **Recurrent neural networks.**



Multilayer Feed-forward Neural Networks(cont…)



- $w_{i,j}$ – weight connecting node i in one layer to node j in the next layer
- Net input of any node j $= in_j = \sum_i w_{i,j}\, a_i + b_j$
- Output of a node $j = a_j = g(in_j)$ where g is any activation function
- Output for nodes in the input layer $a_j = x_j$

Learning in Multilayer Neural Networks

- Objective: Find weights and biases such that the computed output from the network approximates for the actual output.
- To quantify the objective we define a **cost function** or **loss function** as

$$Loss(h_w) = \sum_{k=1}^{n}\left(y_k - h_w(\boldsymbol{x})\right)^2 = \sum_{k=1}^{n}(y_k - a_k)^2.$$

- a_k : activation function output of the neuron(s) in the output layer.
- $a_k = \sigma(\sum_{i=1}^{m} w_i \,.\, x_i + b)$ (for sigmoid activation).
- $\boldsymbol{x}$: input sample (a vector (x_1, x_2, -------, x_m))
- y : corresponding actual output
- $h_w(\boldsymbol{x})$: predicted output for $\boldsymbol{x}$

Learning in Multilayer Neural Networks (cont…)

Back Propagation for weight updation

- A supervised learning algorithm used to train ANNs by minimizing the error between the predicted output and the actual target values.

Step 1- Forward Pass:

- Input data is fed forward through the network to produce the predicted output.
- For each neuron compute the weighted sum of its inputs, apply an activation function, and pass the result to the next layer.

Step 2 - Compute Error:

- The error is calculated by comparing the predicted output to the actual target values using a chosen loss or cost function.

Learning in Multilayer Neural Networks (cont…)

Step 3 - Backward Pass (Backpropagation):

- Starting from the output layer and moving backward through the network do:
- Compute Output Layer Gradients: Calculate the gradient of the loss with respect to the output of each neuron in the output layer.
- Update Output Layer Weights: Adjust the weights of connections in the output layer using the computed gradients and an optimization algorithm (commonly stochastic gradient descent).
- Propagate Gradients Backward: Compute the gradients for each neuron in the hidden layers by propagating the error backward through the network.

Learning in Multilayer Neural Networks (cont…)

Step - 4 Update Weights:

- For each layer, update the weights using the computed gradients and the optimization algorithm.
- The optimization algorithm adjusts the weights in the direction that reduces the error.

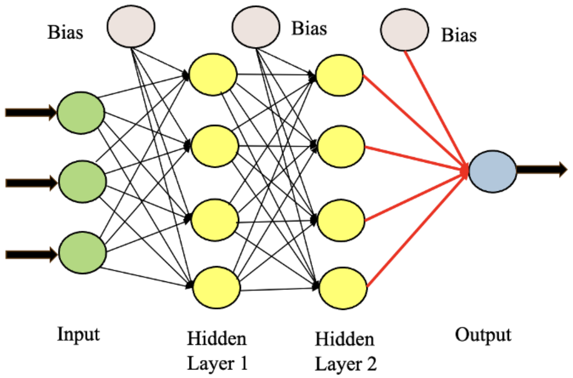
Repeat Steps 1- 4 for multiple iterations (epochs) or until the error converges to an acceptable level.

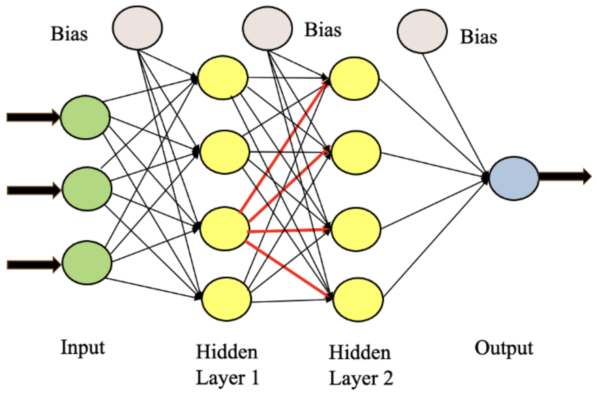
Learning in MLP – Backward Propagation

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ in *examples* **do**
    / Propagate the inputs forward to compute the outputs */*
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_j \leftarrow g(in_j)$
    / Propagate deltas backward from output layer to input layer */*
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
    / Update every weight in network using deltas */*
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

