DEEP LEARNING

Limitations of Machine Learning:

(1) feature extraction

(2) for complex problems such as object recognition or handwriting recognition is a huge challenge

prediction of future, whether match will be conducted or not, you can consider dataset many features. Sunny, windy, humidity all those features provide in our dataset

Deep learning to the rescue:

(1) DL models are capable to focus on the right features by themselves, requiring little guidance from the programmer.

(2) These models also partially solve the dimensionality problem.

Def: the Idea behind Deep learning is to build learning algorithms that mimic brain.

(1) DL is implemented through Neural Networks

(2) Motivation behind the NN is the biological Neuron

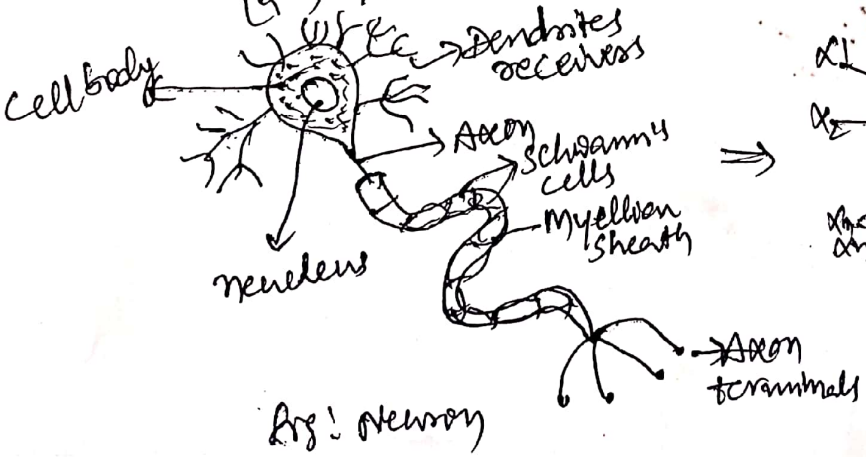


Fig: Neuron

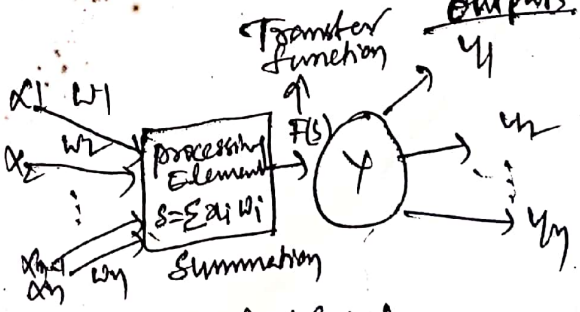
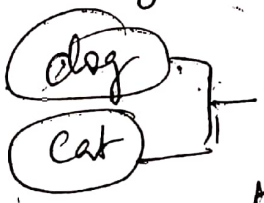


