Date. 7-Feb
Page No.

Deep Learning

History of AI with Deep Learning.

1. 1940's → Artificial neuron
2. 1950's → Turing Test
3. 1956 → birth of AI
4. 1957 → Perceptron
5. 1960 → ADALINE
6. 1969 → XOR Problem

First Golden Age

7. 1980 → Neocognitron

First Dark Age

8. 1986 → Back propagation
9. 1989 → UAT
10. 1995 → SVM's

Second golden age

11. 1998 → CNN
12. 2006 → RBM Initialization

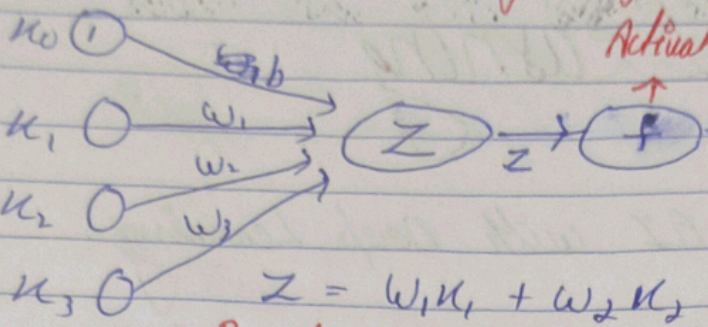
Second dark age

13. 2012 → Alex Net
14. 2014 → GAN
15. 2017 → Transformer
16. 2020 → gpt-3
17. 2022 → chat-gpt

Third golden age

Date.
Page No.

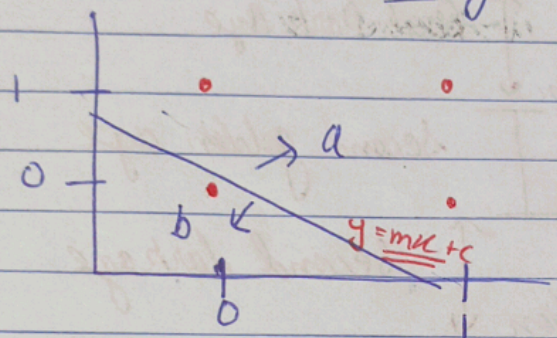
Perceptron → First idea in the field of deep learning.



$$z = w_1x_1 + w_2x_2 + w_3x_3 + 1 \cdot b$$

* Perceptron map your input with a linear functions giving us some linear output we can use for train.
  ↳ this is the weight that it gives us

XOR Problem → ① OR fun → Truth table



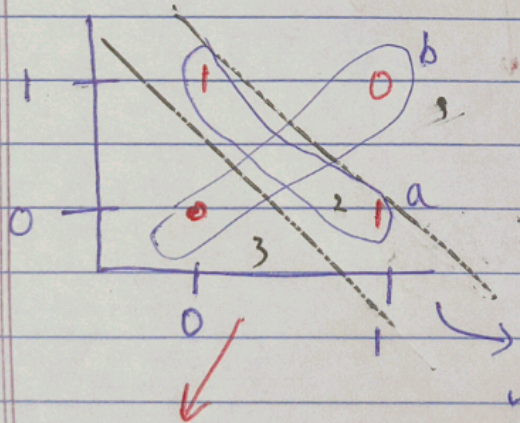
In		out
0	1	1
1	1	1
1	0	1
0	0	0

a, b

$y = mx$ ⇒ Linear line

So How can i draw a linear line to classify data points

② XOR fun → Truth Table

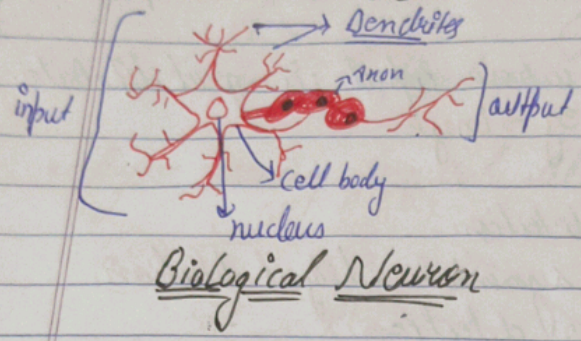


In		out
1	0	1
0	1	1
1	1	0
0	0	0

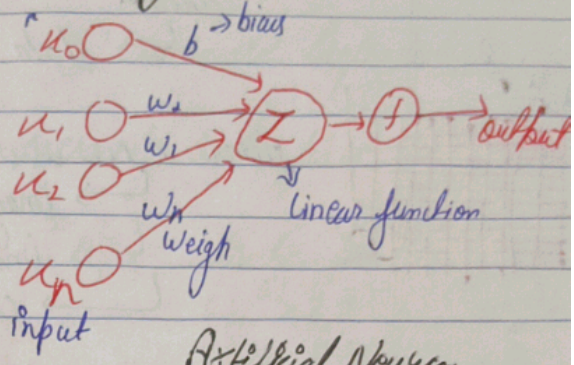
Here linear function is failing, linear fun can divide in two classes only.