note: $\Delta[i] = \Delta_i, \Delta[j] = \Delta_j$

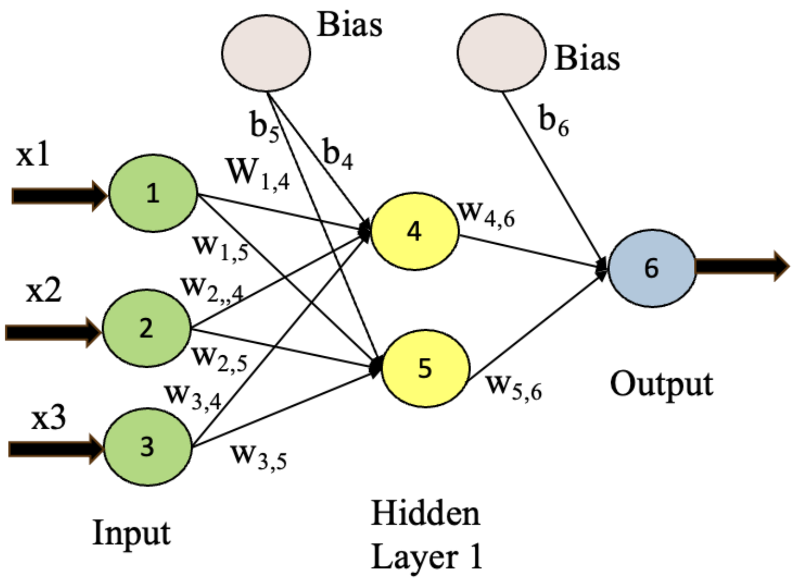




Source: Artificial Intelligence – A modern approach. Stuart J. Russel & Peter Norvig

Example

- Consider the network as shown in the figure.
- How the weights of this network are learned for the input **x** = (x1,x2,x3) = (1,0,1) and the corresponding output y =1 ?
- Learning rate ∝ = 0.9, and let the activation be Sigmoid
- Let the initial weights be as follows:



Input-Hidden layer

$w_{1,4}$	$w_{1,5}$	$w_{2,4}$	$w_{2,5}$	$w_{3,4}$	$w_{3,5}$
0.2	-0.3	0.4	0.1	-0.5	0.2

