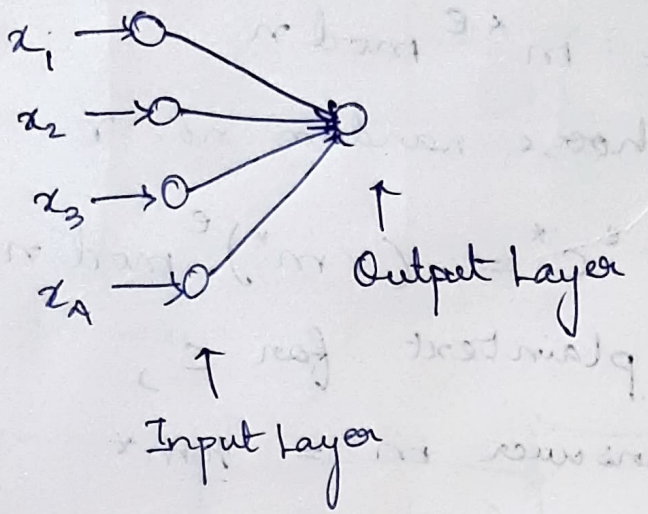
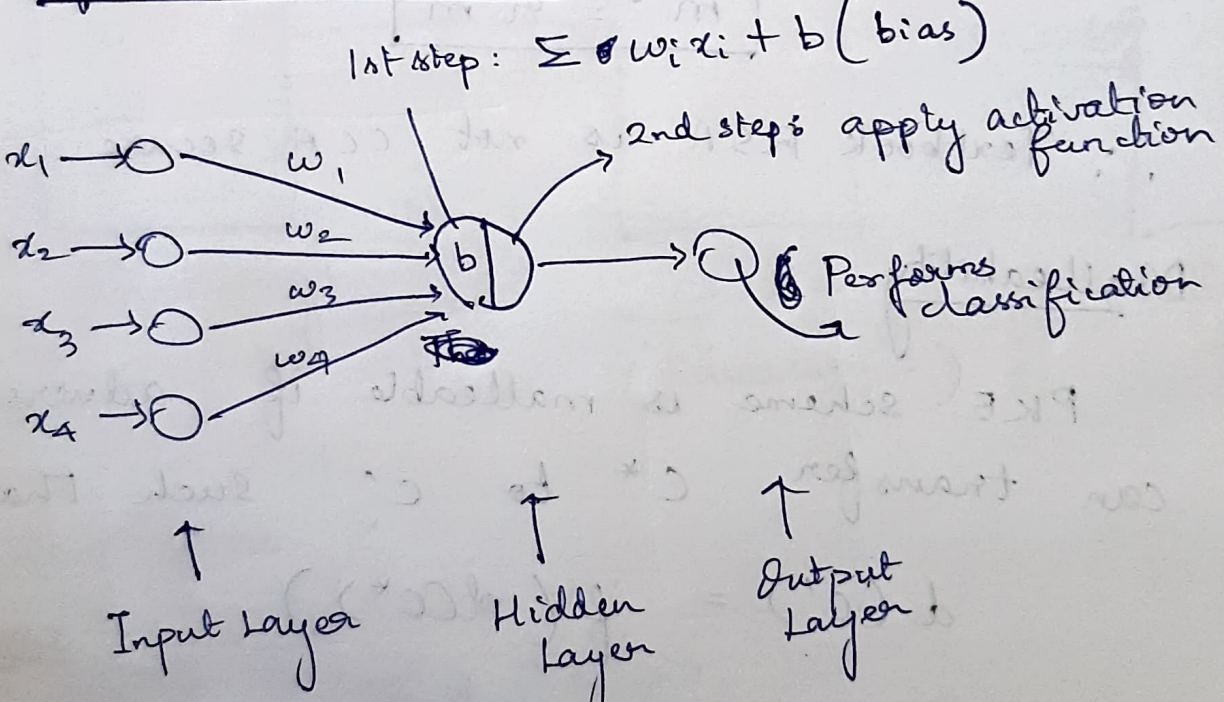
Deep Learning

Perceptron

1. Single Layer perceptron



Artificial Neural Network



Forward propagation

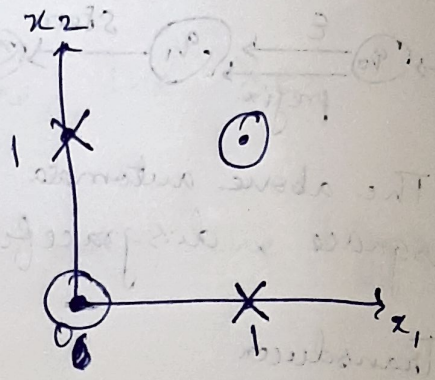
Input → Input Layer → Hidden Layer → Output layer → Output

Deep Learning

Single Layer perceptron	Multi Layer perceptron
1. No hidden layer	Hidden Layer is present
2. No activation function	Activation function
3. Only applicable when data is linearly separable	3. Can be applied for non-linear data.

