DLT UNIT 1

deep learning (Anna University)

UNIT - 1.

Neural Network:

Neural network is a computational model inspired by structure and functioning of biological neural n/ws, such as human brain. It is a n/w of interconnected artificial neurons, also known as nodes or units organized into layers. Each neurons receives input signals, processes them through activation function and produces an output signal that may serve as input to other neurons.

Neural n/ws are designed to learn & recognize patterns, relationships, and features with in input data.

They accomplish this through a process called training, where the n/w adjusts its internal parameters known as weights, based on example data and specified objective/target.

The goal is to minimize diff b/w n/w's output and desired output, thus n/w allowing to make accurate predictions / classifications for new, unseen data.

Types Of Neural N/ws.

1. Feed forward neural n/ws
2. Recurrent neural n/ws
3. Convolutional neural n/ws and more.

Each types has specific architecture & characteristics suited for diff tasks and data types.

Neural n/w applications

* Image, Speech Recognition.
* Natural language Processing
* Recommender Systems.
*

Characteristics of Neural Network:

Neural n/w Possess several key characteristics to their effectiveness and versatility in solving a wide range of problems.

Here are some important characteristics of neural networks.

1. Distributed Processing: Operate through Parallel & distributed Processing. Information is processed simultaneously across multiple neurons or nodes allowing for efficient computation and ability to handle large amount of data in Parallel.

2. Non-linearity: use nonlinear activation functions sigmoid or rectified Linear Unit (ReLU) - which introduce non-linearity into the n/w. This enables neural n/w to learn and model complex relationships in data, making them capable of solving problems that involve non-linear Patterns.

3. Adaptability and learning: n/w have ability to learn and adapt to i/p data. neural n/w: adjust their internal parameters (weights) based on example data and an objective function. This learning process allows them to improve their performance over time, and make accurate predictions/classifications for new/unseen data.

4. Generalization: capability to generalize from the training data to make predictions on unseen data. By learning from a diverse set of data examples during training, neural n/w can capture underlying patterns & generalize them to new instances, enabling them to handle real-world scenarios.

5. Feature Extraction: can automatically learn relevant features from raw input data. Rather than requiring manual feature engineering, which can be time consuming and challenging, neural n/w can extract meaningful features directly from data, reducing need for explicit feature selection.

6. Fault tolerance & Robustness: exhibit a certain level of fault tolerance and robustness. Due to their distributed nature, ability to generalize, still provide reasonable o/ps even some neurons/connections are damaged/missing. This fault tolerance make them more resilient in dealing with noisy/incomplete data.

7. Scalability: can be scaled up/down to handle diff. complexities of problems and data sizes. By adjusting no. of layers, neurons, connections, neural n/w can be tailored to specific tasks and computational resources, allowing for flexibility in their application.

The historical development of Neural N/W principles.

This spans several decades and can be traced back to the early beginnings of computing and artificial intelligence. Here is a brief overview of the key milestones in the development of neural n/w principles.

1. McCulloch-Pitts Neuron (1943) - introduced mathematical model of an artificial neuron, which laid foundation for the concept of neural n/ws. Their model described binary threshold unit that received inputs and produced a binary output based on set of predefined rules.

2. Perceptron: (1957) Frank Rosenblatt developed the Perceptron. This consisted of single layer of artificial neurons could be trained to classify linearly separable patterns. It was capable of learning and adjusting its weights based on error-feedback - which marked important step.

3. BackPropagation algorithm (1974) - fundamental technique for training multi-layer neural n/ws → Paul Werbos. It enables neural n/w with multiple layers to learn from examples by propagating errors backward through n/w and adjusting the weights accordingly.

4. Connectionism and Parallel Distributed Processing (PDP) - 1980s
David Rumelhart, Jame Mcclelland, Geoffrey Hinton explored field of connectionism which emphasized power of parallel processing and distributed representation in neural n/w.

Their work on PDP led significant advancements in understanding learning algorithms and architecture for neural nws

5. Recurrent Neural Networks (RNN) 1986 Elman, Jordan introduced RNN. have connections → loops → information to be stored and propagated over time.

Valuable in tasks involving sequential data - speech recognition, language modeling and time series predictions.

6. Convolution Neural Nws (CNN) (1989) : Yann LeCun colleagues designed for analysing visual data - images.

CNN - hierarchical arrangements of layers and localized receptive fields to extract & learn features directly from raw input. revolutionizing computer vision task.

7. Deep Learning: Renaissance (2000s). Advances in computer power, availability of large scale datasets, refinements in training algorithms sparked resurgence of interest in neural nws. DL. characterized by use of deep neural nws with multiple hidden layers, has achieved remarkable success in image & speech recognition, natural language processing, autonomous vehicle.

8. Transfer Learning & Pretrained models (2010s) : TL. Powerful in DL. enabling the transfer of knowledge learned from task/domain to another.

Pre-trained models based on architectures - AlexNet, VGG, ResNet - trained on massive datasets, available to broader research, boost. Performance. allow neural nws.

These milestone highlight the literature progress made in understanding neural n/w principles including developments of diff. architectures, learning algorithms and applications.

Artificial Neural Networks (ANN):

Biological Neuron:

It refers to group of biological Nerve cells that are connected to one another.

A typical biological neural network is brain. The brain is composed of a no. of neurons that are interlinked to form a huge n/w..

Biological neuron has a cell body with a nucleus inside.

Biological Neuron	Artificial Neuron
Dendrite -	Inputs - are tree-like structures that designed to receive commn. from other cell
Axon -	Nodes -
Synapses -	weights -
Nucleus -	Output -

ANN Terminology:

1. Neuron/Node: It represents an artificial equivalent of a biological neuron. Neurons receives I/P signals, performs computations, and generate output signals.

2. I/P Layer: It is 1st layer of NN where I/P fed into the N/w. Each neuron in I/P layer → feature/attribute of i/p data.

3. Hidden Layer: It is intermediate layer b/w I/P & O/P layer. It performs computations and on the I/P data & pass the results to the next layer.

4. O/P Layer: It is final layer of NN that produces N/w's O/P. The no. of neurons in the O/P layer depends on the nature of the problem (eg. classification, regression)

5. Activation function: It defines the o/p of a neuron based on its i/p. common AF include sigmoid, tanh, ReLU, and softmax. AFs introduces non-linearity into the N/w. allowing it to model complex relationships.

6. Weight: Each connection ~~neuron~~ b/w neurons in diff. layers has a weight associated with it. weights → strength of the connection and play a crucial role in signal propagation and learning.