Hidden-Output layer

$w_{4,6}$	$w_{5,6}$
-0.3	-0.2

Bias weights

b_4	b_5	b_6
-0.4	0.2	0.1

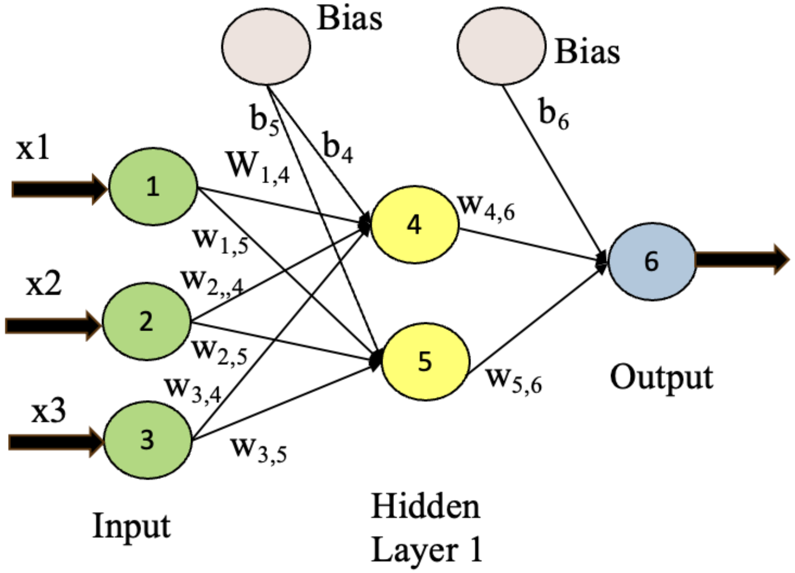
Example

Forward propagation of Inputs

Net Input and Output calculations : $in_j = \sum_i w_{i,j}\, a_i + bj$

Since the activation is Sigmoid, we have $a_j = \frac{1}{1+e^{-in_j}}$

x = (x1,x2,x3) =(1,0,1) , ∝ = 0.9 and y =1



Node (j)	Net input (in_j)	Output a_j
4	0.2 * 1 + 0.4 * 0 + -0.5 *1 + (-0.4) = -0.7	$1/(1+ e^{-(-0.7)}) = 0.332$
5	-0.3 * 1 + 0 + 0.2 + 0.2 = 0.1	$1/(1+ e^{-(0.1)}) = 0.525$
6	-0.3*0.332 + (-0.2*0.525) + 0.1 = -0.105	$1/(1+ e^{-(10.105)}) = 0.474$

Example (cont…)

Backpropagate the error

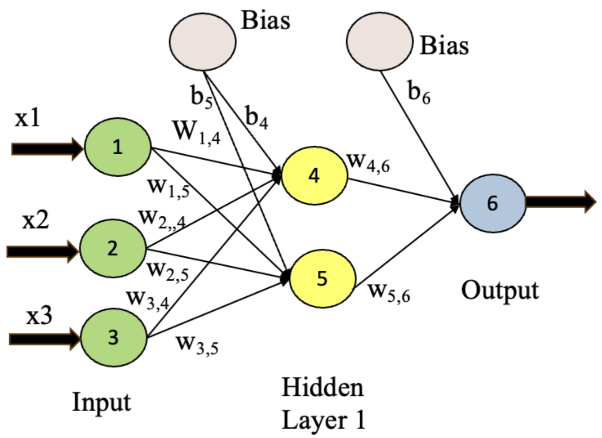
For each neuron in the Output layer compute the error as

$$\Delta_j = \ a_j * (1 - a_j) * (y_j - a_j)$$

For each neuron in the hidden layer compute the error as

$$\Delta_i = \ a_i * (1 - a_i) * \sum_j w_{i,j}\, \Delta_j$$

Here *j* represents the nodes in the next higher layer



Node (*j*)	Δ_j
6	0.474 * (1 - 0.474) * (1 - 0.474) = 0.1311
5	0.525 * (1 - 0.525) * (-0.2 * 0.1311) = -0.0065
4	0.332 * (1- 0.332) * (-0.3 * 0.1311) = -0.0087

Example (cont…)