if we want to represent a complex problem like XOR we want some complex eq^n

Neural Networks And its types



Biological Neuron



Artificial Neuron

$$z = x_1 w_1 + x_2 w_2 \cdots x_n w_n + b$$

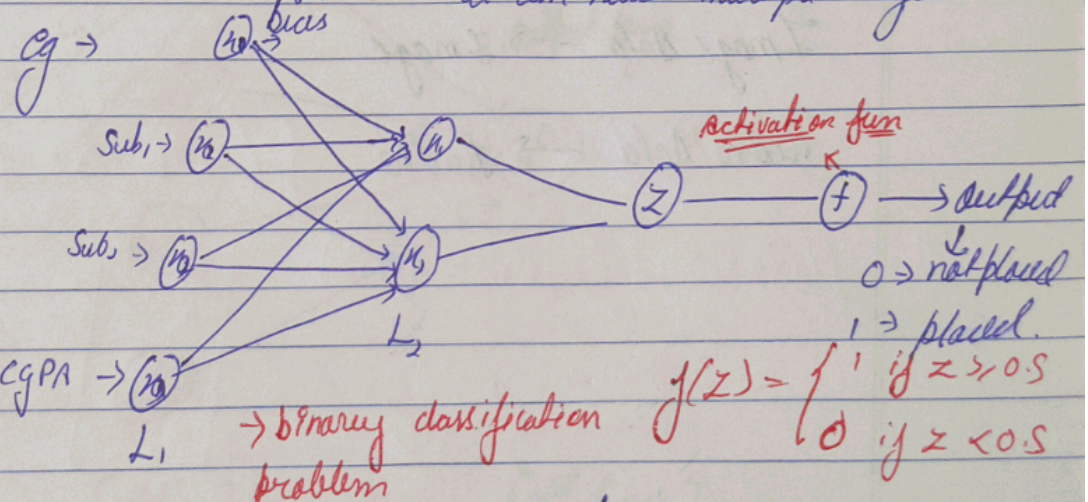
Types of neural network

(1) Feed Forword Neural Network ⇒ FNN or FFNN

Perceptron = Most Basic Feed forward Neural Network

↳ Why basic ?

↳ it can have multiple layer



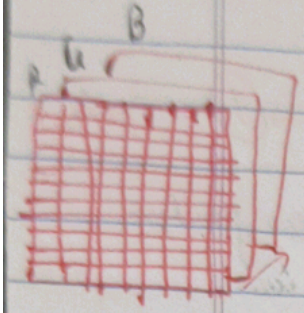
→ binary classification problem

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0.5 \\ 0 & \text{if } z < 0.5 \end{cases}$$

$$z = f(x) + b \quad \begin{cases} x = \langle x_0, x_1, x_2, x_3 \rangle \\ b = x_0 b \end{cases}$$

Date.
Page No.

(2) Convolutional Neural Network (CNN)
↳ special Neural Network to handel complex data
↳ data whose input is grid like data
↳ eg → image



where CNN used
↳ image detection
↳ image segmentation / object detection
↳ cancer detection

(3) Recurrent Neural Network (RNN)
↳ good for sequential data
↳ eg → audio, text, speech, Time series
Transformers are also a type of RNN → multilayered RNN

(4) GAN ⇒ Generative Adversarial Network
↳ used to generate synthetic / Artificial Data

Image Data → Image

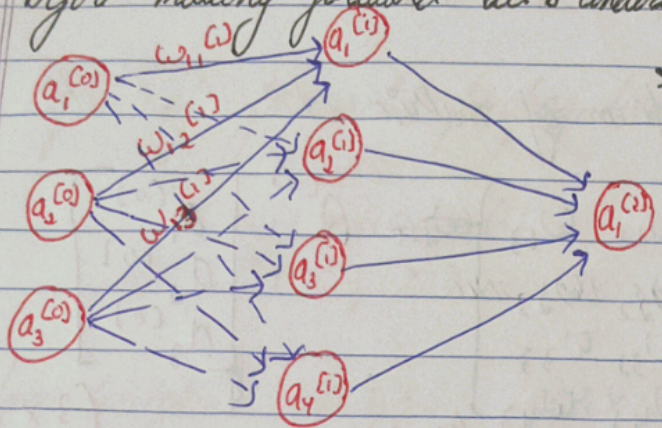
Audio Data → Audio

Date.
Page No.

