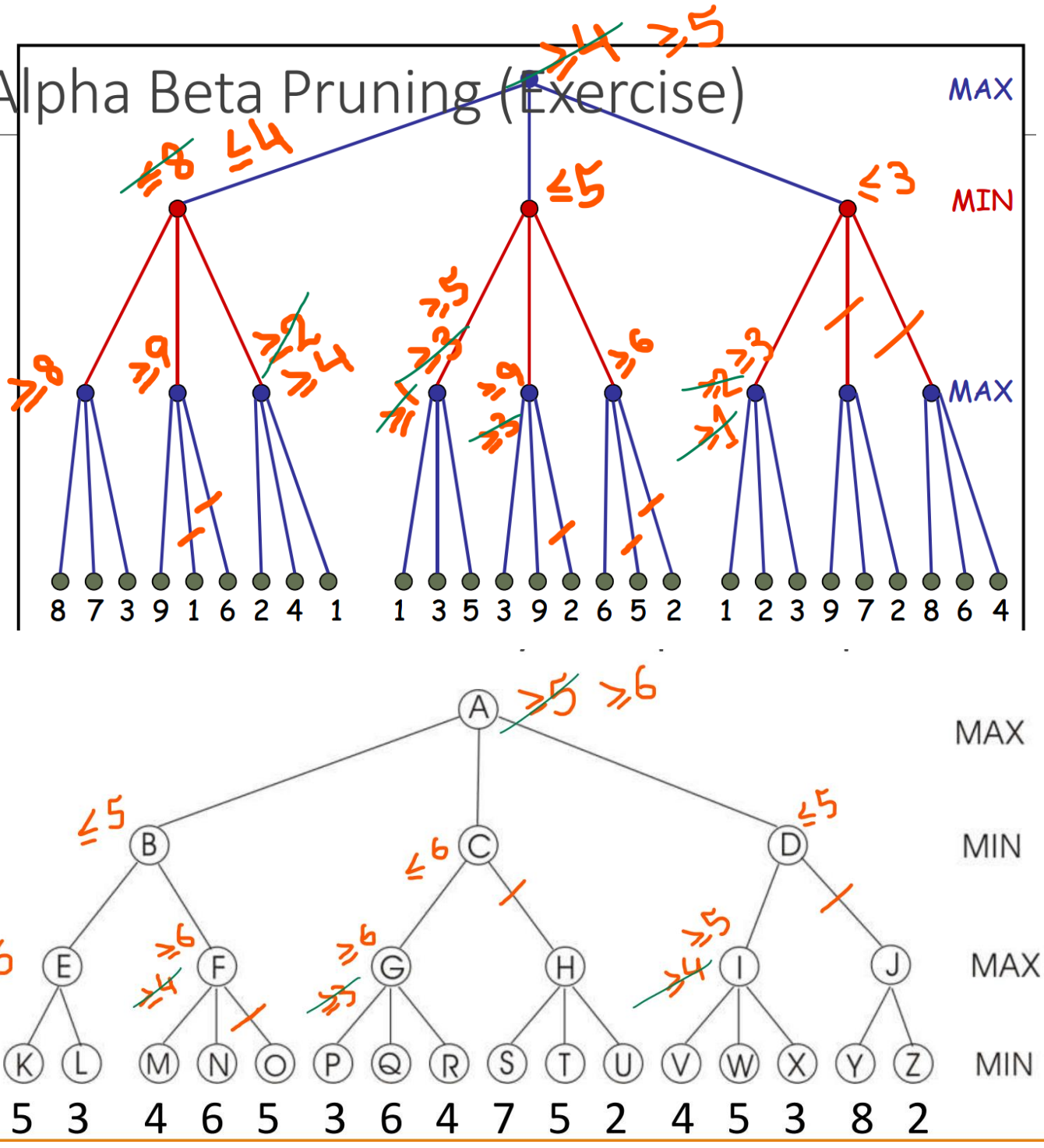
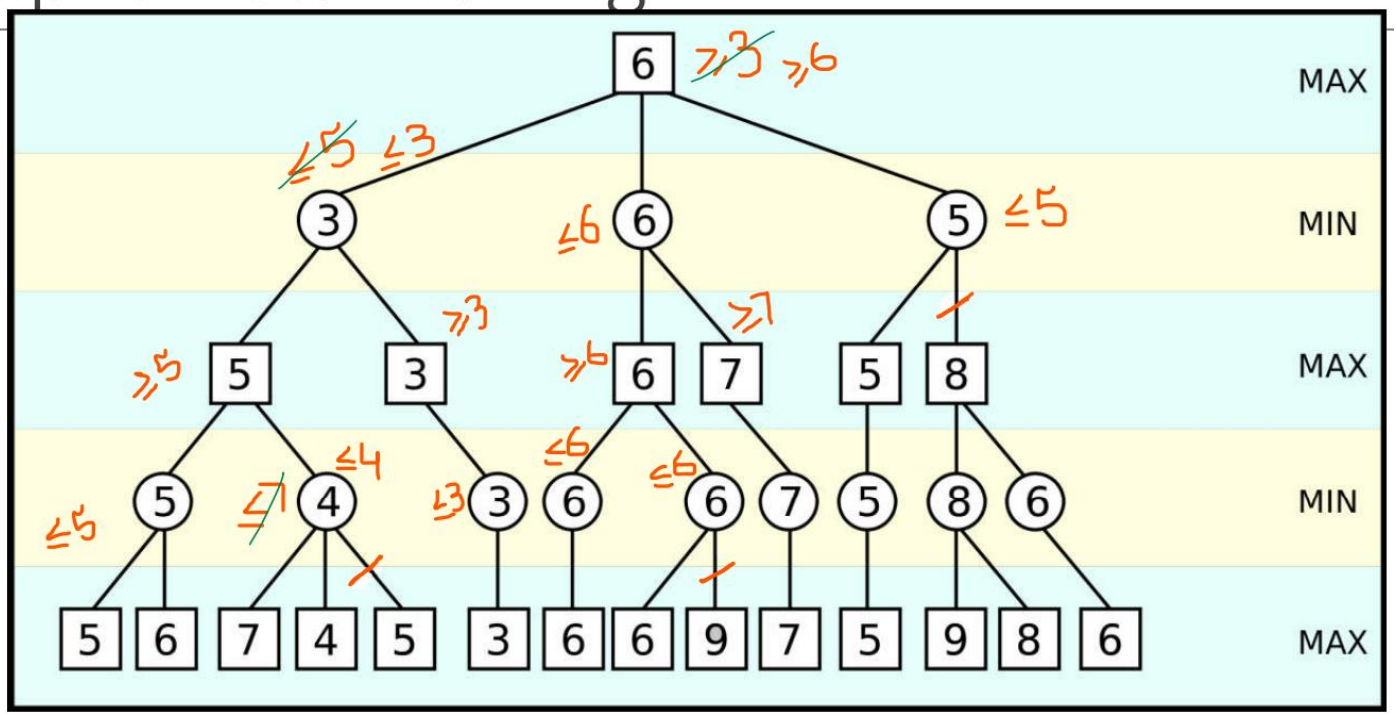
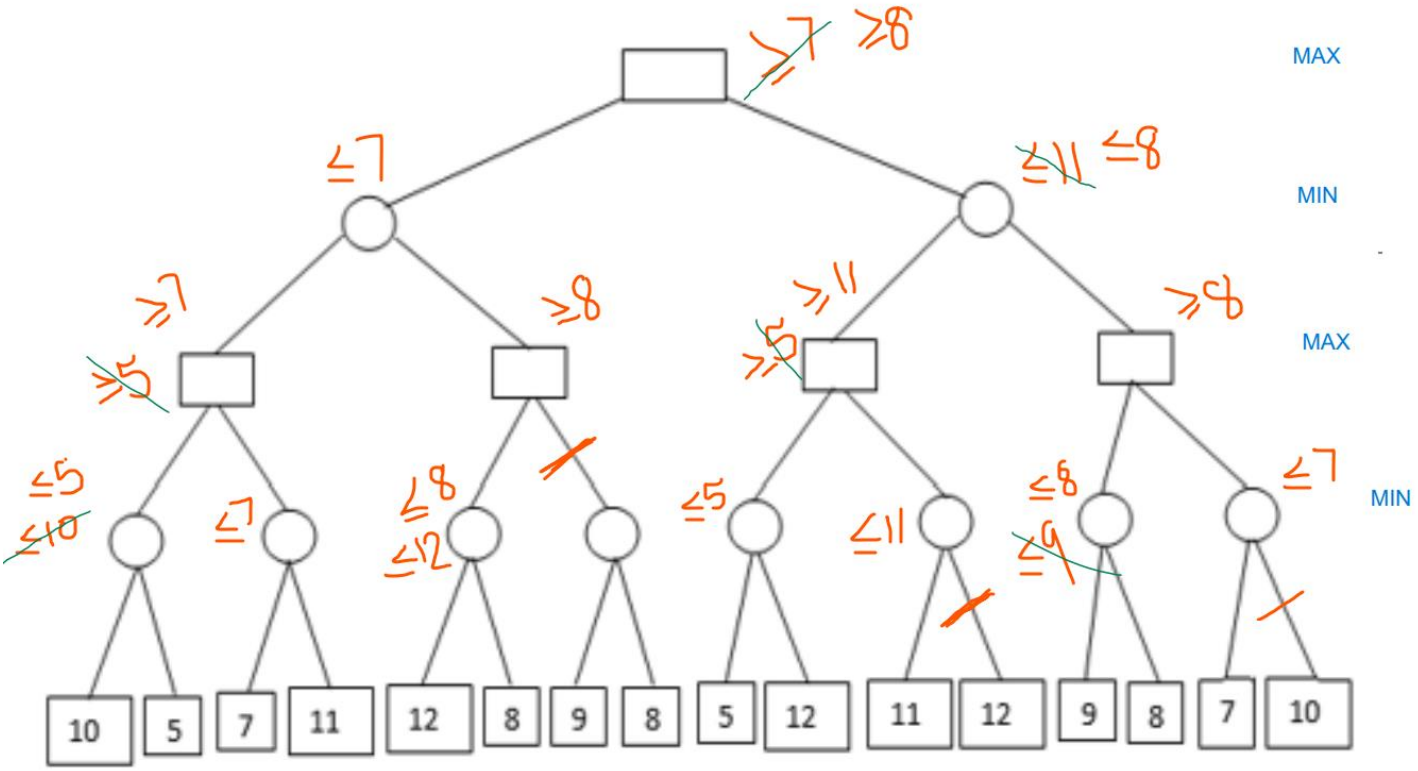
Alpha Beta Pruning (Exercise)