Update the weights

Weights are updated using the formula

$$w_{i,j} = w_{i,j} + \propto * \; a_i * \; \Delta_j$$

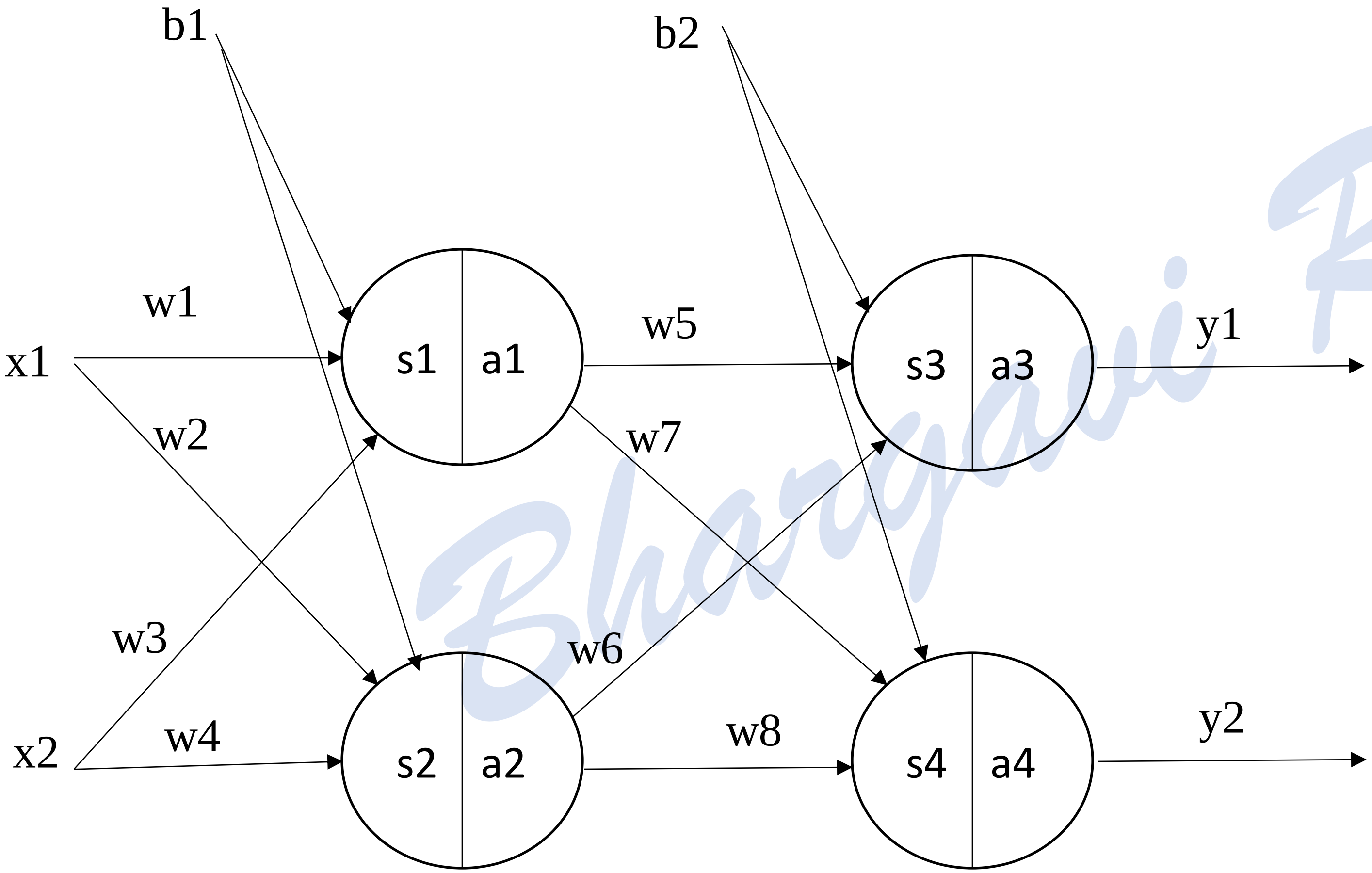
Weights	Updated Weights
$w_{4,6}$	-0.3 + (0.9 * 0.332 * 0.1311) = -0.261
$w_{5,6}$	-0.2 + (0.9 * 0.525 * 0.1311) = -0.138
$w_{1,4}$	0.2 + (0.9 * 1 * -0.0087) = 0.192
$w_{1,5}$	-0.3 + (0.9 * 1 * -0.0065) = -0.306
$w_{2,4}$	0.4 + (0.9 * 0 * -0.0087) = 0.4
$w_{2,5}$	0.1 + (0.9 * 0 * -0.0065) = 0.1
$w_{3,4}$	-0.5 + (0.9 * 1 * -0.0087) = -0.508
$w_{3,5}$	0.2 + (0.9 * 1 * 0.0065) = 0.194