7. Bias: It is an additional term added to the weighted sum of I/Ps of a neuron. It allows shifting the AFs O/p and helps the N/w learn diff. features and make predictions.

8. Feedforward Neural N/w: The simplest form of ANN where info. flows only in one direction. from IP to O/P. There are no loops / feedback connections in this type of N/w.

9. Back Propagation: It is an algorithm used to train ANN by adjusting weights based on diff b/w N/w's O/P and desired O/P.

It calculates gradient of error with respect to each weight, enabling n/w to learn from the training data.

10. EPOch: refers a single Pass of the entire training data set through neural N/w. multiple epochs are typically required during training to improve the n/w's Performance.

11. Weighted Sum:

$$W.S = \sum_{i=1}^{n} x_i \times w_i + b_i$$

12. Loss function: diff. b/w predicted O/P and actual O/P. is measured by using loss function. The choice of Loss function depends on the type of P&Im being solved.

Ex:

Mean Squared Error - Regression tasks

Cross Entropy Loss - classification tasks.

The McCulloch-Pitts N/W:

It is the simplest N/W model.

Warren McCulloch - Walter Pitts - introduced in 1943.

The i/p $\{x_1, x_2, x_3, \ldots x_n\}$ given to a neuron and produces o/p y.

The o/p is binary. The neuron should either fire(1) or not fire(0) based on the i/p.

The processing unit is found in o/p layer. To combine i/p and give it to the processing unit, the i/p is summed. multiple i/p are combined using summation as follows -

$$\text{Net Input}, I = \sum_{i=1}^{n} x_i$$

The activation function takes net i/p as I/P & produces required o/p.

↑ Threshold function (binary step function).

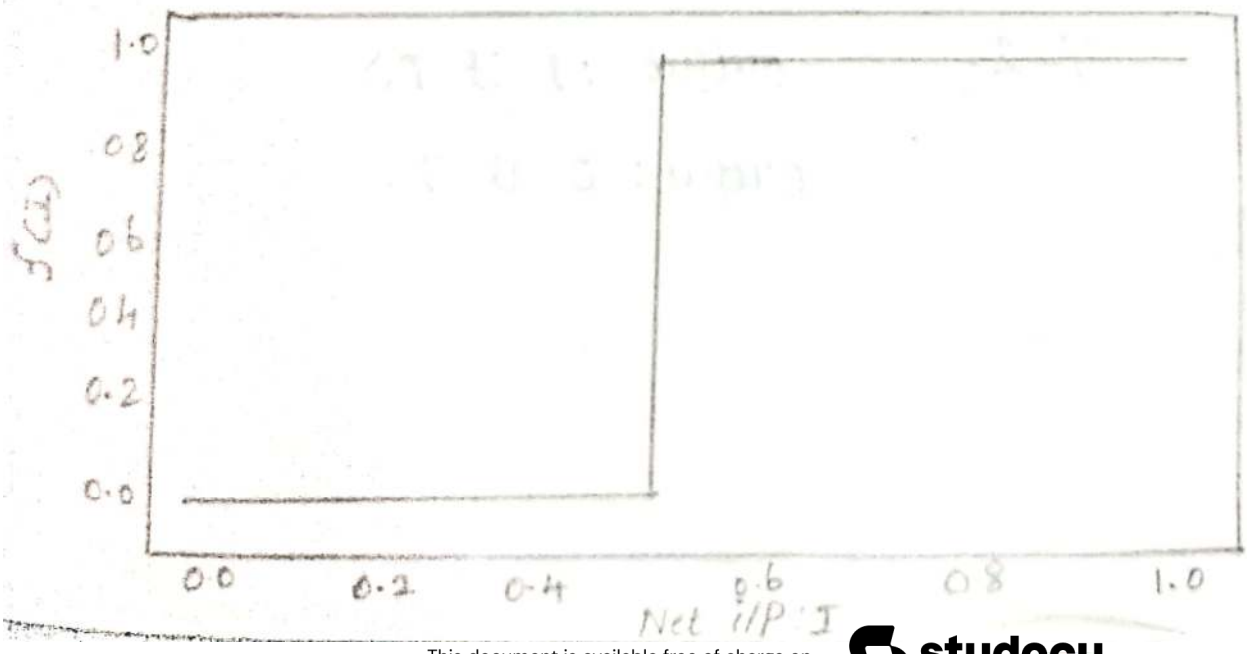
$$\text{Activation function} = f(I) = \begin{cases} 1, & \text{if } I \geq T \\ 0 & \text{if } I < T \end{cases}$$ - binary step function.

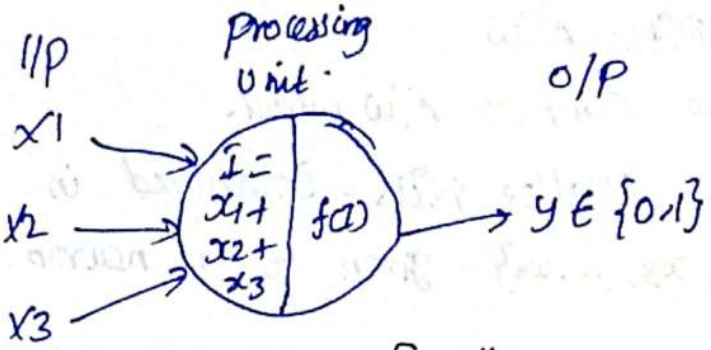
↓ o/p is either 0/1 based on Threshold (T).

$T = 0.5$

i/p → < 0.5 → o/p → 0

i/p → ≥ 0.5 → o/p → 1





modeling a simple AND function

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

The net input given $\sum_{i=1}^{2} x_i$ is.

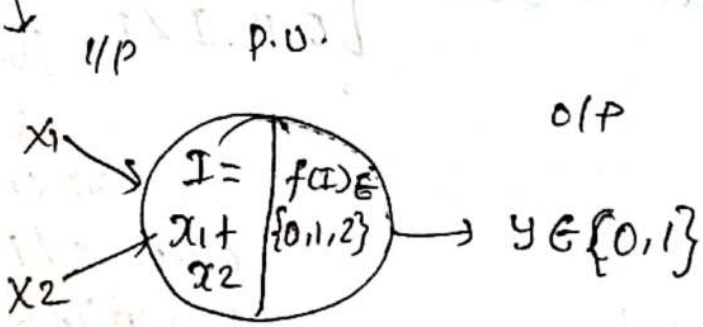
Net i/p (1,1) : 1+1=2

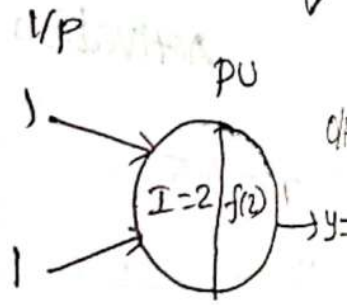
" (1,0) : 1+0=1

" (0,1) : 0+1=1

" (0,0) : 0+0=0.

when the value of net i/p exceeds '1' the n/w should O/P -1.