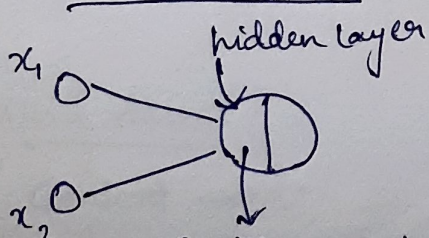
XOR

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



The points cannot be separated by a linear line. This data is not linearly separable.

Activation Function



① $\sum w_i x_i + b$

② Activation function

Types of activation functions:

(i) Linear

(ii) Non-Linear

Linear function

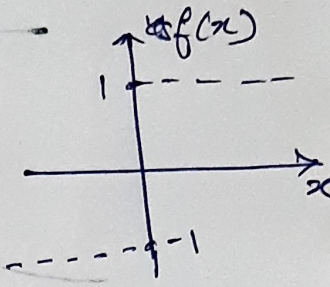
(i) Step function

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

Domain = $(-\infty, \infty)$ Range = $\{0, 1\}$

(ii) Signum function

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



Domain = $(-\infty, \infty)$

Range = $\{-1, 1\}$

(iii) Linear function

$$f(z) = z$$

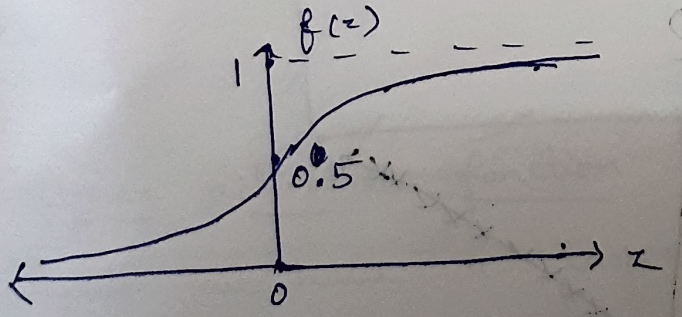
Domain = $(-\infty, \infty)$

Range = $(-\infty, \infty)$

Non-Linear function

(i) Sigmoid function

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



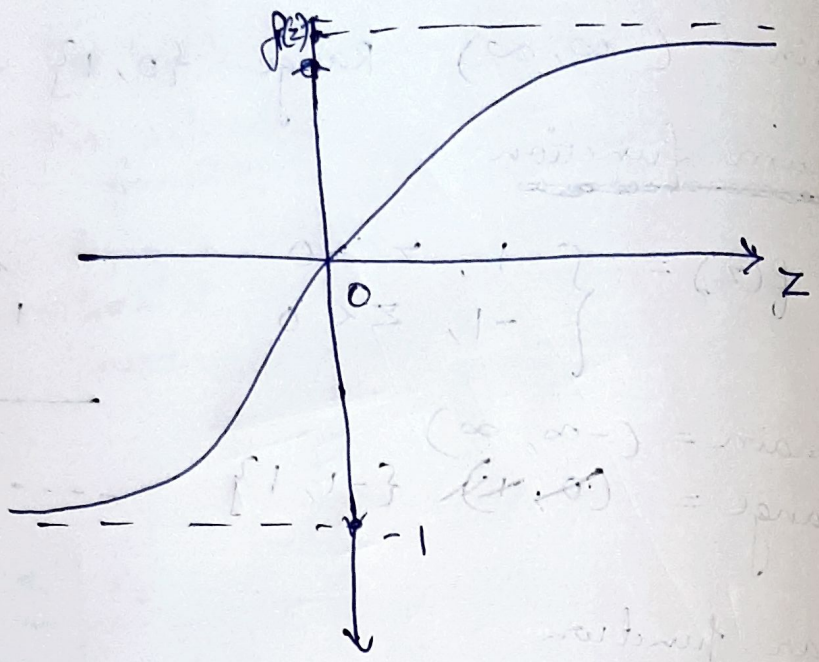
Domain = $(-\infty, \infty)$

Range = $(0, 1)$

Since this function doesn't touch 0, it is called non-zero central activation function.