Bias weights are updated using the formula

$$b_i = b_i + \propto * \; \Delta_i$$

Bias	Updated Bias
6	0.1 + (0.9 * 0.1311)= 0.218
5	0.2 + (0.9 * -0.0065) = 0.194
4	-0.4 + (0.9 * -0.0087) = -0.408

Example

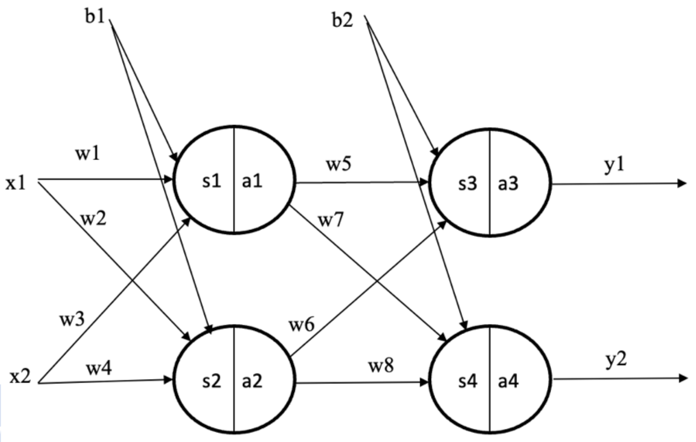


x1 = 0.1
x2 = 0.5
y1 = 0.05
y2 = 0.95
w1 = 0.1
w2 = 0.2
w3 = 0.3
w4 = 0.4
w5 = 0.5
w6 = 0.6
w7 = 0.7
w8 = 0.8
b1 = 0.25
b2 = 0.35

Example (cont…)

Forward pass

Net input	Output a_j
s1 = 0.1*0.1 + 0.3*0.5 + 0.25 = 0.41	a1 = $1/(1+ e^{-(0.41)}) = 0.601$
s2 = 0.2*0.1 + 0.4*0.5 + 0.25 = 0.47	a2 = $1/(1+ e^{-(0.47)}) = 0.615$
s3 = 0.5*0.601 + 0.6*0.615 + 0.35 = 1.0195	a3 = $1/(1+ e^{-(1.0195)}) = 0.7349$
s4= 0.7*0.601 + 0.8*0.615 + 0.35 = 1.2627	a4 = $1/(1+ e^{-(1.2627)}) = 0.7795$



x1 = 0.1
x2 = 0.5
y1 = 0.05
y2 = 0.95
w1 = 0.1
w2 = 0.2
w3 = 0.3
w4 = 0.4
w5 = 0.5
w6 = 0.6
w7 = 0.7
w8 = 0.8
b1 = 0.25
b2 = 0.35

Compute Total Error

$$Error_{Total} = \frac{1}{2}\sum(target - predicted)^2$$

$$Error_{Total} = Error_1 + Error_2$$

$$Error_1 = \frac{1}{2}(y_1 - \hat{y}_1)^2 = \frac{1}{2}(0.05 - 0.7349)^2 = 0.2345$$

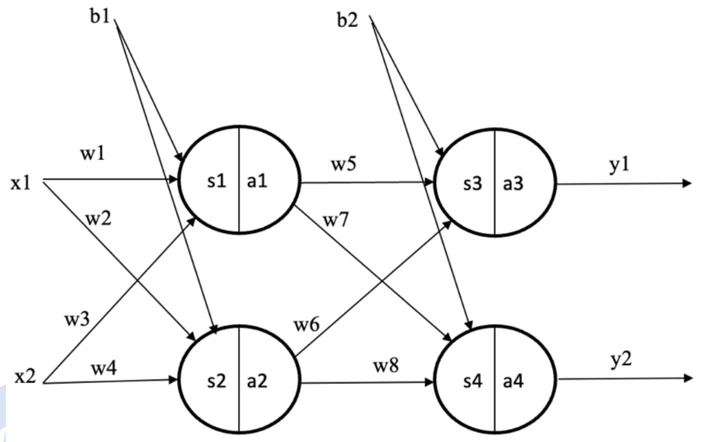
$$Error_2 = \frac{1}{2}(y_2 - \hat{y}_2)^2 = \frac{1}{2}(0.95 - 0.7795)^2 = 0.0145$$

$$Error_{Total} = Error_1 + Error_2 = 0.2345 + 0.0145 = 0.249$$

Example (cont…)