Fig: Artificial neural networks

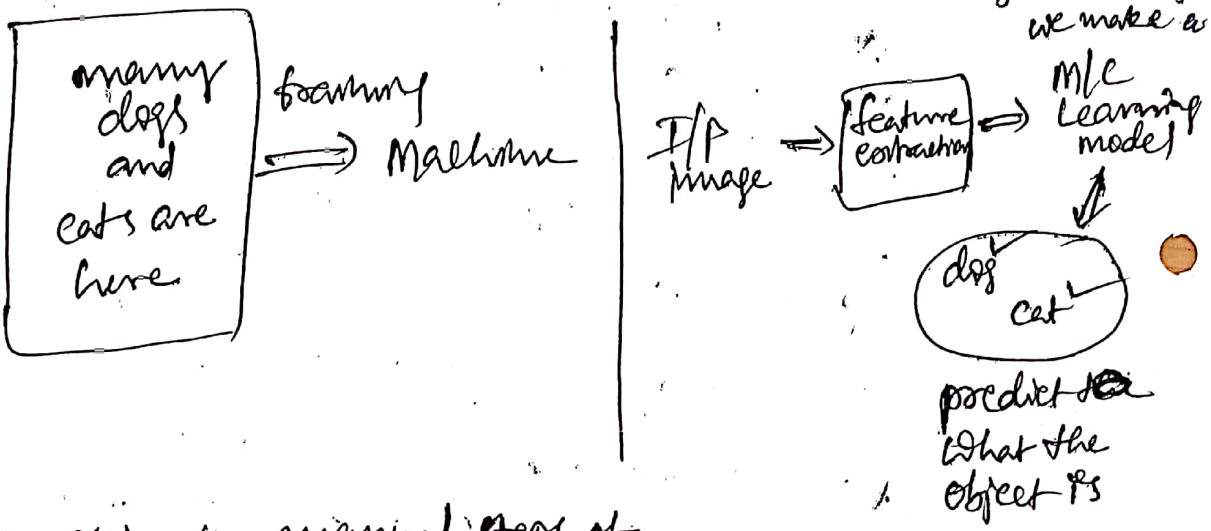
Agenda

- What is Deep learning?
- How DL works
- Single layer perceptron → Artificial neurons = brain cells
- Single layer perceptron Examples & limitations
- Multi layer perceptron & examples

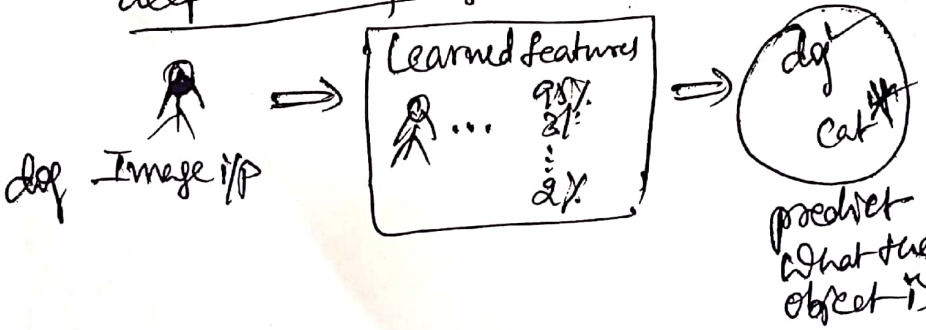
Deep learning: Humans have seen so many dogs and cats, they can identify the difference between the two



How a given image is dog or a cat?



DL skips the manual steps of extracting features, you can directly feed images to the deep learning algorithm, which then predicts the object



If we put human brains in the machines machine never tired and give good results. Therefore Artificial neurons are to be created. There is a system can imitate our brains artificial neurons are classify the flowers based on features. The system can seperate the species based on its features. That system is artificial neuron.

Why we need Artificial Neuron?

With the help of Artificial Neuron we can seperate the two species of flowers.

→ one type of Artificial Neuron is perceptron.

→ Perceptron Learning Algorithm:

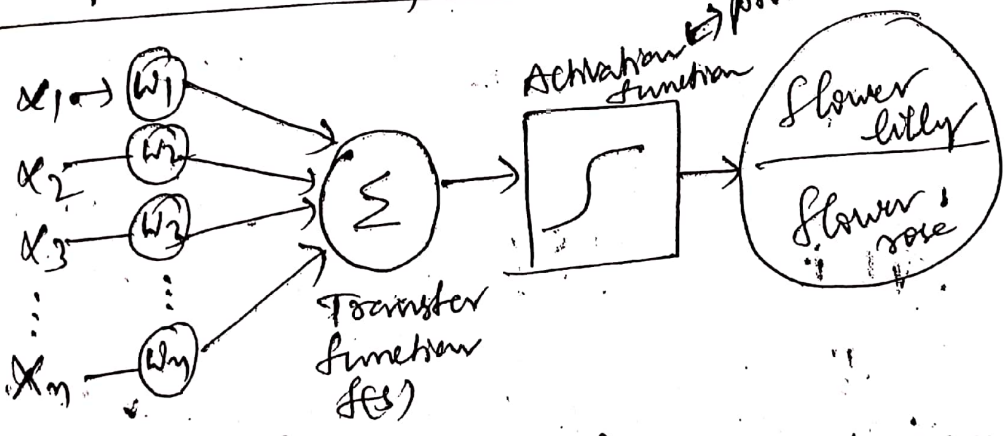


Fig: Schematic for a neuron in a neural net.