(ii) tanh activation function / Hyperbolic tangent function

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

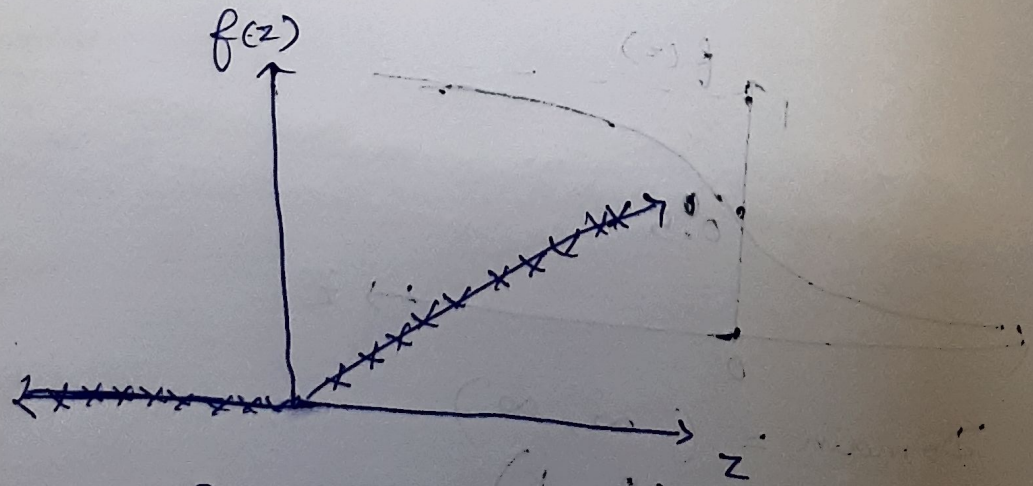


Range = (-1, 1)

Since the graph crosses 0, this is called zero central activation function.

(iii) ReLu (Rectified Linear Unit)

$$f(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

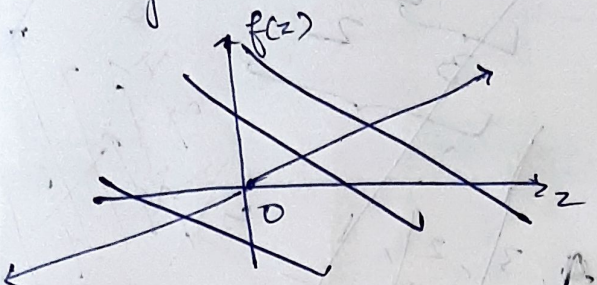


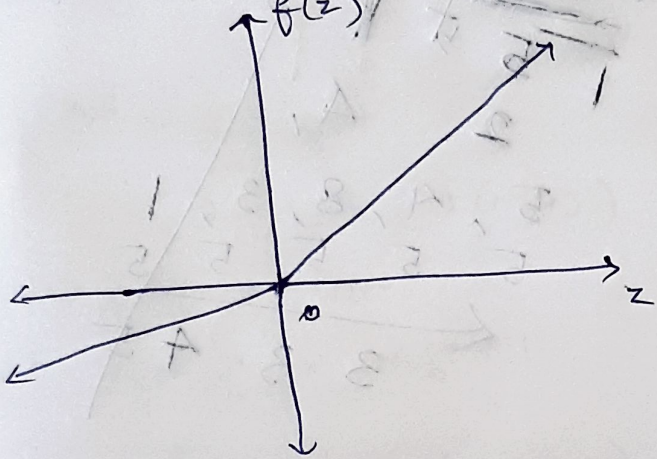
Range = $(0, \infty)$

(iv) Leaky ReLU

$$f(z) = \begin{cases} z, & z \geq 0 \\ az, & z < 0 \end{cases}$$ and a is any constant

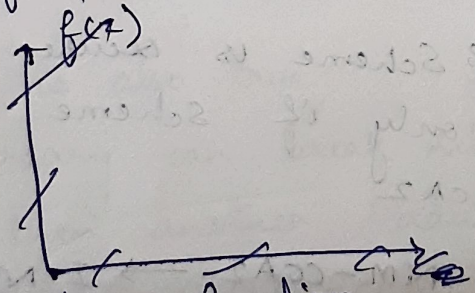
Range = $(-\infty, \infty)$





Note:

For Multi-class classification, we use soft-max ~~function~~ activation function



Softmax activation function

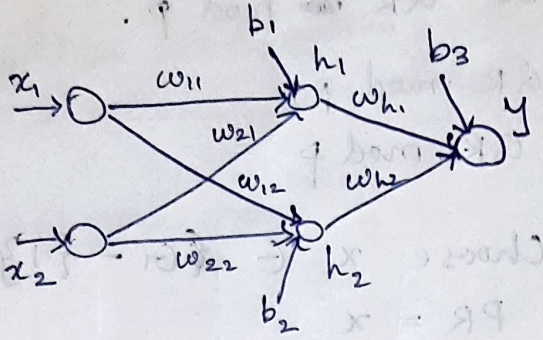
Suppose we have n classes

$$f(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots e^{z_n}}$$

$$\vdots$$

$$f(z_n) = \frac{e^{z_n}}{e^{z_1} + e^{z_2} + \dots + e^{z_n}}$$

Deep Learning



$$(h_1)_{in} = w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1$$

$$(h_1)_{out} = \text{Activation}\,(h_1)_{in}$$

$$(h_2)_{in} = w_{12} x_1 + w_{22} x_2 + b_2$$

$$(h_2)_{out} = \text{Activation}\,(h_2)_{in}$$

$$y_{in} = w_{h_1} * (h_1)_{out} + w_{h_2} * (h_2)_{out}$$

$$y_{out} = \text{Activation}(y_{in})$$

Problem

x_1	x_2	y_{out}
0	0	0
0	1	1
1	0	1
1	1	0

$w_{11} = w_{21} = 2$

$w_{12} = w_{22} = -2$

$[b_1, b_2] = [-1, 3]$

$w_{h_1} = w_{h_2} = 2$

$b_3 = -3$

(0, 0)