Backpropagation

Compute w5,w6, w7, and w8 (output layer weights)



$$\frac{\partial Error_{Total}}{\partial w5} = \frac{\partial Error_{Total}}{\partial a3} * \frac{\partial a3}{\partial s3} * \frac{\partial s3}{\partial w5}$$

$$\frac{\partial Error_{Total}}{\partial a3} = \frac{1}{2} * 2 * (y_1 - \hat{y}_1) * (-1)$$

$$= \hat{y}_1 - y_1 \; or \; a3 - y_1 = 0.7349 - 0.05 = 0.6849$$

$$\frac{\partial a3}{\partial s3} = a3(1 - a3) = 0.7349 * (1 - 0.7349) = 0.1948$$

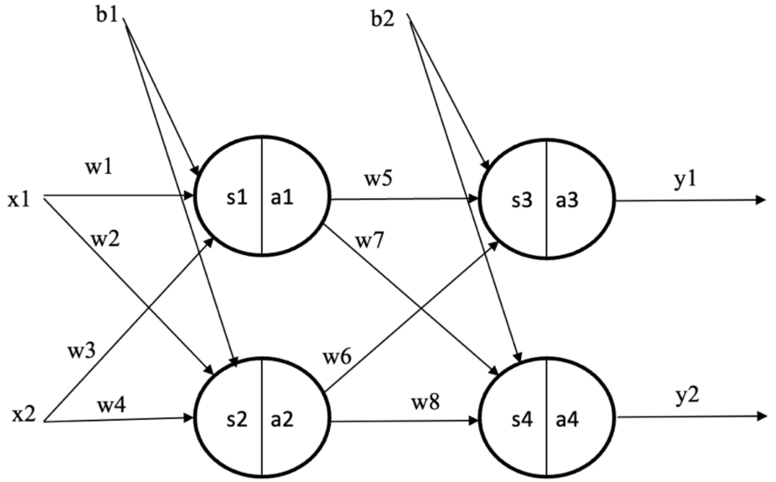
$$\frac{\partial s3}{\partial w5} = a1 = 0.601$$

$$\frac{\partial Error_{Total}}{\partial w5} = 0.6849 * 0.1948 * 0.601 = 0.080$$

Now update w5

$$w5 = w5 - \eta \frac{\partial Error_{Total}}{\partial w5} = 0.5 - 0.6 * 0.080 = 0.4518$$

Example (cont…)



x1 = 0.1
x2 = 0.5
y1 = 0.05
y2 = 0.95
w1 = 0.1
w2 = 0.2
w3 = 0.3
w4 = 0.4
w5 = 0.5
w6 = 0.6
w7 = 0.7
w8 = 0.8
b1 = 0.25
b2 = 0.35

Compute w6

$$\frac{\partial Error_{Total}}{\partial w6} = \frac{\partial Error_{Total}}{\partial a3} * \frac{\partial a3}{\partial s3} * \frac{\partial s3}{\partial w6}$$

$$\frac{\partial s3}{\partial w6} = a2$$

$$\frac{\partial Error_{Total}}{\partial w6} = 0.6849 * 0.1948 * 0.615 = 0.082$$

Now update w6

$$w6 = w6 - \eta \frac{\partial Error_{Total}}{\partial w6} = 0.6 - 0.6 * 0.082 = 0.5507$$