T= 2.

output : 1 if $T \geq 2$

output : 0 if $T < 2$

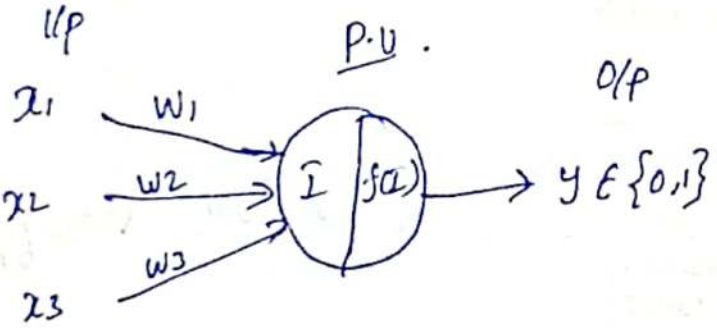
Perceptron Algorithms:

The improvement to the McCulloch-Pitts N/W is the Perceptron. It is linear classifier that performs binary classifications.

This has single layer ie. one i/p layer, one o/p layer.
↓ referred to o/p layer.

The i/p to a Perceptron is vectored with real values. The o/p is binary → 0/1.

The major improvement over Mcculloch-Pitts N/W is possibility to introduce weights in N/W & learn the weights.



weights $\{w_1, w_2, w_3, \ldots w_n\}$ are associated with each i/p. The neuron should either fire (1) or not fire (0) based on i/p & weights

The Net input $i = \sum_{i=1}^{m} w_i * x_i$.

m → Size of vector.
w_i → real valued weights

Dot product of two vectors as $W \cdot x$ / Product of two matrices as $W^T \cdot x$.

As given in mcculloch-pitts N/W

If $\sum_{i=1}^{m} x_i * w_i \geq T$, it belongs to class 1

and

If $\sum_{i=1}^{m} x_i * w_i < T$, it class 0.

can be written as

If $\sum_{i=1}^{m} x_i w_i - T \geq 0$, class 1

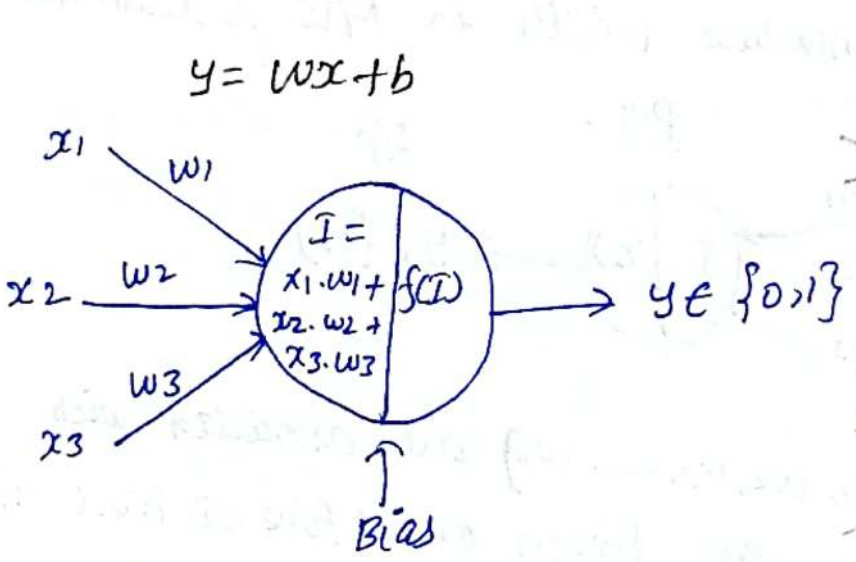
If $\sum_{i=1}^{m} x_i w_i - T < 0$, class 0.

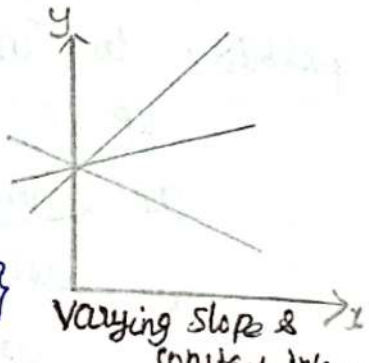
$w \cdot x - T = 0$ is the line separating two classes and can be written as $w \cdot x + b = 0$, where $b = -T$ and is called bias.

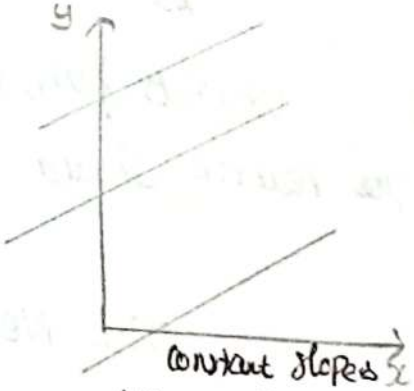
I/p → $x_1, x_2, \ldots x_n$, weight vector ⇒ $w_1, w_2, \ldots w_n$.