S will get checked. T and U will get pruned

Alpha Beta Pruning





Hill Climb:

Your next step should be better than your current step.

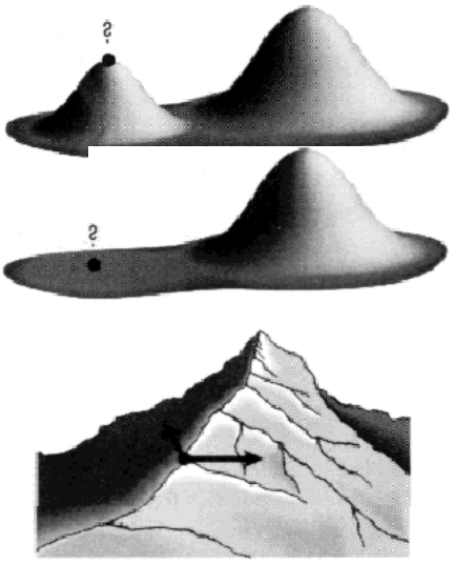
1. First Limitation is Local Maxima. When you reach local maxima then you need to come down and go up again to reach the global maxima but in this algo, your next step should always be better than your current step so that means we can't go down and we are stuck.
2. Second Limitation is Plateau. This happens when we are at a flat area and all the values of the successors are equal so the algo will not think that the next step is better than my current step and it will be stuck again.
3. Third Limitation is Ridge

Hill-Climbing Search

ocal maxima: a local maximum is a peak that is higher than each f its neighboring states, but lower than the global maximum. ill-climbing algorithms that reach the vicinity of a local maximum ill be drawn upwards towards the peak, but will then be stuck ith nowhere else to go.

lateaux: a plateau is an area of the state space landscape where ie evaluation function is flat. It can be a flat local maximum, from hich no uphill exit exists, or a **shoulder,** from which it is possible to ake progress.