$(h_1)_{in} = -1$

$$(h_1)_{out} = \frac{1}{1 + e^{-(h_1)_{in}}} = \frac{1}{1+e} = 0.26 \approx 0$$

$(h_2)_{in} = 3$

$$(h_2)_{out} = 1/1+e^{-3} = 0.95 \approx 1$$

$y_{in} = -1$, $y_{out} = \text{sigmoid}(-1) = 0.26 \approx 0$

For (0, 0), $y_{out} = 0$

(0, 1):

$(h_1)_{in} = 1$, $(h_1)_{out} = 0.75 \approx 1$

$(h_2)_{in} = 1$, $(h_2)_{out} = 0.75 \approx 1$

$y_{in} = 1$, $y_{out} = 0.75 \approx 1$

For (0, 1) $y_{out} = 1$

(1, 0):

~~$(h_1)_{in} = 1$, $(h_1)_{out}$~~

Same as (0, 1)

(1, 1):

$(h_1)_{in} = -1$, $(h_1)_{out} = 0.26 \approx 0$

$(h_2)_{in} = 3$, $(h_2)_{out} = 0.95 \approx 1$

$y_{in} = -1$ $y_{out} = 0.26 \approx 0$

For (1, 1) $y_{out} = 0$

x_1	x_2	h_1	h_2
0	0	0	1
0	1	1	1
1	0	1	1
1	1	1	0

Similar to OR gate

Similar to NAND gate

Use ReLU for this problem:

$w_{11} = w_{12} = w_{21} = w_{22} = 1$

$b_1, b_2 = 0, -1$

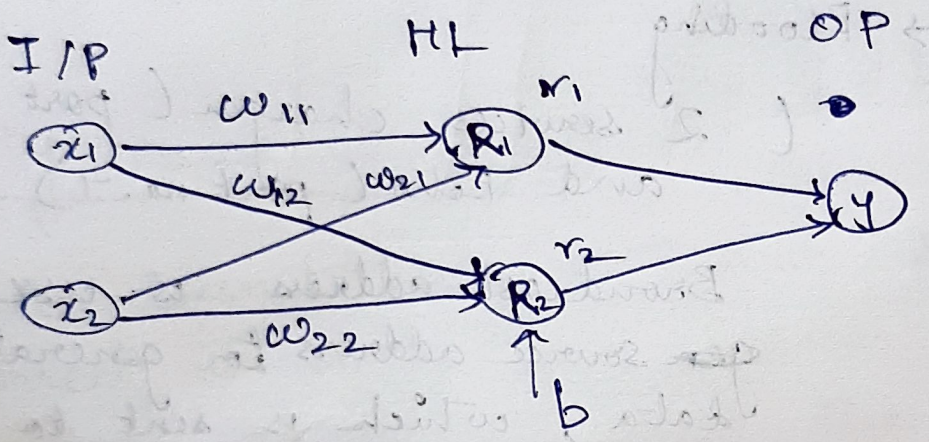
$w_{h1}, w_{h2} = 1, -2$

$b_3 = 0$

x_1	x_2	$(h_1)_{in}$	$(h_1)_{out}$	$(h_2)_{in}$	$(h_2)_{out}$	y_{in}	y_{out}
0	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1
1	0	1	1	0	0	1	1
1	1	2	2	1	1	0	0

Deep learning

Forward Neural Network (Multi-Layer Perceptron)



$y = f(x, \theta)$, where

θ - parameters updated during ~~learning~~ training

$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{21}x_2 + b_1 \\ w_{12}x_1 + w_{22}x_2 + b_2 \end{bmatrix}$$

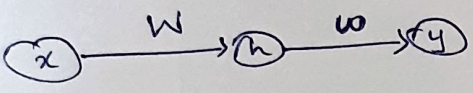
(2×2) (2×1)

$$+ \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

(2×1)

Weight matrix - (no. of layer $\times$ prev. layer)

Representing weights as matrices, we get:



$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$w = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad b = 0$$

$$X = \begin{matrix} x_1 & x_2 \end{matrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{matrix} T \end{matrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$X \times W = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

$$\text{Add } c: ((XW)^T + c)^T = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