Weight — how important of the i/p.

Perceptron: It is classified the species linearly

It is able to classify which are linearly seperable

Algorithm

Step 1: Initialize the weights and threshold → w_j – Initial weight

↓

provide the Input and calculate the o/p → x – Input, y – output

↓

Update the weights → $w_j(t+1) = w_j(t) + \eta(d-y)x$

↓ η – learning rate between 0-1

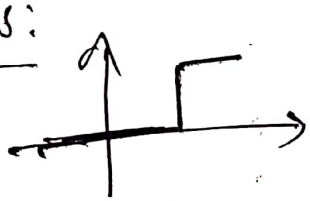
↓

Repeat Step 2 & Step 3

$w_j(i+1)$ – Updated weight
$w_j(t)$ – old weight
d – Desired output
y – Actual output
x – input

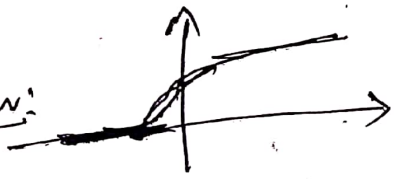
→ Activation functions:

(1) Step function:

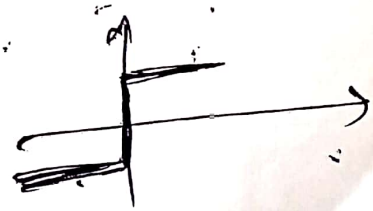


Here if the o/p exceeds above the i/p function the neuron will fire

(2) Sigmoid function:



(3) Sign function:



Applications:

(1) It is used to classify any linearly seperable set of inputs.