idges: Ridges result in a sequence of local maxima that is very ifficult for greedy algorithms to navigate. (the search direction is ot towards the top but theside)



Criteria	Stochastic Hill Climbing	First-Choice Hill Climbing	Random Restart Hill Climbing
Selection of Neighbors	Randomly chooses a neighbor at each step	Selects the first better neighbor found	Examines multiple neighbors but selects only one
Determinism	Non-deterministic	Deterministic	Non-deterministic (due to random restarts)
Efficiency	Less efficient compared to deterministic approaches	Efficient when there are many neighbors to consider	Can be computationally expensive due to multiple restarts
Escape from Local Optima	Less prone to getting stuck in local optima due to randomness	Can get stuck in local optima if the first better neighbor is not the global optimum	Effective in escaping from local optima by exploring different regions
Computational Complexity	Moderate	Low to Moderate	Moderate to High
Application	Suitable for noisy or uncertain environments	Suitable for problems with many neighbors	Suitable for complex search spaces with multiple local optima

Genetic Algorithm:

$$Prob = \frac{f(x)}{\sum f(x)} \qquad Expected\ Count = \frac{f(x_i)}{Avg(\sum f(x))}$$

String No.	Initial Population (Randomly Selected)	X Value	Fitness $f(x) = x^2$	Prob	% Prob	Expected Count	Actual Count
1	01100	12	144	0.1247	12.47	0.4987	1
2	11001	25	625	0.5411	54.11	2.1645	2
3	00101	5	25	0.0216	2.16	0.0866	0
4	10011	19	181	0.3126	31.26	1.2502	1
Sum			1155	1.0	100	4	4
Average			288.75	0.25	25	1	1
Maximum			625	0.5411	54.11	2.1645	2

String No.	Mating Pool	Crossover Point	Offspring after crossover	X Value	Fitness $f(x) = x^2$
1	01100	4	01101	13	169
2	11001		11000	24	576
3	11001	2	11011	27	729
4	10011		10001	17	289
Sum					1763
Average					440.75
Maximum					729

String No.	Offspring after crossover	Mutation Chromosome for flipping	Offspring after mutation	X Value	Fitness $f(x) = x^2$
1	01101	10000	11101	29	841
2	11000	00000	11000	24	576
3	11011	00000	11011	27	729
4	10001	00101	10100	20	400
Sum					2546
Average					636.5
Maximum					841

Design a Hebb Net to implement logical AND function

Initially the weights and bias are set to zero, i.e.,

$$w_1 = w_2 = b = 0$$

First input [x1 x2 b] = [111] and target = 1 [i.e., y = 1]:

Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

Setting the initial weights as old weights and applying the Hebb rule, we get

$$w_i(\text{new}) = w_i(\text{old}) + \Delta w_i$$

$$\Delta w_i = x_i y$$

$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$

$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$

$$\Delta b = y = 1$$

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0 + 1 = 1$$

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0 + 1 = 1$$

$$b(\text{new}) = b(\text{old}) + \Delta b = 0 + 1 = 1$$

Design a Hebb Net to implement logical AND function

- Second input [x1 x2 b] = [1 –1 1] and y = –1:
- The weight change here is

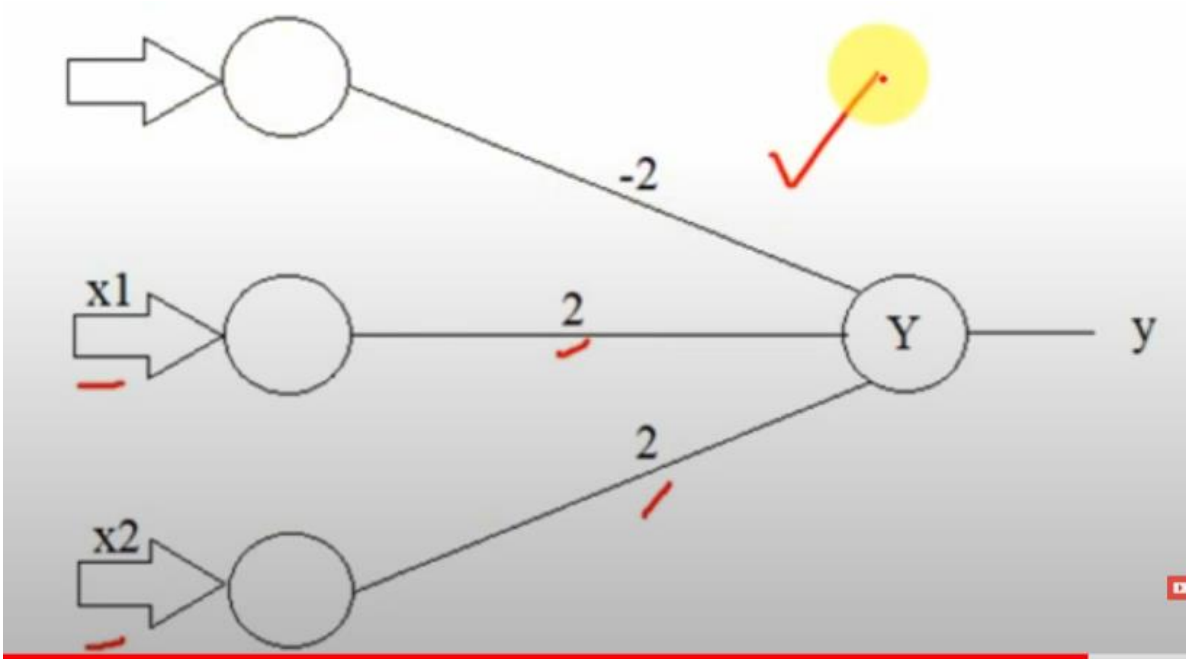
$$\Delta w_1 = x_1 y = 1 \times -1 = -1$$
$$\Delta w_2 = x_2 y = -1 \times -1 = 1$$
$$\Delta b = y = -1$$

- The new weights here are

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 1 - 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 1 + 1 = 2$$
$$b(\text{new}) = b(\text{old}) + \Delta b = 1 - 1 = 0$$

Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

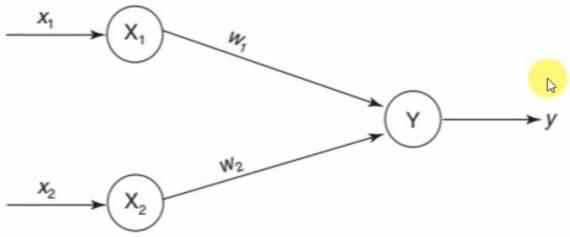
Inputs			Weight changes				Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0	w_1 0	b 0)
1	1	1	1	1	1	1	1	1	1
1	−1	1	−1	−1	1	−1	0	2	0
−1	1	1	−1	1	−1	−1	1	1	−1
−1	−1	1	−1	1	1	−1	2	2	−2



assign the lowest rank to the one who has the highest fitness
250 has highest fitness (0.250) so we assign last rank to it

QUESTION 2: Suppose there are 10 chromosomes with finesses as shown in table. What will be the selection probability according to the proportionate and linear rank selection methods? **(5 + 10)**

Chromosome No.	Fitness *fi*	Proportionate	Linear Rank	Linear Rank Calculations		
		$p_i = f_i / \sum_{j=1}^{n} f_j$		Sort	$nf_i = (P - r_i) + 1$	$np_i = nf_i / \sum_{j=1}^{n} nf_j$
A	50	0.05	0.055	250	10	0.182
B	25	0.025	0.036	140	9	0.164
C	25	0.025	0.018	125	8	0.145
D	100	0.1	0.109	110	7	0.127
E	75	0.075	0.073	100	6	0.109
F	125	0.125	0.145	100	5	0.091
G	250	0.250	0.182	75	4	0.073
H	110	0.110	0.127	50	3	0.055
I	140	0.140	0.164	25	2	0.036
J	100	0.1	0.091	25	1	0.018
Total	1000	1	1	1000	55	1

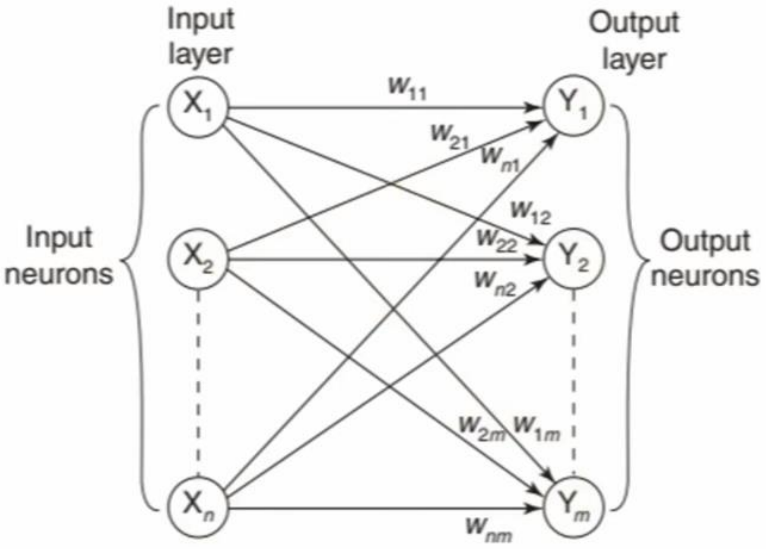


- Input neurons X1 and X2 are connected to the output neuron Y, over a weighted interconnection links (W1 and W2).
- For the above simple neuron net architecture, the net input has to be calculated in the following way: $y_{in} = x_1 w_1 + x_2 w_2$ $y = f(y_{in})$

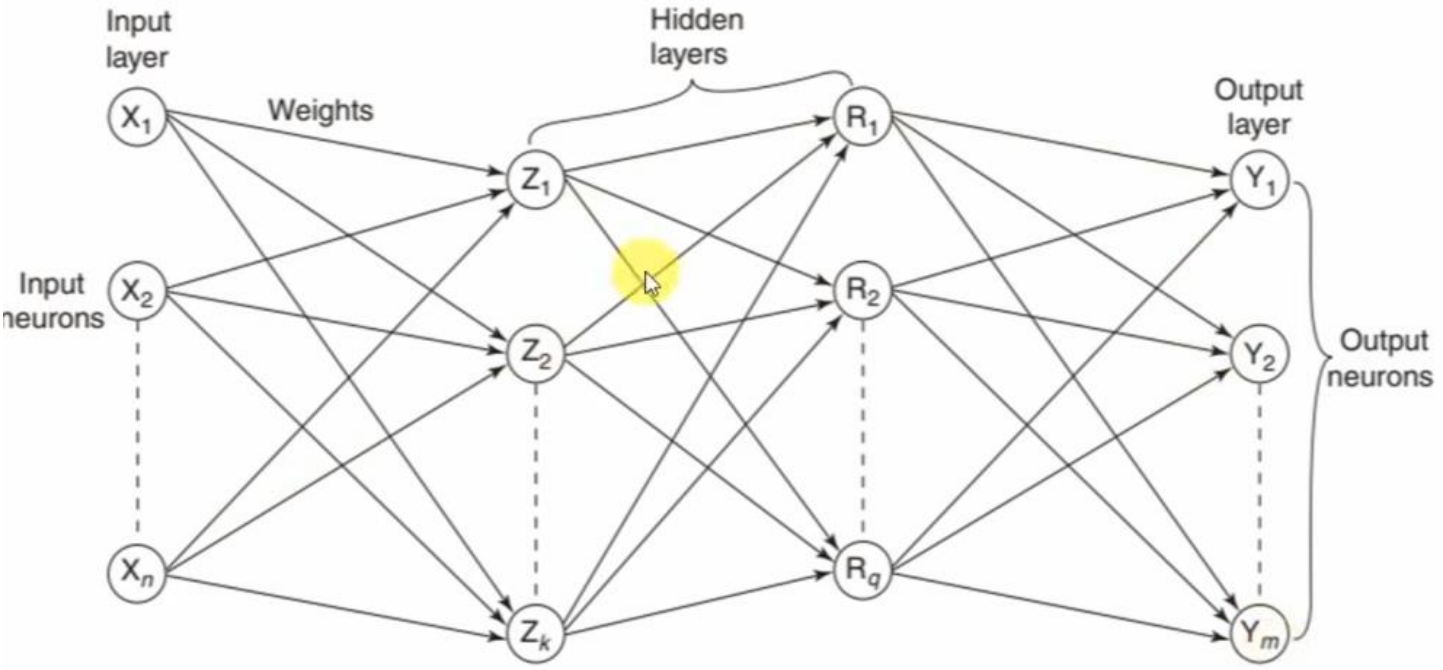
There exist five basic types of neuron connection architectures. They are:

1. single-layer feed-forward network
2. multilayer feed-forward network
3. single node with its own feedback
4. single-layer recurrent network
5. multilayer recurrent network.