Apply Relu,

we get, $H = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$

Input space = $\{(0,0), (0,1), (1,0), (1,1)\}$

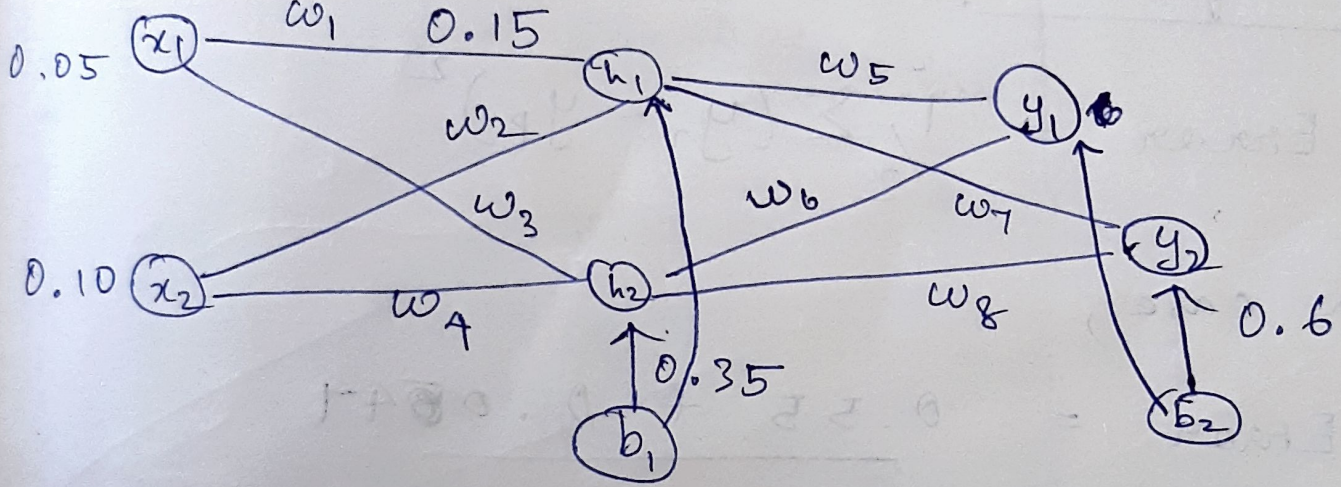
Hidden Layer Space = $\{(0,0), (1,0), (2,1)\}$

$H \times w = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = y$

Note

We are getting target output the same as actual output because, the weights and biases are proper.

Deep Learning



$w_1 = 0.15$ $\quad$ $w_5 = 0.4$

$w_2 = 0.20$ $\quad$ $w_6 = 0.45$

$w_3 = 0.25$ $\quad$ $w_7 = 0.5$

$w_4 = 0.30$ $\quad$ $w_8 = 0.55$

$$[0.05\ 0.10] \begin{bmatrix} 0.15 & 0.25 \\ 0.20 & 0.30 \end{bmatrix} + [0.35\ 0.35]$$

(X) (W) b

$$= [0.0275\ \ 0.0425] + [0.35\ \ 0.35]$$

$$= [0.3775\ \ 0.3925]$$

(h_1) (h_2)

$$\text{Sigmoid}([0.3775\ \ 0.3925]) = [0.5933\ \ 0.5969]$$

(r_1) (r_2)

$$[0.5933\ \ 0.5969] \begin{bmatrix} 0.4 & 0.5 \\ 0.45 & 0.55 \end{bmatrix} = [0.505\ \ 0.624]$$

$$+ [0.60\ \ 0.60] = [1.105\ \ 1.224]$$

$$\text{Sigmoid}([1.105\ \ 1.224]) = [0.7514\ \ 0.7729]$$

y_1 y_2

	Actual values	Predicted values
y_1	0.01	0.7514
y_2	0.99	0.7729

Mean Square Error

$$\text{Error} = \frac{1}{2} \Sigma (y_T - y_P)^2$$

In our case,

$$\text{Error} = \frac{0.55 + 0.0847}{2}$$

$$= 0.32984$$

Error functions

For regression,

$$\text{Mean Square Error} = \frac{1}{2m} \sum (T_i - P_i)^2$$

For binary ~~class~~ classification problem

$$\text{Cross Entropy function} = -[y_i \log a_i + (1-y_i) \log (1-a_i)]$$

Deep Learning

- The objective in training a model is not only to reduce training error but also to minimize generalization error. This is done because models with large generalization error tend to overfit.

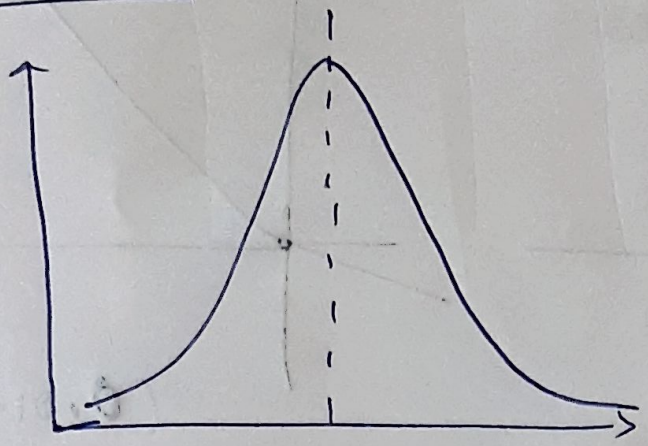
∴ Total cost = Training cost function + Regularization