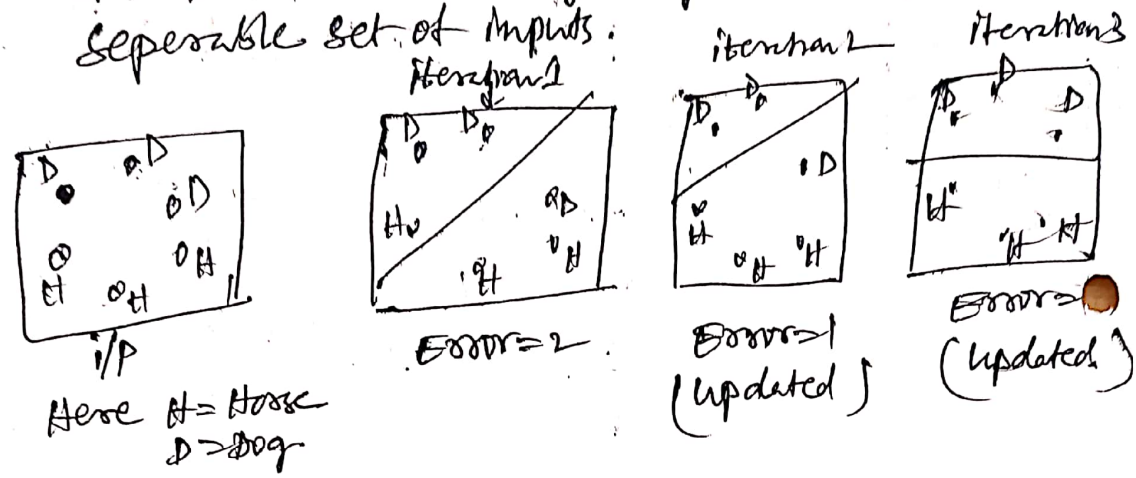
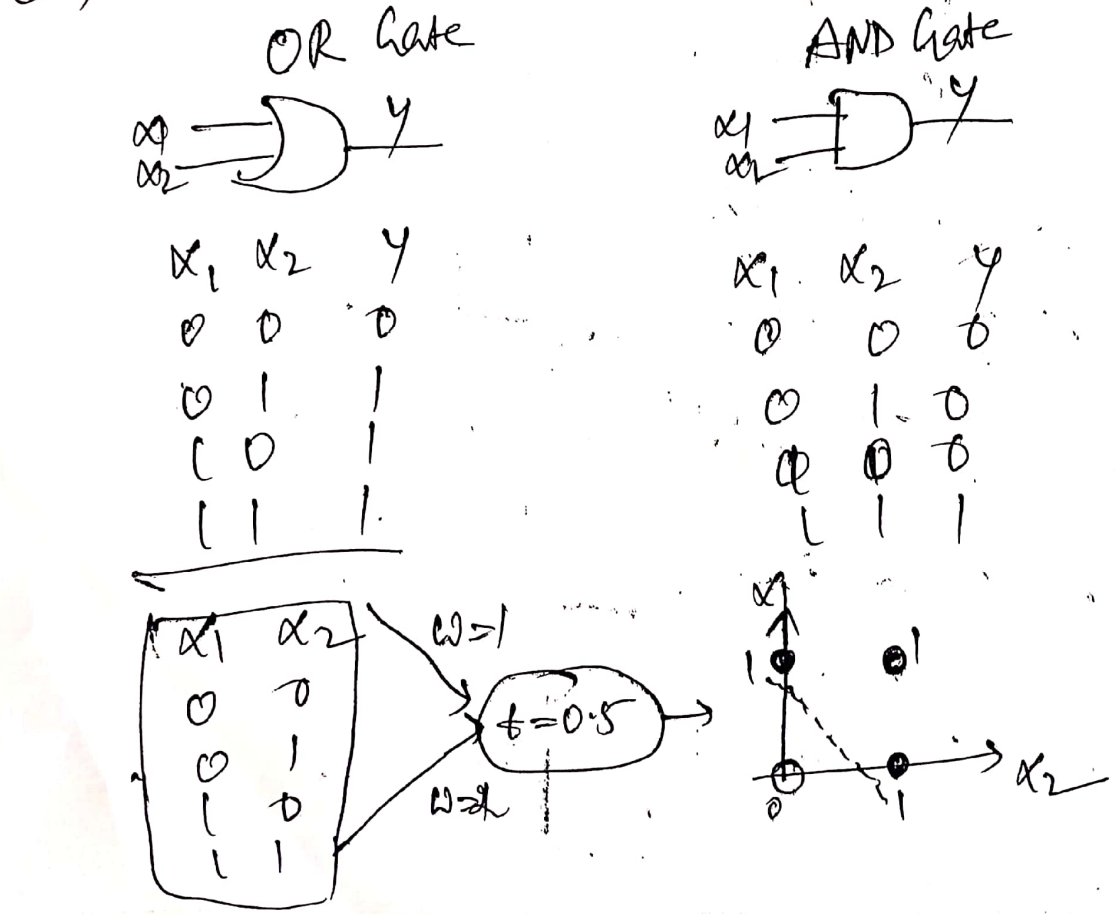


Fig: Single layer perceptron

(2) It can be used to implement Logic gates



$0+0 < 0.5$ — output will not fire

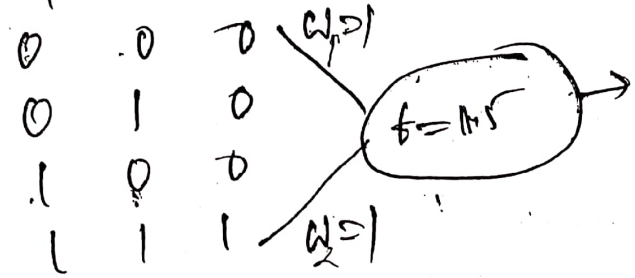
$0+1 > 0.5$ — output will fire

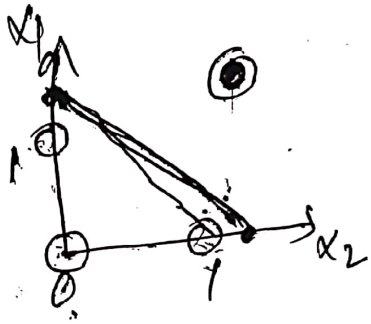
$1+0 > 0.5$ — " "

$1+1 > 0.5$ — " "

In AND Gate:

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

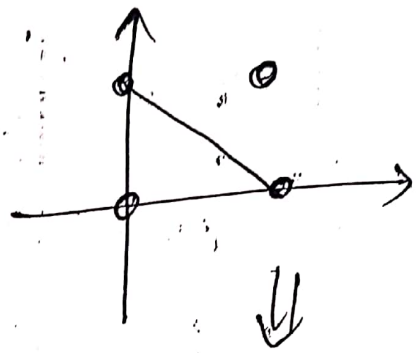




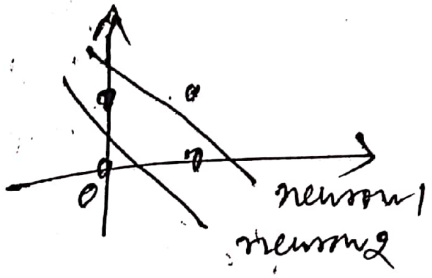
Limits of Single-layer perceptron:

When you are using XOR gate

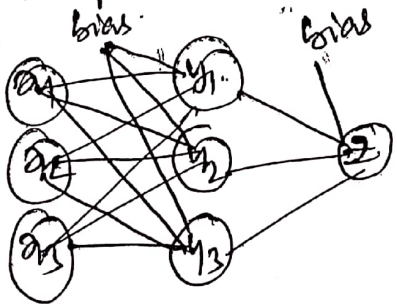
x_1	x_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

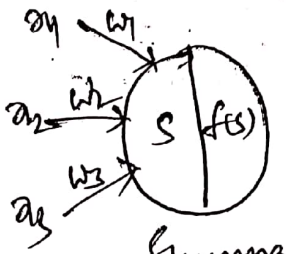


We will solve this problem with two neurons multiple neurons (Multilayer perceptron with backpropagation.



→ **Multi-layer perceptron:** It is like Single layer perceptron Structure but with one or more hidden layers ∴ considered a deep neural networks.





Summation:

$$S = \sum_{i=1}^{n} w_i * a_i$$

Transformation

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

How it works:

- The weights between the units are the primary means of long-term information storage in neural networks.
- Updating the weights is the primary way the Neural network learns new information.
- A set of inputs is passed to the first hidden layer, the activations from that layer are passed to the next layer and so on, until you reach the output layer.

- **Back propagation:** It is a supervised learning algorithm for multilayer perceptron.

In order to classify the leads on the basis of priorities, we need to provide the maximum weight to the most important lead.
For that we can calculate the difference between the actual output and the desired output.
According to that difference we can update the weights.

Introduction to ANN: (Artificial Neural Networks) → man made

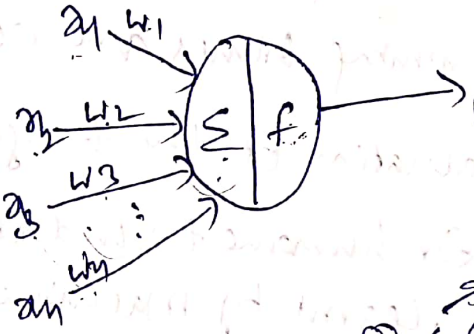
Human brain — millions of neurons

Biological Neurons

formal — Nodes

Natural functions are done artificially

Example when we touch a hot vessel



A node has two parts: ① Σ → Summation (calculate weighted sum of all i/p's.) ② f (activation function) threshold value

$x_1 w_1 + x_2 w_2 + \cdots x_n w_n = \Sigma x_i w_i$

once weighted sum is calculated sent to a activation function. if sum is greater than threshold value then o/p come out.

② Activation function: Generate the o/p, based on i/p given

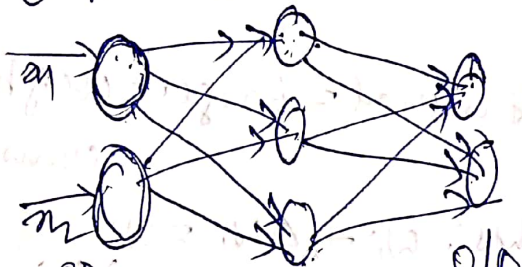
↓

$\Sigma w_i x_i$

Representation of ANN:

ANN are divided into 3 parts.