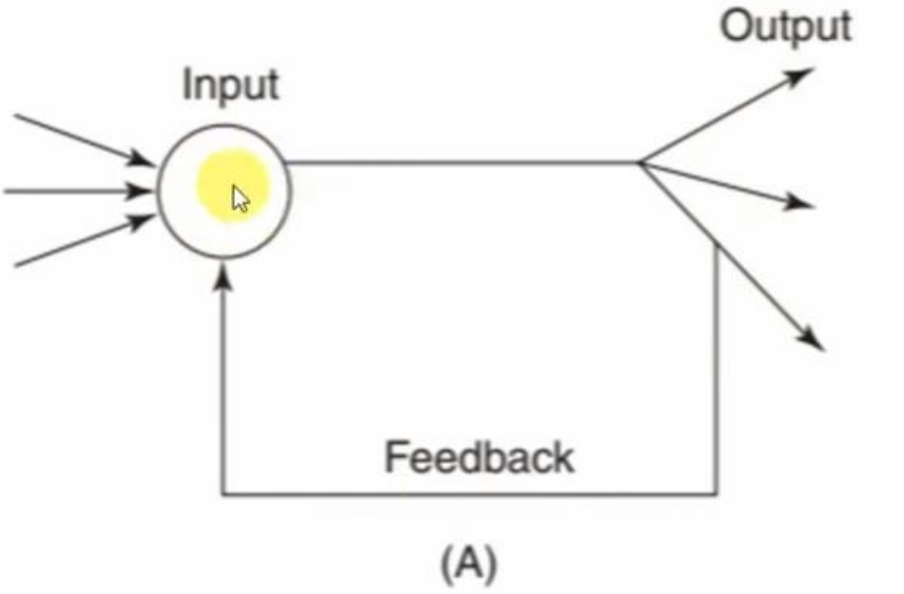
1. Connections



Single-layer feed-forward network.



Multilayer feed-forward network.

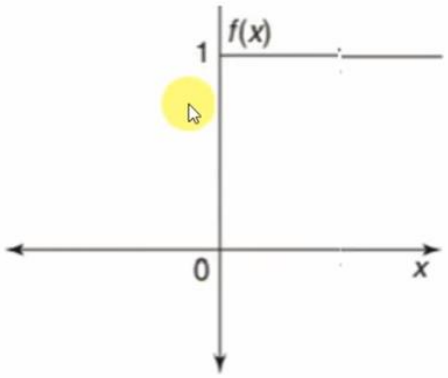


(A)

Single node with own feedback.

2. Binary step function: This function can be defined as

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



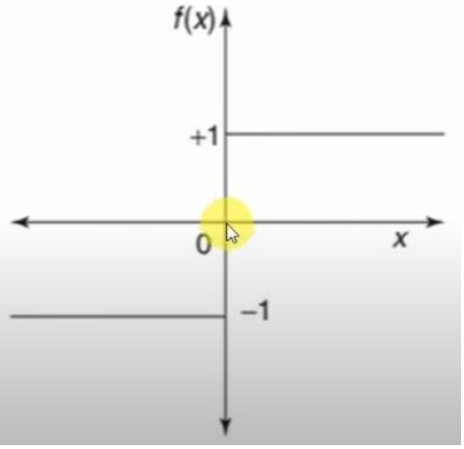
where q represents the threshold value.

This function is most widely used in single-layer nets

to convert the net input to an output that is a binary (1 or 0).

3. Bipolar step function: This function can be defined as

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



where θ represents the threshold value.

This function is also used in single-layer nets to convert

the net input to an output that is bipolar (+1 or –1).

5.1 Binary sigmoid function: It is also termed as logistic sigmoid function or unipolar sigmoid function. It can be defined as

$$f(x) = \frac{1}{1+e^{-\lambda x}}$$

where λ is the steepness parameter. The derivative of this function is

$$f'(x) = \lambda f(x)[1-f(x)]$$

Here the range of the sigmoid function is from 0 to 1.

Sigmoid Activation Function Solved Example

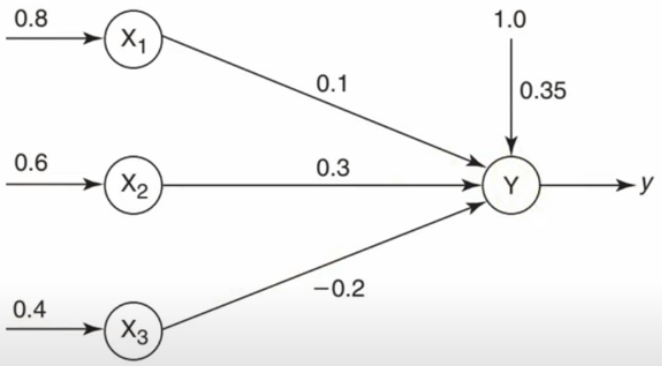
- The net input to the output neuron is

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

$$= b + x_1 w_1 + x_2 w_2 + x_3 w_3$$

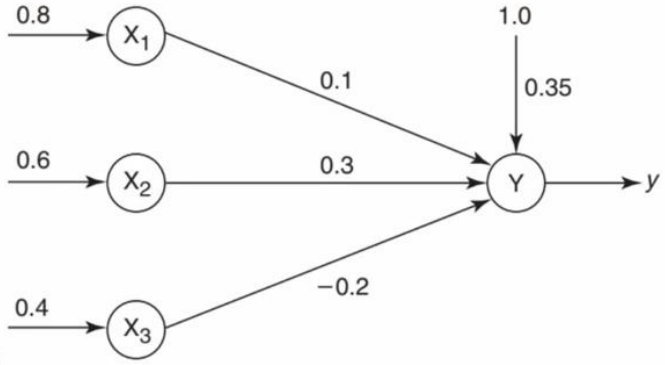
$$= 0.35 + 0.8 \times 0.1 + 0.6 \times 0.3 + 0.4 \times (-0.2)$$