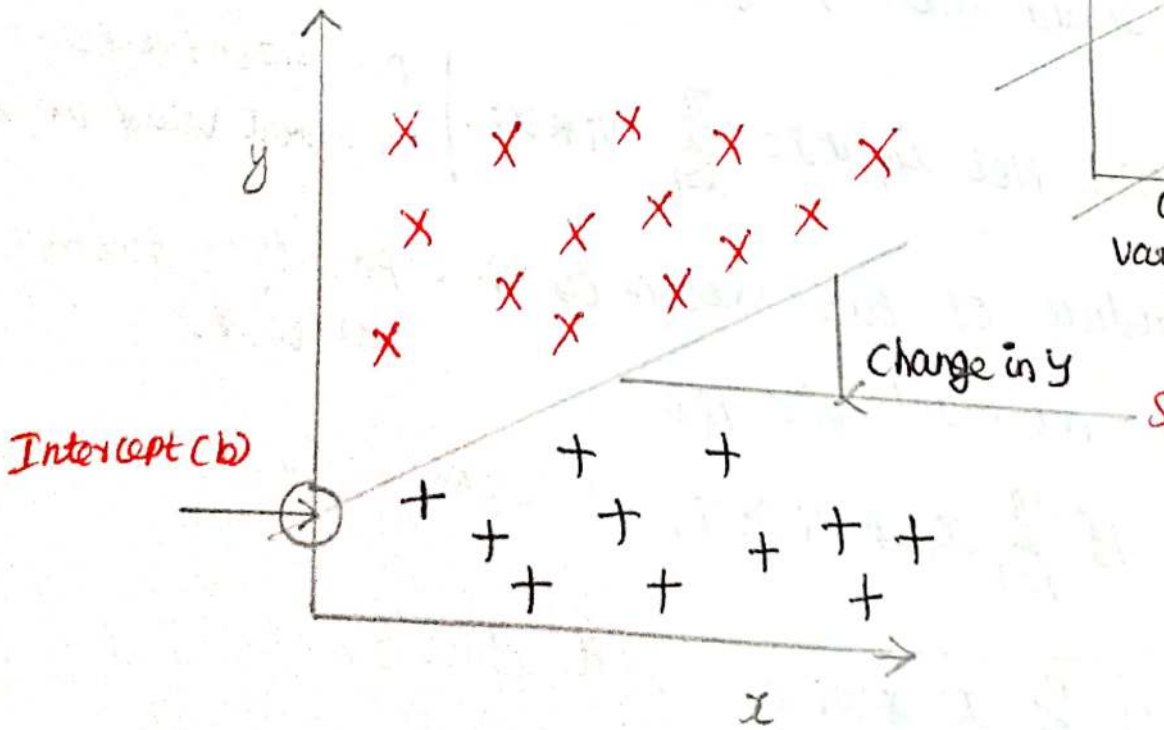
bias $b \rightarrow b_1, b_2, \ldots b_m$. $n \rightarrow$ No. of input nodes.

$m \rightarrow$ " Output Nodes.









Learning Algorithm:

In ANN, learning algorithm is an algorithm that helps to find values of the weights & bias.

1. Initialize the weights and bias randomly.
2. Combine the input, weight and bias as

$$I = \sum_{i=1}^{m} x_i * w_i + b$$

3. Feed I to the neuron in the output layer
4. pass I to the Activation function f(I) to introduce non-linearity.
5. Get the O/P of the o/p neuron as y.
6. Check the diff b/w actual O/P O and the predicted o/p y.
7. If $(y-O) \neq 0$ then,
   (a) Adjust (Increase/Decrease) the weights & bias accordingly
   (b) Go to Step 2.
8. Else
   (a) Stop the process as the weights & bias have been correctly identified.

This process is termed as the training the ANN. At the end process, the model would have ~~been~~ learnt the W & b.

The simplest learning algorithm finds the value of W & b by trial & Error.

But the recent advancements finds the value of W & b using optimization techniques.

The Sigmoid Neuron:

If the pblm requires an O/P of real numbers between (0,1) and (-1,1) the best option is to use a sigmoid activation function. The sigmoid function produces S-shaped curve commonly, referred to as a sigmoid curve.

The two commonly used sigmoid function in ANN are, logistic function. the hyperbolic tang

Logistic function:

Given by.

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

It squashes the I/P to a value in the range (0,1). Functions that either only increase or decrease as the i/p increases are monotonic functions.

First derivative of logistic function

the logistic function rewritten as.

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

first derivative is also gradient of the logistic function. This can be derived as follows:

$$\frac{df(x)}{dx} = \frac{e^{x}(1-e^{x}) - e^{x}.e^{x}}{(1+e^{x})^{2}} = \frac{e^{x}}{(1+e^{x})^{2}} = \frac{e^{x}+e^{x}.e^{x}-e^{x}.e^{x}}{(1+e^{x})^{2}}$$

$$= \left(\frac{e^{x}}{(1+e^{x})}\right)\left(\frac{1+e^{x}-e^{x}}{1+e^{x}}\right) = \left(\frac{e^{x}}{(1+e^{x})}\right)\left(\frac{1+e^{x}}{(1+e^{x})} - \frac{e^{x}}{(1+e^{x})}\right)$$

$$= f(x)\,[1-f(x)]$$

Hyperbolic Tangent:

given by,

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

It squashes the I/P in the range (-1,1). In most cases, the hyperbolic Tangent is preferred over the logistic function. because it has stronger gradients.

Linear seperable Problem:

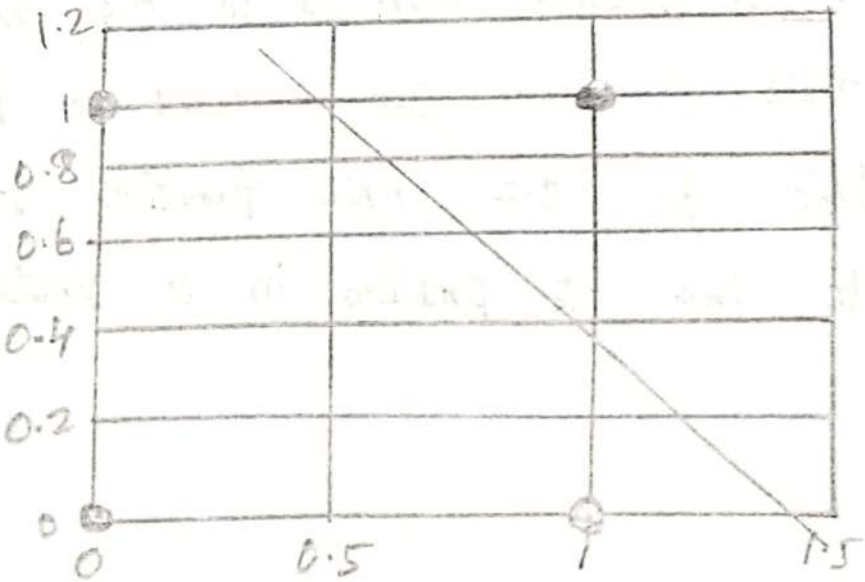
perceptron solve only linearly seperable problems. consider binary classification problems. that is there are only two possible class labels. for ex. AND, OR functions can be viewed as a binary classification pblm with class labels 0 and 1.