Similarly compute w7, and w8

$$\frac{\partial Error_{Total}}{\partial w7} = \frac{\partial Error_{Total}}{\partial a4} * \frac{\partial a4}{\partial s4} * \frac{\partial s4}{\partial w7} = -0.0176$$

$$w7 = w7 - \eta \frac{\partial Error_{Total}}{\partial w7} = 0.7105$$

$$\frac{\partial Error_{Total}}{\partial w8} = \frac{\partial Error_{Total}}{\partial a4} * \frac{\partial a4}{\partial s4} * \frac{\partial s4}{\partial w8} = -0.018$$

$$w8 = w8 - \eta \frac{\partial Error_{Total}}{\partial w8} = 0.8108$$

Example (cont…)

Compute w1,w2, w3, and w4 (Hidden layer weights)

$$\frac{\partial E_1}{\partial w1} = \frac{\partial E_1}{\partial a3} * \frac{\partial a3}{\partial s3} * \frac{\partial s3}{\partial a1} * \frac{\partial a1}{\partial s1} * \frac{\partial s1}{\partial w1} = 0.00159$$

$$\frac{\partial E_2}{\partial w1} = \frac{\partial E_2}{\partial a4} * \frac{\partial a4}{\partial s4} * \frac{\partial s4}{\partial a1} * \frac{\partial a1}{\partial s1} * \frac{\partial s1}{\partial w1} = -0.00049$$

Now

$$\frac{\partial E_{Total}}{\partial w1} = \frac{\partial E_1}{\partial w1} + \frac{\partial E_2}{\partial w1}$$

Therefore

$$\frac{\partial E_{Total}}{\partial w1} = 0.00159 + (-0.00049) = 0.00110$$

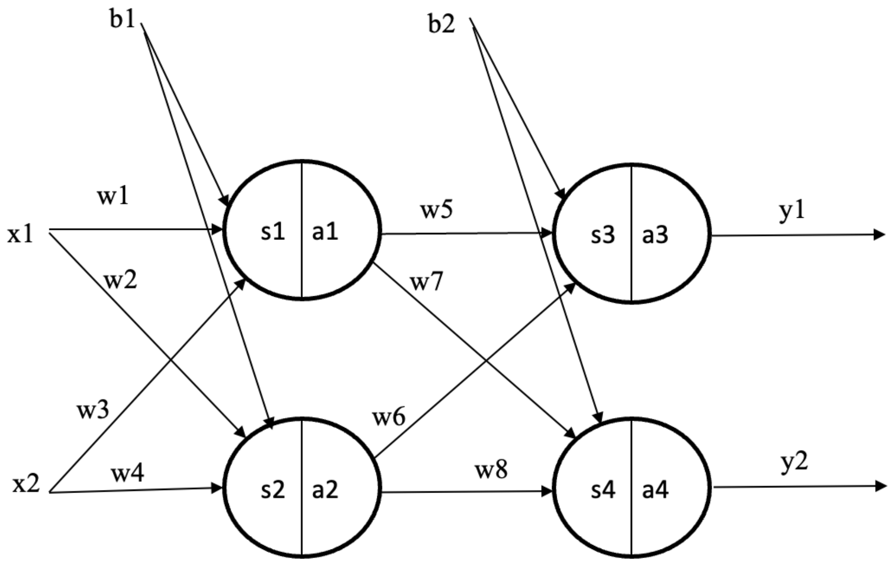
$$w1 = w1 - \eta \frac{\partial E_{Total}}{\partial w1}$$

$$= 0.1 - 0.6 * 0.00110 = 0.0993$$

Similarly w2 = 0.19919

w3 = 0.2966

w4 = 0.3959



x1 = 0.1
x2 = 0.5
y1 = 0.05
y2 = 0.95
w1 = 0.1
w2 = 0.2
w3 = 0.3
w4 = 0.4
w5 = 0.5
w6 = 0.6
w7 = 0.7
w8 = 0.8
b1 = 0.25
b2 = 0.35