$$= 0.35 + 0.08 + 0.18 - 0.08 = 0.53$$



- Binary Sigmoid Activation Function

$$y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-0.53}} = 0.625$$

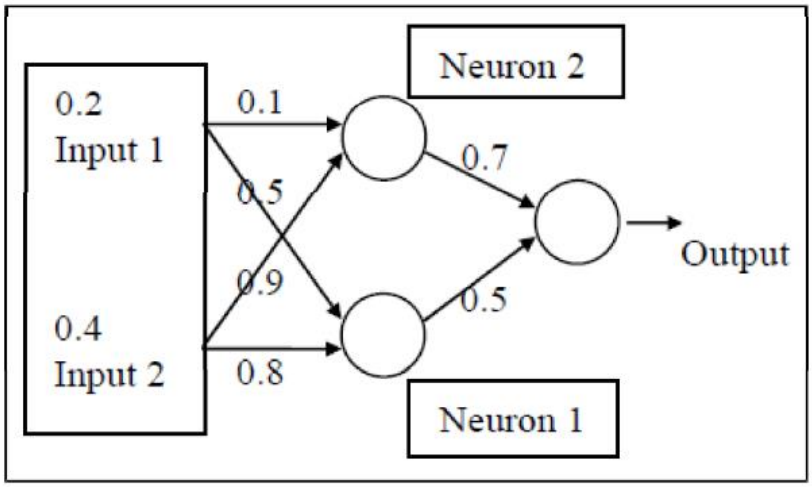


- Bipolar Sigmoid Activation Function

$$y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1 = \frac{2}{1+e^{-0.53}} - 1 = 0.259$$

Example of multilayer ANN

- Calculate the output from this network assuming a Sigmoid Squashing Function.



$$\text{Input to neuron 1} = (0.2 \times 0.5) + (0.4 \times 0.8) = 0.42. \text{ Output} = \frac{1}{1+e^{-0.42}} = 0.603$$

$$\text{Input to neuron 2} = (0.2 \times 0.1) + (0.4 \times 0.9) = 0.38. \text{ Output} = \frac{1}{1+e^{-0.38}} = 0.594$$

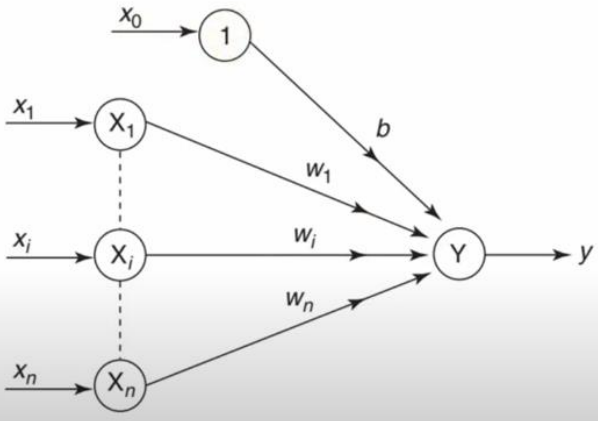
$$\text{Input to final neuron} = (0.594 \times 0.7) + (0.603 \times 0.5) = 0.717.$$

$$\text{Final Output} = \frac{1}{1+e^{-0.717}} = 0.672$$

Perceptron Training Algorithm - Single Output Class

- Initialize the weights and the bias. Also initialize the learning rate ά (0< ά ≤ 1).
- Until the final stopping condition is false.
  - for each training pair indicated by s:t.
    - Set each input unit i = 1 to n: $x_i = s_i$
    - Calculate the output of the network.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i \qquad y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$



- Weight and bias adjustment:

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$

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

else we have

$$w_i(\text{new}) = w_i(\text{old})$$

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

Perceptron Learning Rule

- In case of the perceptron learning rule, the learning signal is the difference between the calculated output and actual (target) output of a neuron.
- The output "y'' is obtained on the basis of the net input calculated and activation function being applied over the net input.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

If $y \neq t$, then

$$w(\text{new}) = w(\text{old}) + \alpha t x \quad (\alpha - \text{learning rate})$$

else, we have

$$w(\text{new}) = w(\text{old})$$

- Weights are updated using the formula

AND function using Perceptron Rule Solved Example

$$y_{in} = b + x_1 w_1 + x_2 w_2$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1; \quad \Delta w_2 = \alpha t x_2; \quad \Delta b = \alpha t$$

Input x_1	Input x_2	Target (t)	Net input (y_{in})	Calculated output (y)	Weight changes Δw_1	Weight changes Δw_2	Weight changes Δb	Weights w_1	Weights w_2	Weights b
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1
1	−1	−1	1	1	−1	1	−1	0	2	0
−1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	−1	−3	−1	0	0	0	1	1	−1

AND function using Perceptron Rule Solved Example

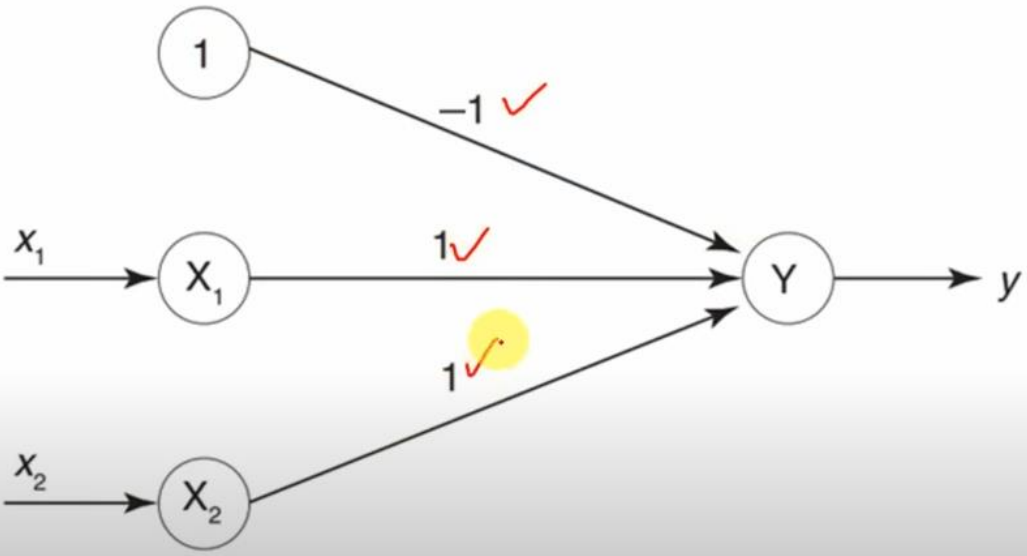
$$y_{in} = b + x_1 w_1 + x_2 w_2$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1; \quad \Delta w_2 = \alpha t x_2; \quad \Delta b = \alpha t$$