Common activation functions in Output layer

- Linear
- Sigmoid
- Softmax

Activation function based on different distributions (Output Layer)

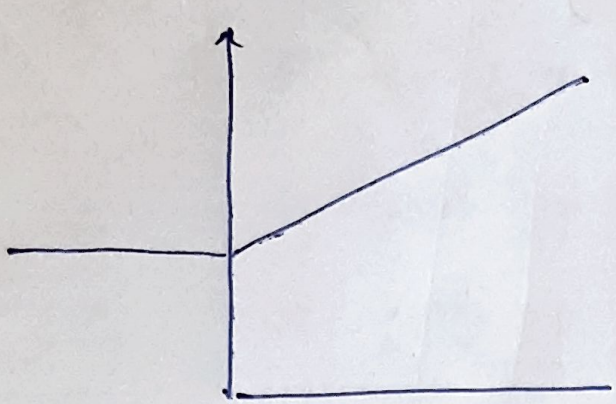
Normal distribution



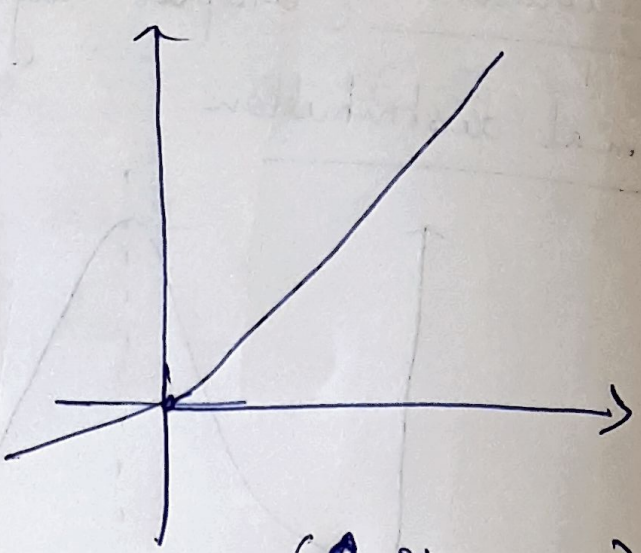
Linear activation function is preferred

Activation function for output layer	Problem to be used for
Linear	Output is continous value (Regression problem)
Sigmoid	Binary class classification
Softmax	Multi class classification

Variants of ReLU

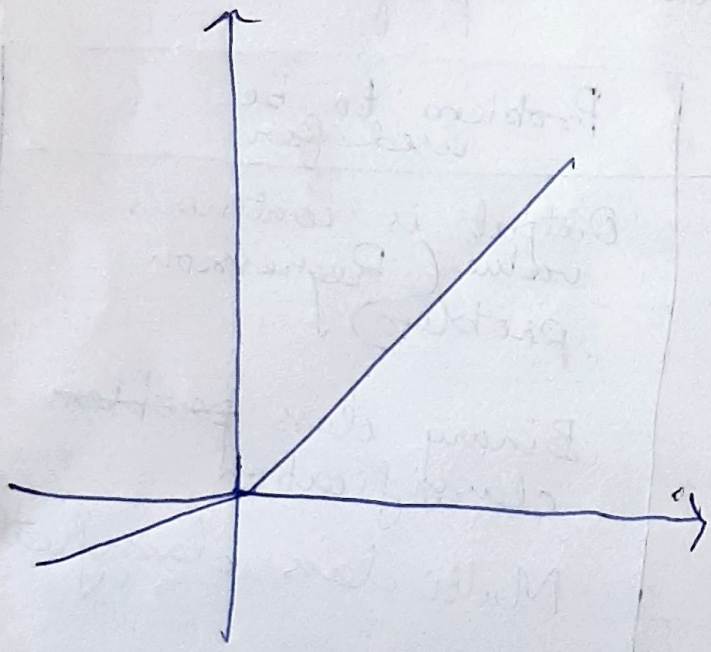


Absolute value rectification



$(0.01x, x)$

Leaky ReLU



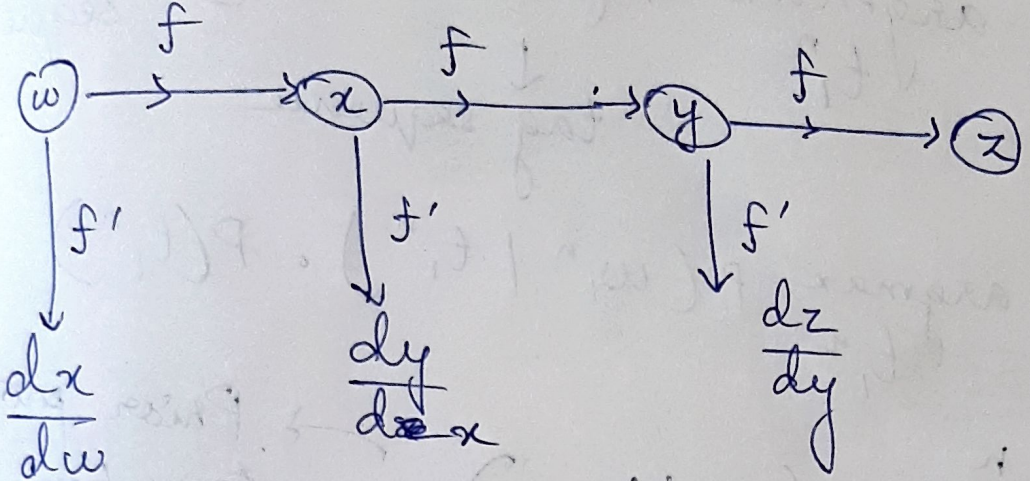
(ax, x)