Every point to the right of the line belongs to class. A single line was enough to classify the 2 classes. These are called linearly seperable pblms. classifiers that solve only linear seperable pblms are linear classifi

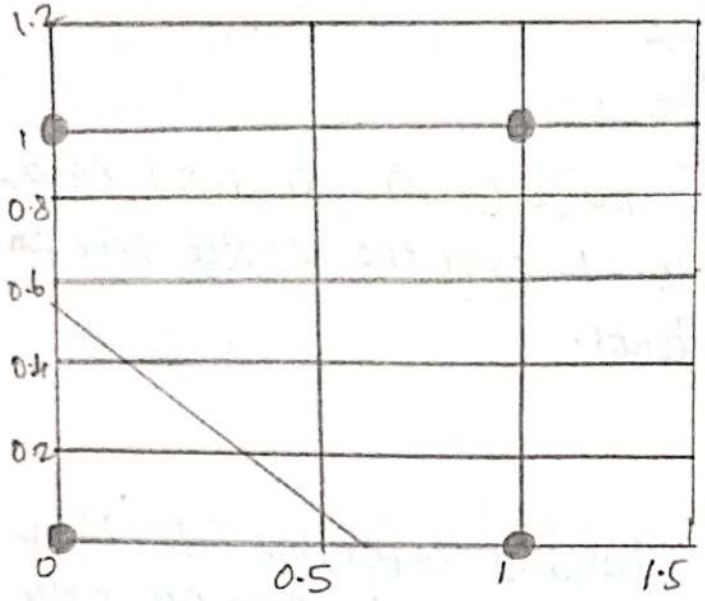
consider XOR truth table,

x	y	x XOR y.
0	0	0
0	1	1
1	0	1
1	1	0

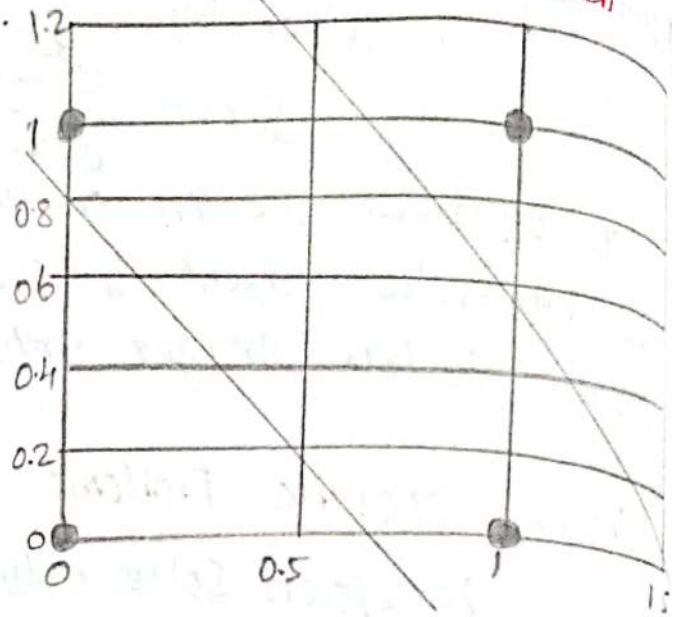
AND FUNCTION



OR Function



XOR Function



Perceptron to solve linearly seperable problems.

x_1	x_2	x_1 AND x_2	Equation of seperator.	inequality
0	0	0	$\sum_{i=1}^{2} x_i * w_i + w_0 < 0$	$0 \cdot w_1 + 0 \cdot w_2 + w_0 < 0$
0	1	0	$\sum_{i=1}^{2} x_i * w_i + w_0 < 0$	$0 w_1 + 1 \cdot w_2 + w_0 < 0$
1	0	0	$\sum_{i=1}^{2} x_i * w_i + w_0 < 0$	$1 \cdot w_1 + 0 \cdot w_2 + w_0 < 0$
1	1	1	$\sum_{i=1}^{2} x_i * w_i + w_0 \geq 0$	$1 \cdot w_1 + 1 \cdot w_2 + w_0 \geq 0$

For example $w_0 = 0.5$, $w_1 = 1.5$, $w_2 = 1$ is one of the many possible solutions to the inequalities. This shows that Perceptron can be used to find the seperating line for the AND function. Solution can be given to the OR problem in a similar manner.

x_1	x_2	x_1 OR x_2	Equation of seperator		Inequalities.
0	0	0	$\sum_{i=1}^{2} x_i * w_i + w_0 < 0$	$0 \cdot w_1 + 0 \cdot w_2 + w_0 < 0$	$w_0 < 0$
0	1	1	$\sum_{i=1}^{2} x_i * w_i + w_0 \geq 0$	$0 \cdot w_1 + 1 \cdot w_2 + w_0 \geq 0$	$w_2 \geq -w_0$
1	0	1	$\sum_{i=1}^{2} x_i * w_i + w_0 \geq 0$	$1 \cdot w_1 + 0 \cdot w_2 + w_2 \geq 0$	$w_1 \geq -w_0$
1	1	1	$\sum_{i=1}^{2} x_i * w_i + w_0 \geq 0$	$1 \cdot w_1 + 1 \cdot w_2 + w_2 \geq 0$	$w_1 + w_2 \geq -w_0$.

one possible solution to this problem is again.

$w_0 = -0.5$, $w_1 = 1.5$, $w_2 = 1$.

x_1	x_2	x_1 XOR x_2.	Equation of seperator		Inequalities.
0	0	0	$\sum_{i=1}^{2} x_i * w_i + w_0 < 0$	$0 \cdot w_1 + 0 \cdot w_2 + w_0 < 0$	$w_0 < 0$
0	1	1	$\sum_{i=1}^{2} x_i * w_i + w_0 \geq 0$	$0 \cdot w_1 + 1 \cdot w_2 + w_0 \geq 0$	$w_2 \geq -w_0$
1	0	1	$\sum_{i=1}^{2} x_i * w_i + w_0 \geq 0$	$1 \cdot w_1 + 0 \cdot w_2 + w_0 \geq 0$	$w_1 \geq -w_0$
1	1	0	$\sum_{i=1}^{2} x_i * w_i + w_0 < 0$	$1 \cdot w_1 + 1 \cdot w_2 + w_0 < 0$	$w_1 + w_2 < -w_0$.

When we try to solve the inequalities. we get 2nd, 3rd from inequalities. $w_1 + w_2 \geq -w_0$ This contradicts fourth inequality $w_1 + w_2 < -w_0$.

A perceptron that solves both above mentioned inequalities is impossible to design. we require a different neural N/w architecture to solve non-linearly seperable pblms. Multi layer perceptron introduced to solve both the linearly seperable and Non-linearly seperable problems.