Input x_1	Input x_2	Target (t)	Net input (y_{in})	Calculated output (y)	Weight changes Δw_1	Weight changes Δw_2	Weight changes Δb	Weights w_1	Weights w_2	Weights b
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1
1	−1	−1	1	1	−1	1	−1	0	2	0
−1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2										
1	1	1	1	1	0	0	0	1	1	−1
1	−1	−1	−1	−1	0	0	0	1	1	−1
−1	1	−1	−1	−1	0	0	0	1	1	−1
−1	−1	−1	−3	−1	0	0	0	1	1	−1

AND function using Perceptron Rule Solved Example



Perceptron Network (Rule) Solved Example

- Find the weights required to perform the following classification using perceptron network.
- The vectors (1, 1, 1, 1) and (- 1, 1 - 1, - 1) are belonging to the class 1, vectors (1, 1, 1, - 1) and (1, - 1, - 1, 1) are belonging to the class -1.
- Assume learning rate as 1
- and Initial weights as 0.

Input					Target
x_1	x_2	x_3	x_4	b	(t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1

Perceptron Network (Rule) Solved Example

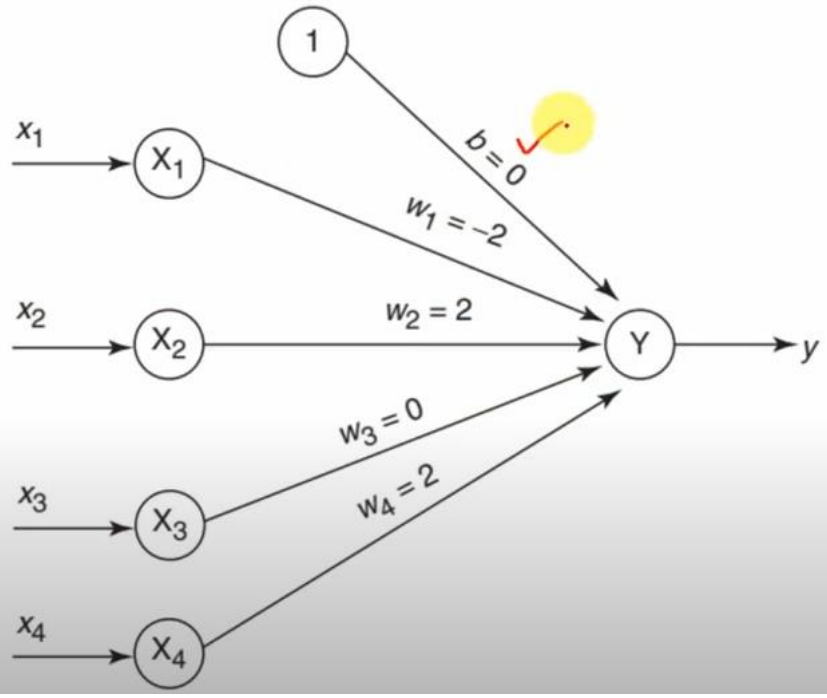
$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta w_3 = \alpha t x_3;$$
$$\Delta w_4 = \alpha t x_4;$$
$$\Delta b = \alpha t$$

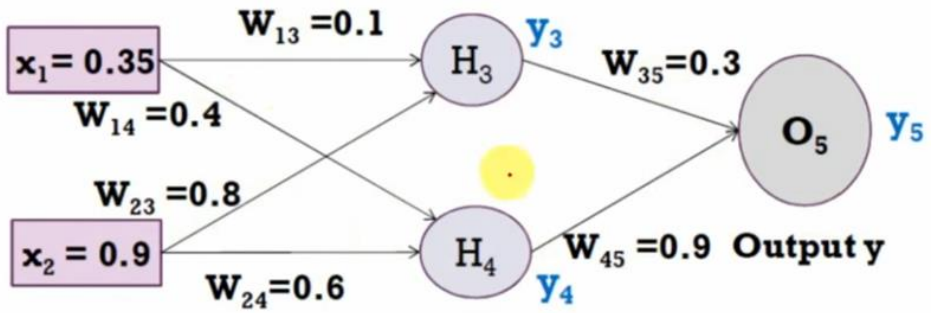
	Inputs				Target	Net input	output	Weight changes					Weights				
	x_1	x_2	x_3	x_4	t	y_{in}	y	Δw_1	Δw_2	Δw_3	Δw_4	Δb	w_1 (0	w_2 0	w_3 0	w_4 0	b 0)
EPOCH-1																	
(1	1	1	1		1	0	0	1	1	1	1	1	1	1	1	1	1
(-1	1	-1	-1		1	-1	-1	-1	1	-1	-1	1	0	2	0	0	2
(1	1	1	-1		-1	4	1	-1	-1	-1	1	-1	-1	1	-1	1	1
(1	-1	-1	1		-1	1	1	-1	1	1	-1	-1	-2	2	0	0	0
EPOCH-2																	
(1	1	1	1		1	0	0	1	1	1	1	1	-1	3	1	1	1
(-1	1	-1	-1		1	3	1	0	0	0	0	0	-1	3	1	1	1
(1	1	1	-1		-1	4	1	-1	-1	-1	1	-1	-2	2	0	2	0
(1	-1	-1	1		-1	-2	-1	0	0	0	0	0	-2	2	0	2	0
EPOCH-3																	
(1	1	1	1		1	2	1	0	0	0	0	0	-2	2	0	2	0
(-1	1	-1	-1		1	2	1	0	0	0	0	0	-2	2	0	2	0
(1	1	1	-1		-1	-2	-1	0	0	0	0	0	-2	2	0	2	0
(1	-1	-1	1		-1	-2	-1	0	0	0	0	0	-2	2	0	2	0

Perceptron Network (Rule) Solved Example

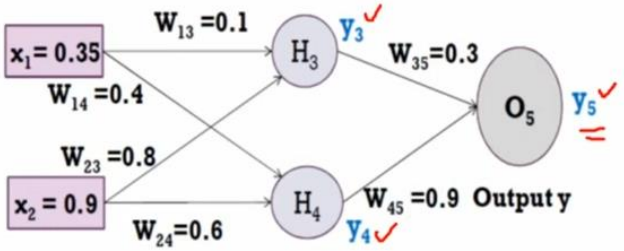


Back Propagation Solved Example - 1

Assume that the neurons have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 0.5 and learning rate is 1. Perform another forward pass.