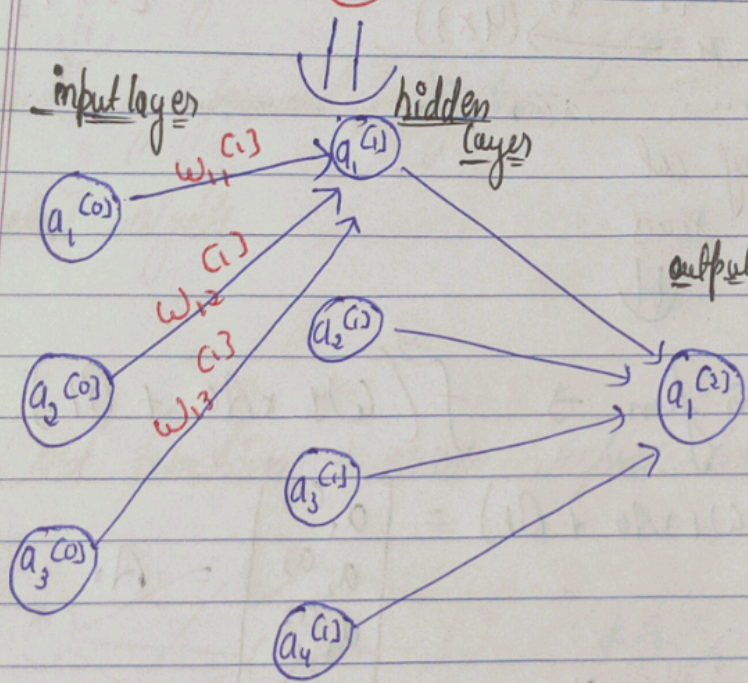
• Learning Mechanism of Neural Network

① Forward Propagation ② Backward Propagation

• Before moving forward let's understand basics first :→



$a_1^{(1)}$ = activation fun($w_{11}^{(1)} \times a_1^{(0)} + w_{12}^{(1)} + a_2^{(0)} w_{13}^{(1)} + a_3^{(0)} + B_0$)



0th layer [1st layer] [2nd layer]

[1] → next layer to which weighted edge is connected

$w_{\square}$ → neuron no. of next layer

$\square$ → neuron no. of previous layer.

$a_1^{[1]}$ → representation of neuron

[1] → layer no.

1 → neuron no.

A_0 ω

$$a_1^{(1)} = \text{activation\_function}\left(\begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \end{bmatrix} \times \begin{bmatrix} \omega_{11}^{(1)} \\ \omega_{12}^{(1)} \\ \omega_{13}^{(1)} \end{bmatrix} + B_1 \right)$$

⇓

This can also written as

$$a_1^{(1)} = \omega \times A + B$$

ω can be written in form of matrix

$$\omega_1 = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{21} & \omega_{22} & \omega_{23} \\ \omega_{31} & \omega_{32} & \omega_{33} \\ \omega_{41} & \omega_{42} & \omega_{43} \end{bmatrix}_{(4\times 3)} \quad A^{(0)} \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \end{bmatrix}_{[3\times 1]}$$

n_y (rows) ↓ , n_x (columns) →

n_x → no. of col

n_y → no of rows

⇓

Activation fun → $f(\omega_1 \times A_0 + B_1)$

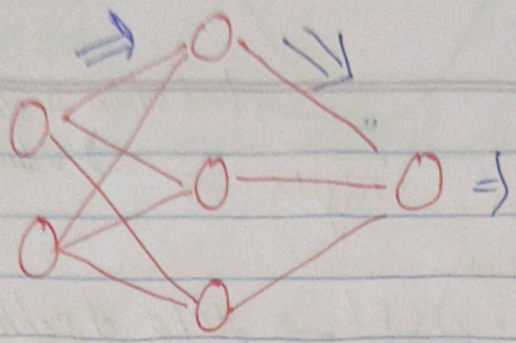
$f(\omega_1 \times A_0 + B_1) = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \end{bmatrix} = A_1$