Multilayer Perceptron (MLP) / Feed forward Networks:

Consider a multilayered directed graph. Multilayered graphs have layers of nodes with the flow of data from the first layer to the second layer, from the second layer to the ~~first~~ 3rd layer and so on. This type of N/w with information flowing from one end to other in one-direction through the intermediate layers is called feed-forward Network.

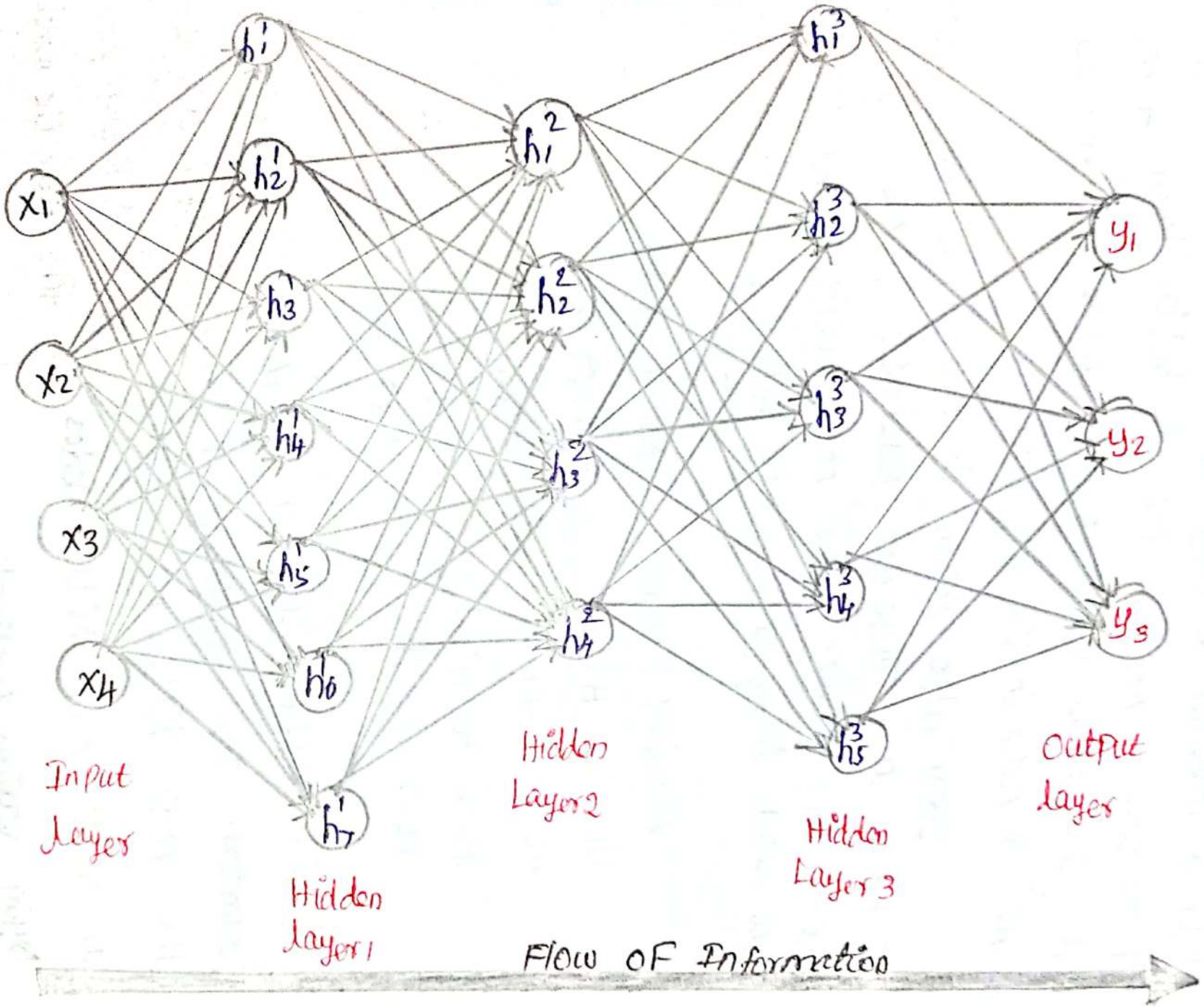
The first layer and last layers are the I/P and O/P layers respectively, the intermediate layers are hidden layers. A N/w with multiple hidden layers is called a multilayered perceptron.

The nodes in the hidden layers and O/P layer are processing units with non-linear activation functions. The multiple hidden layers combined with non-linear activation function helps it to solve non-linearly seperable pbms.

The diff b/w linear and non-linear activation functions is as follows:

Linear functions	Non-Linear functions.
1. Present using Straight line.	1. not represent using Straight line, using curves and other structures.
2. Polynomial of degree ≤ 1	2. polynomial of degree > 1
3. The slope is constant throughout the function.	3. slope is always changing.

Multilayer Perceptron



The N/W is 4 layered N/W with three hidden layers and and an O/P layer. The hidden layers can have any no. of nodes in each layer. But every node in one layer is connected to every node in the next layer. There are 7 nodes in the 1st hidden layer, 4 nodes in the 2nd hidden layer, 5 nodes in the 3rd hidden layer and 3 nodes in the O/P layer.

Every edge shown in the figure has a weight attached to it. These are parameters of the MCP that are to be learnt. The no. of weights and bias learnt b/w each layer is given the below is Table.

Associated layer.	No. of weights	Bias
I/P to H.L. 1	4x7 = 28	7
HL.1 to HL 2	7x4 = 28	4
HL.2 to HL.3	4x5 = 20	5
HL.3 to O/P	5x3 = 15	3

This results in a total of 91+19 = 110 Parameters. The complexity of neural N/W depends on the no. of hidden nodes. The best way known for to right no. of hidden nodes is fine tune the model trying various combinations.

The no. of nodes in i/p layer is determined by the features. For the iris dataset, the sepal and petal lengths and width are the four features of an iris flower. This results in four i/p nodes.

The no. of output node depends on the no. of o/p. In the iris dataset, the possible outputs are Iris Setosa, Iris versicolour, and Iris Virginica. So the no. of o/p nodes for the iris data set is 3.

Multi layered Perceptron introduced a new problem. When we had single layer perceptron, the weights & bias b/w hidden and o/p layers were updated using the diff. b/w actual and predicted o/p in the o/p layer. In multilayer Perceptrons there is no basis to know the expected o/p is a hidden layer.