Parametric ReLU

Maxout

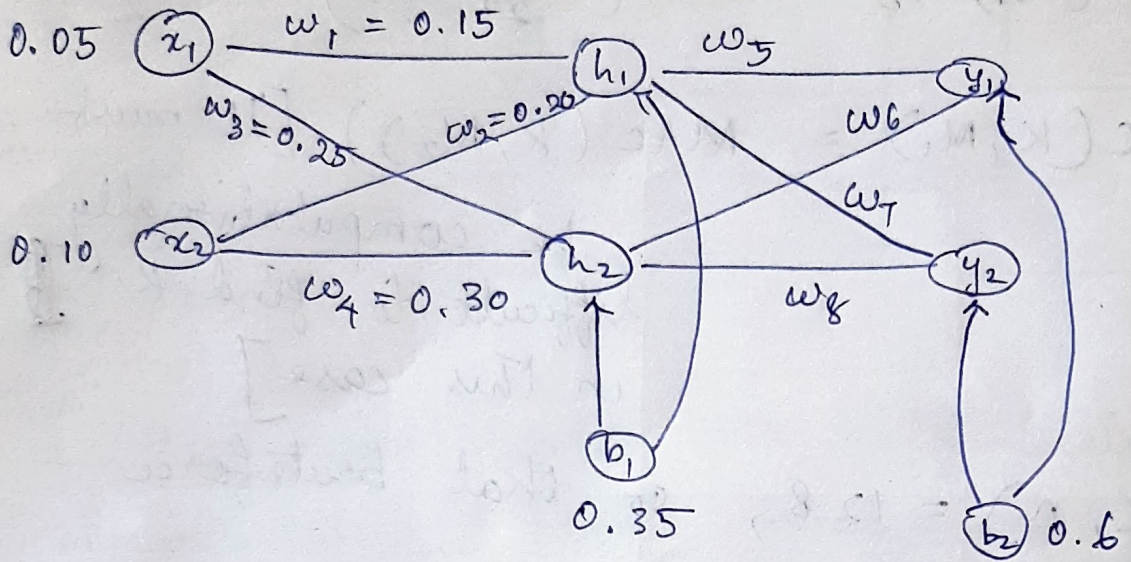
$$\max\left(w_1^T x + b_1, w_2^T x + b_2\right)$$

Deep Learning



$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

$$\frac{dy}{dw} = \frac{dy}{dx} \cdot \frac{dx}{dw}$$



$w_5 = 0.4$

$w_6 = 0.45$

$w_7 = 0.5$

$w_8 = 0.55$

Target values

y_1	y_2
0.01	0.99

Error = 0.2984 (not acceptable)

Deep learning

Update weights :

$$\boxed{w = w - \alpha \nabla} \quad ; \quad \nabla = \frac{\partial \text{Error total}}{\partial w}$$

α - learning rate

$$E_{total} = \frac{1}{2}(T_1 - out\,y_1)^2 + \frac{1}{2}(T_2 - out\,y_2)^2$$

$$\frac{\partial E_{total}}{\partial w_5} = (T_1 - out\,y_1)\frac{\partial out\,y_1}{\partial w_5} + (T_2 - out\,y_2)\frac{\partial out\,y_2}{\partial w_5}$$

$$\frac{d E_{total}}{d w_5} = \frac{d E_{total}}{d out_{y_1}} \times \frac{d out_{y_1}}{d y_1} \times \frac{d y_1}{d w_5}$$

$$\frac{d E_{total}}{d out_{y_1}} = (out_{y_1} - T_1) = 0.7414$$

$$\frac{d out_{y_1}}{d y_1} = \frac{-1 (e^{-y_1})(-1)}{(1+e^{-y_1})^2} = \frac{e^{-y_1}}{(1+e^{-y_1})^2}$$

$$= 0.3312$$

$$= (1+0.3312)^2 = 0.1869$$

$$\frac{d y_1}{d w_5} = out H_1 = 0.5933$$

Reference:

$T_1 = 0.01$, $out_{y_1} = 0.7514$

$T_2 = 0.99$ $out_{y_2} = 0.7729$

$\alpha = 0.5$ $y_1 = 1.105$

$$\frac{d E_{total}}{d w_5} = 0.0822$$

$$w_5 = w_5 - \alpha \frac{d E_{total}}{d w_5}$$

$$= 0.4 - (0.5)(0.0822)$$

$$= 0.3589$$

Regularization

- to avoid overfitting, we use regularization parameter

Note:

Overfitting is a situation where model performs well on training but poorly on testing.