Back Propagation Solved Example - 1



- Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$
$= (0.1 * 0.35) + (0.8 * 0.9) = 0.755$
$y_3 = f(a_1) = 1/(1 + e^{-0.755}) = 0.68$

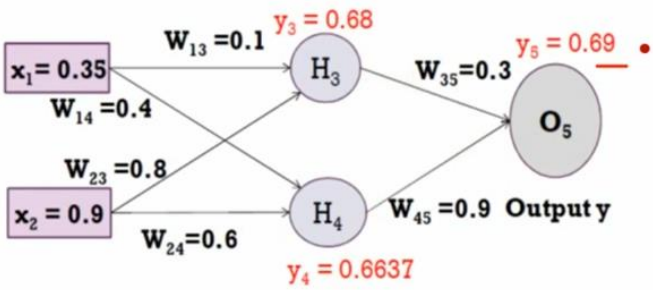
$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$
$= (0.4 * 0.35) + (0.6 * 0.9) = 0.68$
$y_4 = f(a_2) = 1/(1 + e^{-0.68}) = 0.6637$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$
$= (0.3 * 0.68) + (0.9 * 0.6637) = 0.801$
$y_5 = f(a_3) = 1/(1 + e^{-0.801}) = \mathbf{0.69}$ (Network Output)

$\text{Error} = y_{target} - y_5 = -0.19$

0.5 - 0.69

Back Propagation Solved Example - 1



Backward Pass: Compute δ3, δ4 and δ5.

For output unit:

$\delta_5 = y(1-y)\,(y_{target} - y)$

$= 0.69*(1-0.69)*(0.5-0.69) = -0.0406$

For hidden unit:

$\delta_3 = y_3(1-y_3)\, w_{35} * \delta_5$

$= 0.68*(1 - 0.68)*(0.3 * -0.0406) = -0.00265$

$\Delta w_{ji} = \eta \delta_j o_i$

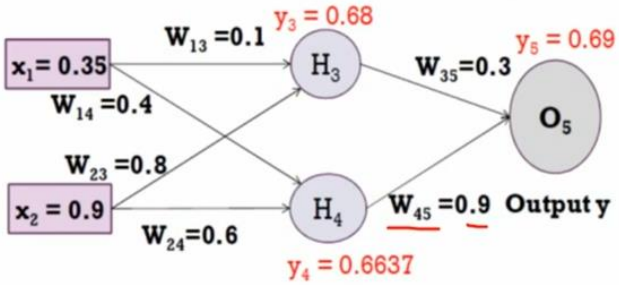
$\delta_j = o_j(1-o_j)(t_j - o_j)$ if j is an output unit

$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$ if j is a hidden unit

$\delta_4 = y_4(1-y_4)w_{45} * \delta_5$

$= 0.6637*(1 - 0.6637)* (0.9 * -0.0406) = -0.0082$

Back Propagation Solved Example - 1



- Backward Pass: Compute δ3, δ4 and δ5.

For output unit:

$\delta_5 = y(1-y)\,(y_{target} - y)$

$= 0.69*(1-0.69)*(0.5-0.69) = -0.0406$

For hidden unit:

$\delta_3 = y_3(1-y_3)\, w_{35} * \delta_5$

$= 0.68*(1 - 0.68)*(0.3 * -0.0406) = -0.00265$

$\delta_4 = y_4(1-y_4) w_{45} * \delta_5$

$= 0.6637*(1 - 0.6637)* (0.9 * -0.0406) = -0.0082$

Compute new weights

$$\Delta w_{ji} = \eta \delta_j o_i$$

$\Delta w_{45} = \eta\, \delta_5\, y_4 = 1 * -0.0406 * 0.6637 = -0.0269$

w_{45} (new) $= \Delta w_{45} + w_{45}$(old) $= -0.0269 + (0.9) = 0.8731$

$\Delta w_{14} = \eta\, \delta_4\, x_1 = 1 * -0.0082 * 0.35 = -0.00287$

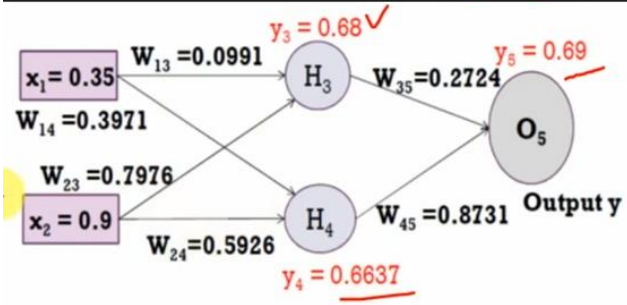
w_{14} (new) $= \Delta w_{14} + w_{14}$(old) $= -0.00287 + 0.4 = 0.3971$

Back Propagation Solved Example - 1

- Similarly, update all other weights

i	j	w_{ij}	δ_j	x_i	η	Updated w_{ij}
1	3	0.1	-0.00265	0.35	1	0.0991
2	3	0.8	-0.00265	0.9	1	0.7976
1	4	0.4	-0.0082	0.35	1	0.3971
2	4	0.6	-0.0082	0.9	1	0.5926
3	5	0.3	-0.0406	0.68	1	0.2724
4	5	0.9	-0.0406	0.6637	1	0.8731

Back Propagation Solved Example - 1



$$\text{Error} = y_{target} - y_5 = -0.182$$

- Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$
$= (0.0991 * 0.35) + (0.7976 * 0.9) = 0.7525$
$y_3 = f(a_1) = 1/ (1 + e^{-0.7525}) = 0.6797$

$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$
$= (0.3971 * 0.35) + (0.5926 * 0.9) = 0.6723$
$y_4 = f(a_2) = 1/ (1 + e^{-0.6723}) = 0.6620$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$
$= (0.2724 * 0.6797) + (0.8731 * 0.6620) = 0.7631$
$y_5 = f(a_3) = 1/ (1 + e^{-0.7631}) =$ **0.6820** (Network Output)