① i/p layer ② hidden layer ③ o/p layer



i/p layer → receive the i/p signals/data/ information

hidden layer → Important has neurons responsible for extracting information from i/p.

o/p layer → producing and also processing the o/p to user.

→ Appropriate problems for NNs:

1. Instances have many attribute value pairs
2. Target function has discrete values, continuous values or combinations of both
3. Training examples with errors or missing values
4. Long training times are acceptable
5. Fast evaluation of the target function learnt
6. Ability for humans to understand the target function learnt by machine is not important.

→ Perception: ① basic unit used to build ANN

② takes real valued i/p, calculates linear combination of these inputs and generates o/p

Output = 1 if result > threshold
= -1 otherwise

Linear combination & perceptron rule:

$O(x_1, x_2, \dots x_n) = 1$ if $w_0 + w_1 x_1 + \dots + w_n x_n > 0$.
$= -1$ otherwise

↓ weighted inputs

$\sum_{i=0}^{n} w_i x_i$.

O → actual o/p
t → target o/p

if actual = target weights are fixed
otherwise changed

How to change weights: $w_i + \Delta w_i = w_i$

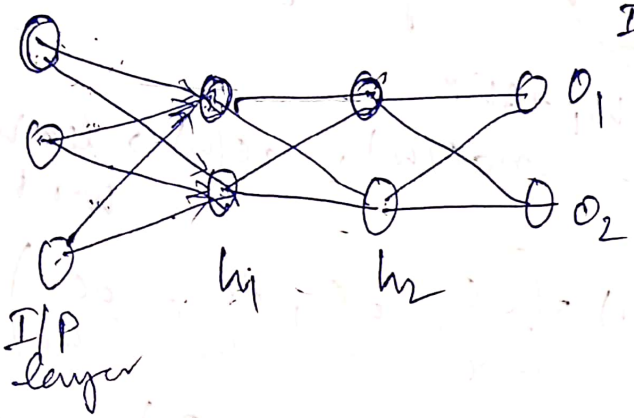
$\Delta w_i = \eta (t - O) x_i$

Where: η → learning rate (0.1 to 0.2) ← Check if $(O = t)$
x_i → i/p associated with w_i

Initially random weights, later keep on applying iterations until

Multilayer Neural networks: It is used for non-linear Seperable data.

It has two hidden layers.



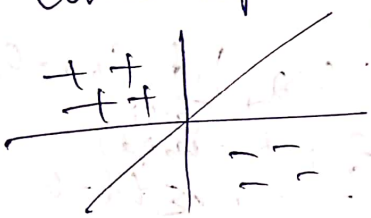
Here we can use back propagation method.

→ Gradient ~~Descent~~ rule and ~~Delta rule~~:

~~Delta rule~~ → used for non-linearly seperable data

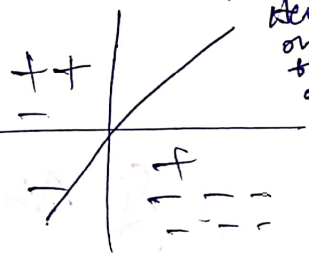
perceptron training rule → Works good only for linearly seperable data

Linear Seperable



+ → one type of data
− → another type of data
if two types of data seperable by one straight line it's called linearly seperable data.

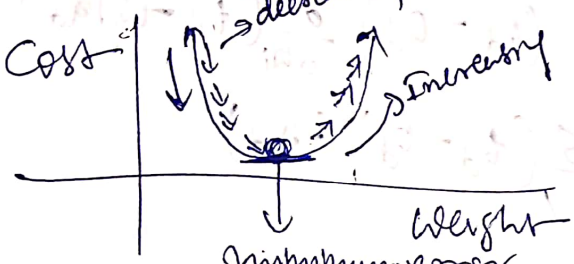
Non linearly Seperable



Here Delta rule is used to seperable data.