The learning algorithm used for single layer perceptron needs to be modified and applied for multi layer perceptrons. The learning algorithm used in multi layer perceptron is the Back propagation algorithm. To have better understanding of the working of an ANN, we shall know the optimization techniques used.

Error Functions: The error function, also termed as the loss function, is used to Present the diff b/w the actual and Predicted outputs.

Some of the common loss functions are the least Absolute Deviations, Least Square Error and Cross-Entropy loss function.

Let us denote actual output O and Predicted o/p y

1. Least Absolute Deviations (LAD): is also termed as L1 norm loss function. It is given by the formula.

$$LAD = \sum_{i=1}^{n} |O_i - y_i| \quad n = \text{no. of Samples.}$$

2. Least Square Error (LSE): is also termed as L2 norm Loss function. It is given by the formula,

$$LSE = \sum_{i=1}^{n} (O_i - y_i)^2.$$

3. Cross-Entropy Loss: If the o/p of a classification model is probability value b/w 0 and 1, the best loss function to uses is the Cross-Entropy Loss (CEL). function given by,

$$CEL = -\sum_{i=1}^{c} O_i \log(y_i)$$

c = no. of classes.

The error function is a function of the weights and bias because the alteration of 'y' and 'x' beyond our control and depends only on the dataset. This can be represented as $L(w.b)$.

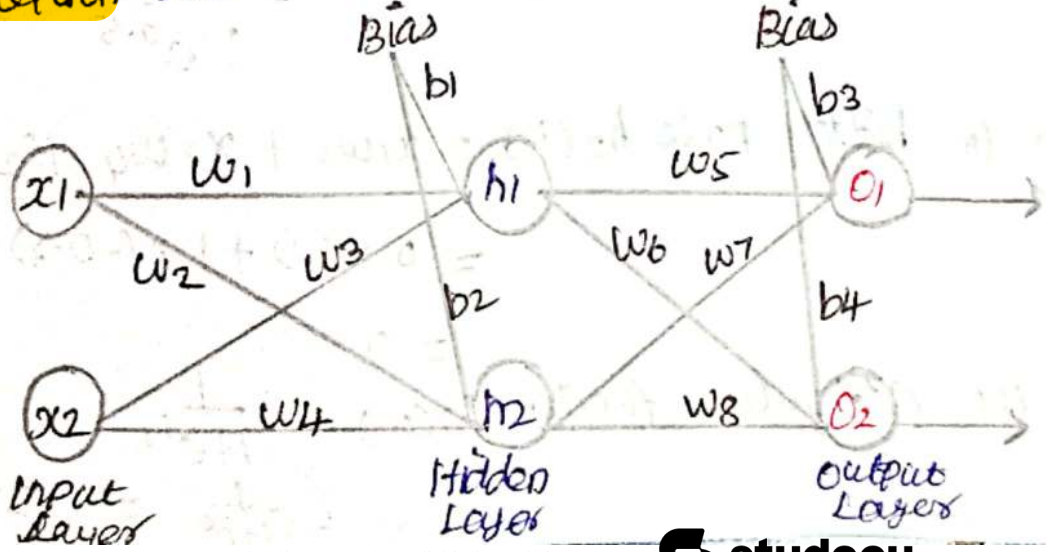
A model with an error of 0 learns the training data perfectly. The problem at hand is how to change the values of w and b such that $L(w,b)$ move towards 0. that is find those 'w'. 'b'. that minimize $\in L(w.b)$

$\nabla w = \frac{\partial L(w.b)}{\partial w}$ - Gradient of the cost function with respect to w.

$\nabla b = \frac{\partial L(w.b)}{\partial b}$ - Gradient of the cost function with respect to b.

Back Propagation Algorithm:

It is termed the **Backpropagation algorithm** because though the **information** flows in the **forward** direction, the **error** is propagated in the **backward direction**. Backpropagation algorithm is the **heart** of the **multilayer Perceptron** and is explained using an example.



This is a Two-layered ANN with 2 nodes in the hidden layer and 2 nodes in the o/p layer. Let us use the logistic function as the activation function. The weights and bias are randomly initialized as given the table.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	b_1	b_2	b_3	b_4
-0.1	0.2	0.6	-0.3	0.4	0.6	0.5	-0.2	-0.1	+0.3	0.4	0.2

The I/P and O/P vectors are given in the following table.

x_1	x_2	O_1	O_2
0	1	0	1

Calculation for Hidden Layers

Net I/P to hidden node $h_1(i_1) = x_1 w_1 + x_2 w_3 + b_1$

$= 0 * (-0.1) + 1 * (0.6) + (-0.1)$

$= 0.5$

Output of the activation function at $h_1(a_1) = \frac{1}{1+e^{-i_1}}$

$= \frac{1}{1+e^{-0.5}} = 0.62$

Net I/P to hidden node $h_2(i_2) = x_1 w_2 + x_2 w_4 + b_2$

$= 0 * (0.2) + 1 * (-0.3) + 0.3$

$= 0$

O/P of the activation function $h_2(a_2) = \frac{1}{1+e^{i_2}}$

$= \frac{1}{1+e^{0}} = 0.5$

Calculation for output layers:

Net I/P to O/P node $O_1(i_3) = a_1 * w_5 + a_2 w_7 + b_3$

$= (0.62) * (0.4) + (0.5) * (0.5) + 0.4$

$= 0.9$

O/P of an activation function at $O_1(y_1) = \frac{1}{1+e^{-i_3}} = \frac{1}{1+e^{-0.9}} = 0.71$

Net I/P to O/P node $O_2(i_4) = a_1 * w_6 + a_2 w_8 + b_4$

$= (0.62) * (0.6) + (0.5) * (-0.2) + 0.2$

$= 0.41$

O/P of an activation function at $O_2(y_2) = \frac{1}{1+e^{-i_4}} = \frac{1}{1+e^{-0.41}} = 0.6$

Calculation of Error:

To find diff b/w actual and predicted o/p's, we use squared error function $[L(w,b)]$.

$$L(w,b) = \sum_{i=1}^{n} \frac{1}{2} (O_i - y_i)^2.$$

For our pblm,

$L(w,b) = (0.5) * (0 - 0.71)^2 + 0.5) * (1 - 0.6)^2$

$= 0.17$

Back Propagation Error:

The Parameters of the Error function are the weights and bias. The goal in Parameter updation is that the weights and bias must be changed in a such a way that the error function approaches 0. That is the Actual and Target values should be as close as possible and preferably same.