⇓

$$A_1 = f(\omega_1 \times A_0 + B_1)$$
$$A_2 = f(\omega_2 \times A_1 + B_2)$$

Here ω_i → (ω_i is matrix containing all weights associated with ~~first layer~~ i^{th} layer)

A_j → (matrix containing neurons of j^{th} layer)



Forward propogation

- Here we are going forward by giving random weight at start to updating it from an activation fun.
- then we calculate loss/cost

→ This is also called forward propagation

Eq^{n} of forward propogation

$Z1 = W1 * A0 + B1$ $\quad$ $Z2 = W2 \times A1 + B2$

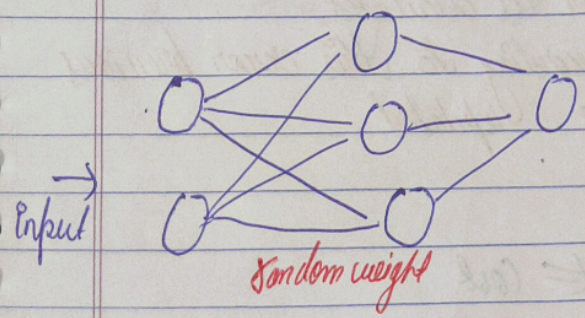
$A1 = f(Z1)$ $\quad$ $A2 = f(Z2)$

Random weights = Trained using forward propogation = Trained weights.

$Z3 = W3 \times A2 + B2$

$A3 = f(Z3)$ → output prediction

Cost function ⇒ error representation



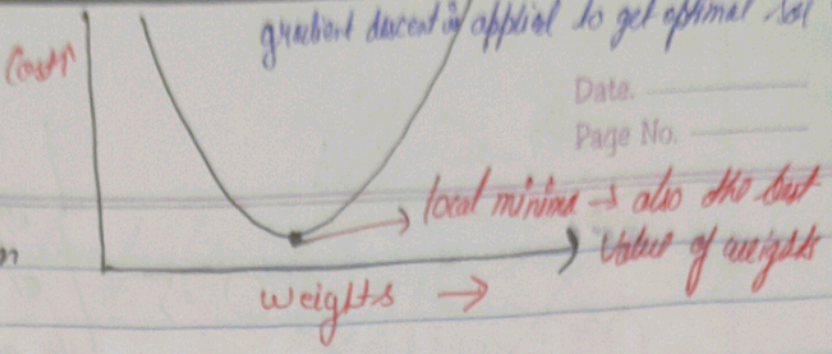
Δ = Actual output − Predicted output

↓

Cost fun

- CF ↑ → output → wrong
- CF ↓ → output → correct

We minimise CF to get best output.



graph of cost fun

$$w' = w - \alpha \cdot \frac{\partial CF}{\partial w}$$

$\frac{\partial CF}{\partial w}$ → slope of graph

α ↓ learning rate

So our Random weights gets updated to optimal weights using gradient ~~to~~ descent.

formulae for gradient descent

Repeat {

$$w' = w - \alpha \cdot \frac{\partial CF}{\partial w}$$

$$B' = B - \alpha \cdot \frac{\partial CF}{\partial B}$$

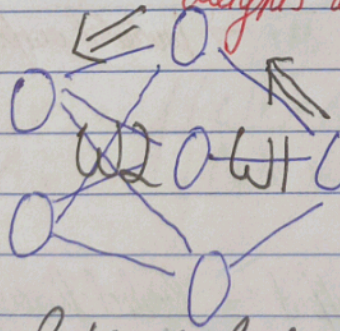
}

Here How weights are updated?

↳ ~~to~~ they are updated in backward dir^n.

① → Cost/error is calculated

② → Then ~~#~~ according to this error previous weights are updated.



Backward Propogation

Date.
Page No.

How A neural network is trained

① → initialise random weights
② → Forward Propogation

$$Z = W1 \times A0 + B1$$
$$A2 = f(Z)$$

③ → find the cost / error

Δ = pred output − actual output

~~$\Delta = \alpha \cdot \frac{\partial cost}{\partial w}$~~

④ Backward Propogation

$$W' = W - \alpha \cdot \frac{\partial Cf}{\partial W}$$
$$B' = B - \alpha \cdot \frac{\partial Cf}{\partial B}$$

⑤ → Repeat ③ & ④ until we get optimal Sol"

1 epoch = Step1 + Step2 + Step3 + Step4
for training we do multiple epochs

Date.
Page No.