→ Main Idea of ~~Delta~~ Delta rule:

uses gradient descent rule and ~~delta~~ finds out the best weight



→ It is also used back propagation in order to find best weights for input

descent means reducing.

errors keep on decreasing by increasing the weights one particular weight the error is minimized. Where error is minimum you have to stop the weight.

How to modify weights:

$w_i \leftarrow w_i + \Delta w_i$

$\Delta w_i = -\eta \partial E(w)$; $\partial E(w)$ → derivative of error w.r.t weight

$$\partial E(w) = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \cdots \frac{\partial E}{\partial w_n}\right)$$

$$\frac{\partial E}{\partial w_0} = \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_{d \in D}(t_d - o_d)^2\right)$$ ⇒ formula to find error

$$E = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$ ⇒ here d → particular training example
D → All training examples.
set of training examples.

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\frac{\partial}{\partial w_i}\sum(t_d - o_d)^2$$

$$= \sum(t_d - o_d)\frac{d}{\partial w_i}(t_d - o_d)$$

$$= \sum(t_d - o_d)\frac{d}{\partial w_i}(t_d - w_d x_d)$$

$o_d = \sum w_i x_i$
$= w_0 x_0 + \cdots w_n x_n$.
$= w_d x_d$

So $\Delta w_i = -\eta \partial E(w) = -\eta \times \sum(t_d - o_d)(-x_d)$

$$= \eta \sum(t_d - o_d)(x_d)$$

η → learning rate.

$w_i = w_i + \Delta w_i$

$\frac{d}{\partial w_i}$(const) = 0 ← constant

$$\frac{\partial E}{\partial w_i} = \sum(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - w_d x_d)$$

$$\frac{\partial E}{\partial w_i} = \sum(t_d - o_d)(-x_d)$$

Back propagation Algorithm:

(Backward propagation of error)

When error occurs, we go in backward direction.

i.e; ~~O/P~~ O/P → hidden layer → i/p layer

Part 1: Calculate forward propagation error:

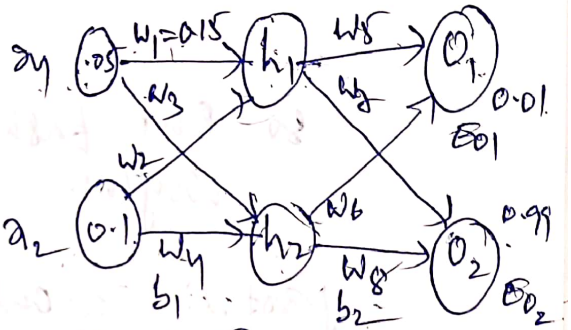
(1) Calculate h_1 (i/p and o/p)

$h_1(i/p) = w_1 a_1 + w_2 a_2 + b_1$

$= 0.377$

$h_1(out) = \frac{1}{1+e^{-h(input)}}$

$= 0.5932$



$w_1 = 0.15$ $w_5 = 0.40$

$w_2 = 0.20$ $w_6 = 0.45$

$w_3 = 0.25$ $w_7 = 0.5$

$w_4 = 0.30$ $w_8 = 0.55$

$b_1 = 0.35$; $b_2 = 0.60$

target outputs are 0.01 & 0.99.

if actual o/p = target o/p there is no error.

b_1 – bias factor between i/p and hidden layer

(2) Calculate h_2 (in and out)

$h_2(in) = a_1 w_3 + a_2 w_4 + b_1$

$= 0.3925$

$h_2(out) = 0.5968 = \frac{1}{1+e^{-h_2(in)}}$

(3) Calculate O_1 (in and out):

$O_1(in) = h_1(out) * w_5 + h_2(out) * w_6 + b_2$

$= 1.105$

$O_1(out) = \frac{1}{1+e^{-O_1(in)}} = 0.7513$

b_2 → bias between hidden layer to o/p layer.

$O_2(in) = h_1(out) * w_7 + h_2(out) * w_8 + b_2$

$= 1.2249$

$O_2(out) = \frac{1}{1+e^{-O_2(in)}} = 0.7729$

O_1 and O_2 values are not matched

∴ there is an error.