We have many methods of regularization

(i) L2 => $|slope|^2$ is Regularization parameter

(ii) L1 => $|slope|$ " " "

(iii) Data Augmentation

(iv) Early stopping criterion

(v) Parameter sharing

(vi) Dropout

(vii) Boosting & Bagging

Deep Learning

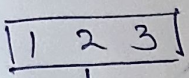
Convolutional Network

Convolution operation is performed by applying a filter to the input.

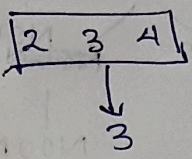
E.g I/P → 1, 2, 3, 4, 5

Filter → 0 1 0

Multiply an element in a window with its respective element in the filter.



$1*0 + 2*1 + 3*0 = 2$



Output = 2, 3, 4

But we are missing the edge elements 1 and 5, so we add padding to the input.

New input is : 0 1 2 3 4 5 0

New output after padding : 1 2 3 4 5

No. of padding to add depends on the dimension (rows/columns) of the filter.

If filter is of size $s \times t$;

No. of rows to be added $= s - 1$

No. of cols to be added $= t - 1$

Note:

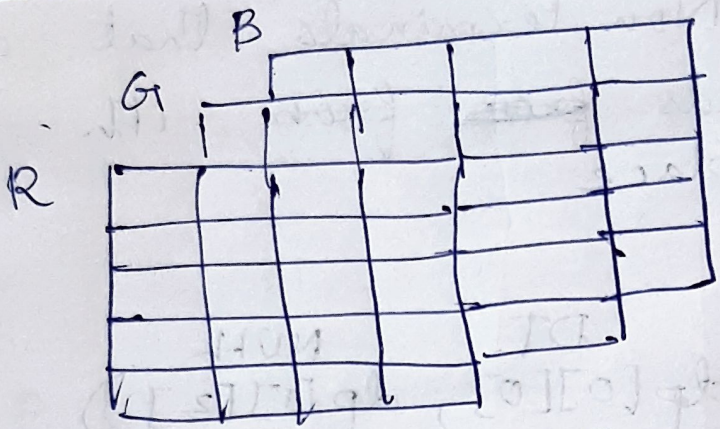
$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ – filter can be used to detect lines in an image.

Case 1:
Single filter & Single image (only 1 output)

Case 2:
Multiple filter & Single image
(as many outputs as filters)

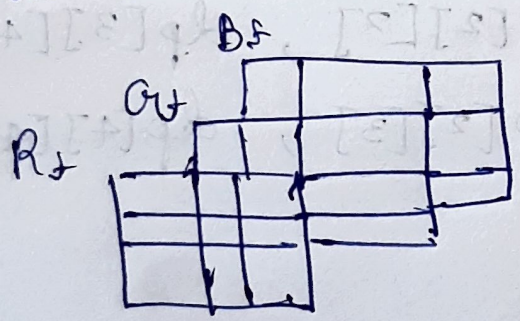
Deep Learning

Case 3 (Multi-Channel Input Image)



m x n x 3

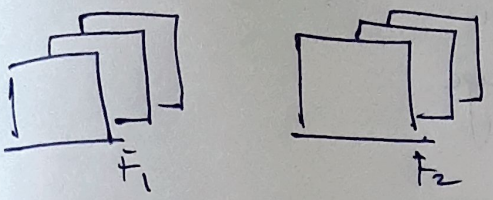
~~Filter with 3~~ 3
We use a 3 channel filter



R_f, G_f, B_f will be applied to R, G, B respectively.

Case 4 (Multiple filter with Multiple Channel)

Multiple filters are applied on a single channel.



F_1, F_2 are separate filters, each of them being Multi-Channel filters. Output will be same as number of filters.

Note:

Output obtained by applying filters ~~are called~~ are nothing but features.

Types of convolution

* Stride
* Transpose
* Dilation

Stride (x, y)

- controls how the filter moves
- parameter x shows how much the filter moves in the horizontal direction
- parameter y ~~is~~ vertical direction

Transpose

- ~~add~~ to get output of greater size than image
- adds padding around the image

Dilation

- adds ~~0~~ zeroes in between pixels
- used for upscaling an image

In CNN, at each Neuron, the following operation is performed:

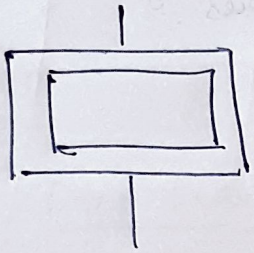
$$\text{Activation}\left(\sum F_i * I_i + b_i\right)$$

F_i - filter
I_i - image

Pooling

- There are 2 types of pooling: Max and Average pooling

Representation of Convolution operation



or