* Activation functions :-

$Z1 = W1 \times A0 + B1$

$A1 = f(Activation\_fun(z1))$

if let's say this function is linear and not changing anything but the same old input

$Z1 = W_1 \times A0 + B1$

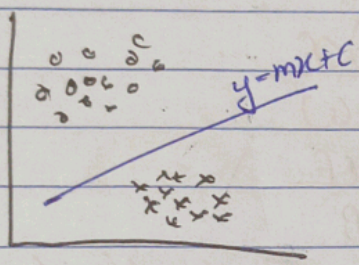
$A1 = f(z_1)$

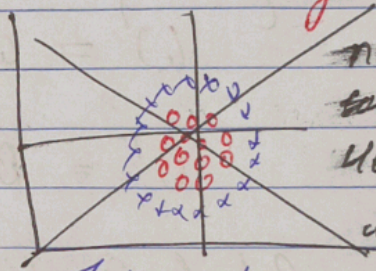
$Z2 = W2 \times A1 + B2$

$A2 = W2 \times (W1 \times A0 + B1) + B2$

$A2 = W2 \times W1 \times A0 + W2 \times B1 + B2$

W_1 ; B' This is again a linear eq^n.



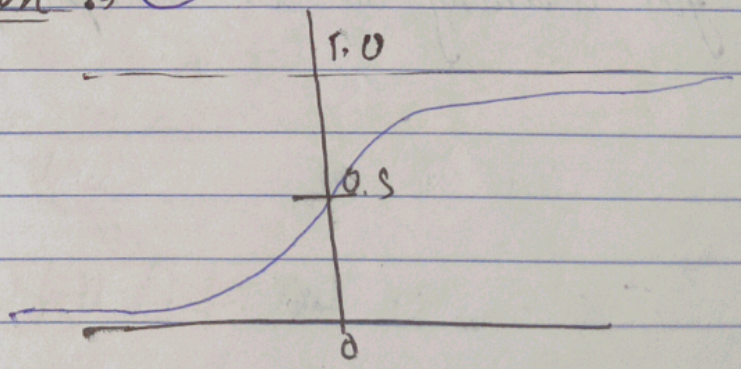


no line can be useful Here if its a linear.

That's why we need a non-linear activation fun.

① Sigmoid Function :-

$$y = \frac{1}{1+e^{-x}}$$

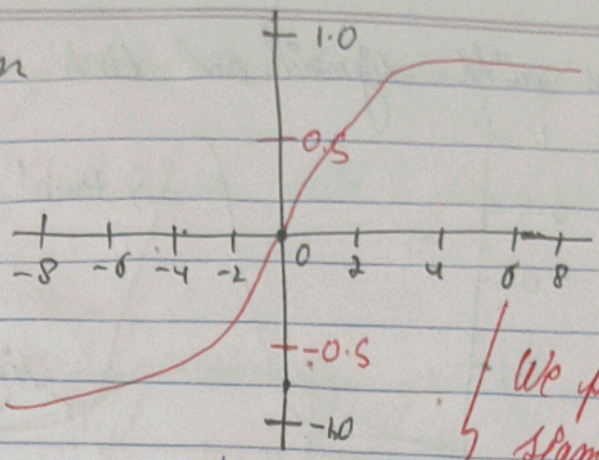


Used :-

- Binary Classification

Date.
Page No.