So what would be the value of w_5 if $L(w,b)$ should tend to 0. This can be written as how does $L(w,b)$ change when w_5 changes and is given by Partial derivative ~~of~~, $\partial L(w,b)/\partial w_5$. Applying the chain rule, we get

$$\frac{\partial L(w,b)}{\partial w_5} = \frac{\partial L(w,b)}{\partial y_1} * \frac{\partial y_1}{\partial i_3} * \frac{\partial i_3}{\partial w_5}$$

$$\frac{\partial L(w,b)}{\partial y_1} = \frac{\partial \left(\frac{1}{2}(o_1 - y_1)^2 - \frac{1}{2}(o_2 - y_2)^2 \right)}{\partial y_1} \quad -0$$

$$= 2 * \frac{1}{2}(o_1 - y_1)^{2-1}(-1) + 0$$

$$= -(o_1 - y_1) = -(0 - 0.71) = 0.71$$

$$y_1 = \frac{1}{1+e^{-i_3}}$$

$$\frac{df(x)}{dx} = f(x)\,[1 - f(x)]$$

Therefore

$$\frac{dy_1}{di_3} = y_1(1-y_1) = 0.71 * (1-0.71) = 0.2$$

also, $i_3 = a_1 w_5 + a_2 w_7 + b_3$ so.

$$\frac{di_3}{dw_5} = 1 * a_1 * w_5^{(1-1)} + 0 + 0$$

$$= a_1 = 0.62$$

Substituting Equations.

$$\frac{dL(w,b)}{dw_5} = 0.71 * 0.2 * 0.62 = 0.09$$

$$\frac{dL(w,b)}{dw_5} = -(O_1 - y_1) * y_1(1-y_1) * a_1$$

Similarly, we can find the weights for w_6, w_7 & w_8 as follows

$$\frac{dL(w,b)}{dw_6} = -(O_2 - y_2) * y_2(1-y_2) * a_1 = -0.06$$

$$\frac{dL(w,b)}{dw_7} = -(O_1 - y_1) * y_1(1-y_1) * a_2 = 0.07$$

$$\frac{dL(w,b)}{dw_8} = -(O_2 - y_2) * y_2(1-y_2) * a_2 = -0.05$$

For generalization, we write the error at output node j due to processing unit h is

$$\frac{\partial L(w,b)}{\partial w_{jk}} = \nabla w_{jk} = Err_j * a_h.$$

where a_h is the op of the activation function at the hidden node. similarly,

$$\frac{\partial L(w,b)}{\partial b_3} = -(O_1 - y_1) * y_1(1-y_1) = 0.15$$

$$\frac{\partial L(w,b)}{\partial b_4} = -(O_2 - y_2) * y_2(1-y_2) = -0.10.$$

The gradient at o/p node j due to processing unit and bias b_k is

$$\frac{\partial L(w,b)}{\partial b_k} = \nabla b_k = Err_j$$

Let $\eta = 0.1$. Then w_5 is updated using gradient descent as follows

$$\text{new\_weight} = \text{old\_weight} - \eta \nabla w$$

$$w_k = w_k - \eta \nabla w_k.$$

$$w_5 = w_5 - \eta \frac{\partial L(w,b)}{\partial w_5}$$

$$= 0.4 - (0.1 * 0.09) = 0.39$$

The bias is updated using gradient descent as follows:

$$new\_bias = old\_bias - \eta \nabla b$$

$$b_K = b_K - \eta \nabla b_K.$$

$$b_3 = b_3 - \eta \nabla b_3 = 0.4 - 0.1 * 0.15$$

$$=$$

The updated weights are.

w_5	w_6	w_7	w_8	b_3	b_4
0.39	0.61	0.49	−0.2	0.39	0.21

Updating Weights at Hidden Layer:

We move backwards and update the weights to the hidden nodes. This updation is slightly more complex because the hidden nodes affects the O/P nodes. Let us use the following notations.

1. w_{jk} is weight of the nodes b/w j and K.
2. w_{kl} is " " b/w K and l.
3. a_h is O/P of the node h at hidden layer K.

Note that, layers j and K can be both hidden nodes or i/p layer j and hidden Layer (K) as in our example. The nodes in layer K affects the O/P of the nodes in the next layer denoted as l.

The error at unit h in the hidden layer is given as.

$$Err_h = (a_h)(1-a_h) \sum_k Err_k w_{kl}.$$

The weights and bias in the hidden layers are updated using the formulas

$$w_{jk} = w_{jk} - \eta\, Err_h \cdot a_h$$

$$b_j = b_j - \eta\, Err_h$$

This is applicable for any no. of hidden layers. Substituting the values in the example Pblm we get.

$$Err_{h_1} = (a_1)(1-a_1)(Err_1 \cdot w_5 + Err_1 \cdot w_6)$$

$$= (0.62)(1-0.62)(0.15 * 0.4 + 0.15 * 0.6)$$

$$= 0.03.$$

$$Err_{h_2} = (a_2)(1-a_2)(Err_2 \cdot w_7 + Err_2 \cdot w_8)$$

$$= 0.5(1-0.5) * [-0.1 * 0.5 + (-0.1) * (-0.2)]$$

$$= -0.01$$

The above mentioned updations are continued till the error function in o/p layer approaches 0. The values of w_i and b_i are calculated as follows.

$w_1 = w_1 - \eta \cdot Err_{h_1} \cdot a_1 = -a_1 - 0.1 \times 0.03 * 0.b_2$

$= -0.1$

$b_1 = b_1 - \eta\, Err_{h_1} = -0.1 - 0.1 * 0.03 = -01.$

The remaining weights are also updated in the following manner.

w_1	w_2	w_3	w_4	b_1	b_2
-0.1	0.2	0.b	-0.3	-0.1	0.3

After the calculation of all the weights, all the weights are updated. This is said to be an epoch. The above mentioned updations are continued till the error function in the O/P layer approaches 0.

Algorithm Weight_Update_BackPropagation (i, j, old_weight)

{

for (Specified no. of epochs) OR (until minima of the error function is reached)

for each weight j between hidden layer and the O/P layer.

$$Err_j = -(O_j - y_j) * y_j (1 - y_j)$$

$$\Rightarrow w = Err_j * a_h$$

$$\Rightarrow b = Err_j$$

End for

for each weight jk in the hidden layer

$$Err_h = (a_h)(1-a_h) \sum_K Err_K w_{jk}$$

$$\nabla w = Err_h \cdot a_j$$

$$\nabla b = Err_h .$$

End for

new_weight = old_weight $- \eta \nabla w$

new_bias = old_bias $- \eta \nabla b$

}.