(5) Calculate E_{Total}

$$E_{Total} = \sum \frac{1}{2}(target - o/p)^2$$

$$= E_{O_1} + E_{O_2}$$

$$= \frac{1}{2}(0.01 - 0.3513)^2 + \frac{1}{2}(0.99 - 0.7728)^2$$

$$= 0.29837 \text{ approximately}$$

↓ Error which is present now means before modifications.

So our task is minimize the error by modifying weights.

Part 2: Calculating backward propagation error: (Output layer to hidden layer)

w_5, w_6, w_7, and w_8 How to adjust the weights between hidden layer and o/p layer.

→ First let us adjust w_5

modified weight of w_5: $w_5^+ = w_5 - \eta \frac{\partial E_{total}}{\partial w_5}$ | w_5 is modifying here η = learning rate we fixed to 0.6

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{O_1}} * \frac{\partial out_{O_1}}{\partial net_{O_1}} * \frac{\partial net_{O_1}}{\partial w_5}$$ | by chain rule

$$\frac{\partial E_{total}}{\partial out_{O_1}} = out_{O_1} - target_{O_1} = 0.351365 - 0.01$$

$$= 0.341356$$

$$\frac{\partial out_{O_1}}{\partial net_{O_1}} = out_{O_1}(1 - out_{O_1}) = 0.18681$$

$$\frac{\partial net_{O_1}}{\partial w_5} = out_{H_1} = 0.593269 \text{ (h.out)}$$

$$\frac{\partial E_{Total}}{\partial w_5} = 0.082167\overline{04}$$

$$w_5^* = w_5 - \eta \frac{\partial E_{Total}}{\partial w_5} = 0.35\overline{8}9\overline{9}\overline{7}\overline{36}$$

↓ modified w_5; in this way we have to calculate w_6^*, w_7^*, w_8^*

Part 3: Calculating backward propagation error

(hidden layer → input layer)

(w_1, w_2, w_3, w_4)

→ first let us adjust $w_1 = w_1^*$

$$w_1^* = w_1 - \eta \frac{\partial E_{Total}}{\partial w_1}$$

$$\frac{\partial E_{Total}}{\partial w_1} = \frac{\partial E_{Total}}{\partial out(h_1)} * \frac{\partial out(h_1)}{\partial net(h_1)} * \frac{\partial net(h_1)}{\partial w_1} = 0.03631$$

↓ final error

$E_{Total} = E_{O1} + E_{O2}$

$$\frac{\partial E_{Total}}{\partial out(h_1)} = \frac{\partial E_{O1}}{\partial out(h_1)} + \frac{\partial E_{O2}}{\partial out(h_1)}$$

$$\frac{\partial E_{O1}}{\partial net_{O1}} * \frac{\partial net_{O1}}{\partial out_{O1}}$$

↓

$$\frac{\partial E_{O1}}{\partial out_{O1}} * \frac{\partial out_{O1}}{\partial net_{O1}}$$

$$\frac{\partial E_{O2}}{\partial net_{O2}} * \frac{\partial net_{O2}}{\partial out\, h_1}$$

↓

$$\frac{\partial E_{O2}}{\partial out_{O2}} * \frac{\partial out_{O2}}{\partial net_{O2}}$$

$$\frac{\partial E_{O2}}{\partial out_{O2}} = (out_{O2} - target_{O2}) = 0.222 - 0.99 = -0.21702$$

→ $\frac{\partial out\, h_1}{\partial net\, h_1} = 0.2413$; → $\frac{\partial net\, h_1}{\partial w_1} = \frac{\partial}{w_1}(w_1 out_{w_1} + w_2 out_{w_2} + b_1) = x_1 = 0.05$

$out\, h_1 (1 - out\, h_1)$

$$\frac{\partial E_{Total}}{\partial w_1} = 0.004388\,65$$

$$w_1^{+} = w_1 - \eta \frac{\partial E_{Total}}{\partial w_1} = 0.149786.$$

Like that apply to same process to calculate w_2^{+}, w_3^{+}, and w_4^{+}.

$$=$$

$$\frac{d\,out_{o_2}}{d\,net_{o_2}} = out_{o_2}(1 - out_{o_2}) = 1.3551005\Omega$$