② Tanh function

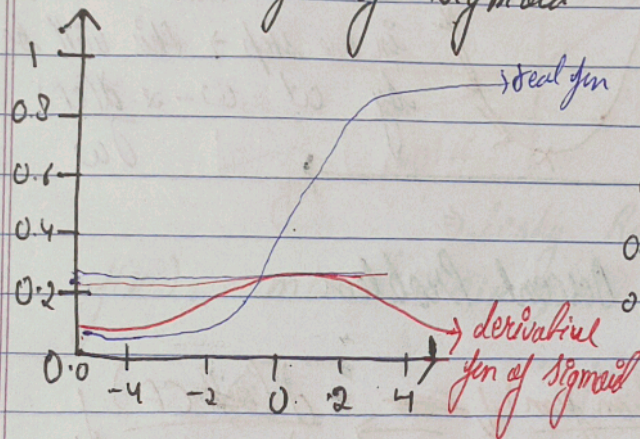


$$\text{tanh fun } (y) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$

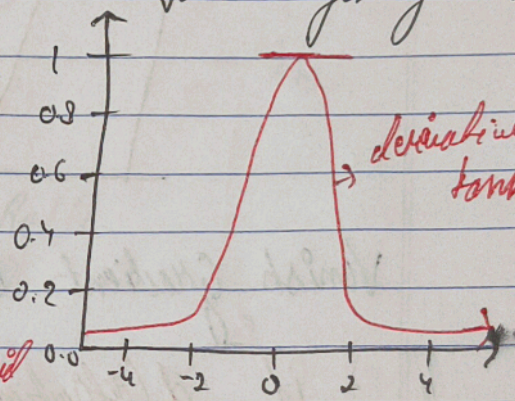
We prefer tanh over sigmoid in the hidden layer because:
↳ while back propagation → we do derivation derivative of (cost fun) w.r.t (w)

So we also need derivative of Activation fun as well ← So the cost fun depends on ← Activation fun.

Derivative fun of Sigmoid



Derivative fun of tanh



peak value of derivative of Sigmoid ≈ 0.2
" " " " " tanh ≈ 1

$$w' = w - \alpha * \frac{\partial CF}{\partial w}$$

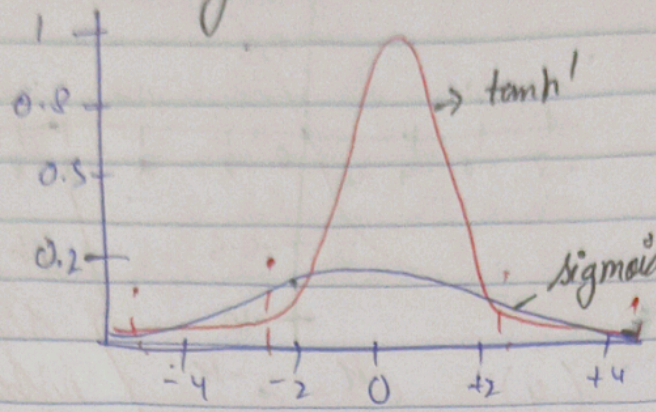
⇒ Here derivative value gets multiplied by the learning rate

Why tanh over Sigmoid

- Centered around zero.
- So here (tanh fun) is more useful for fast learning

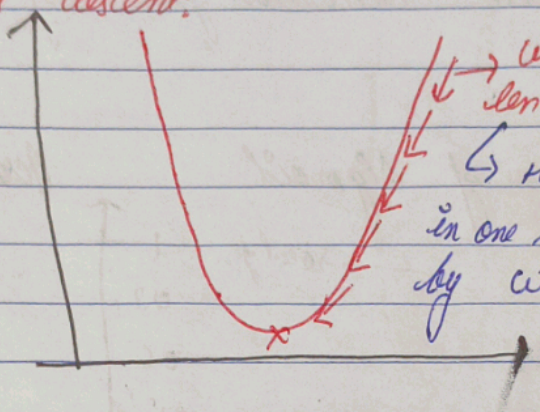
Date.
Page No.

Problems with sigmoid and tanh



Problem with these function is as the value of x increases → the ~~slow~~ derivative of both fun get almost zero

↳ this arises a problem of vanishing gradient descent.



→ what will determine the len of arrow?

↳ How much to descent in one step → this will be tell by $w' = w - \alpha \cdot \frac{d(cf)}{dw}$

Vanish Gradient Descent Problem

if $\frac{d(\text{activation fun})}{dw}$ ↓ ⟹ $\frac{d(\text{~~act~~ cf})}{dw}$ ↓

(very low) (also very low)

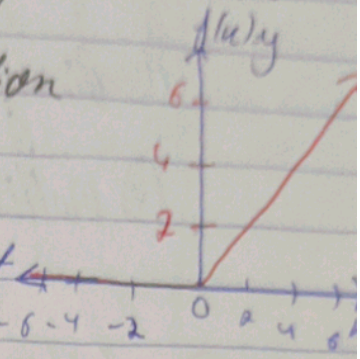
↳ So we will move very slow to the minima

↳ This is for both sigmoid and tanh function

Solution for gradient descent vanishing gun is ReLU

(3) ReLU → (Rectified Linear Unit) function

$f(x) = \max(0, x)$



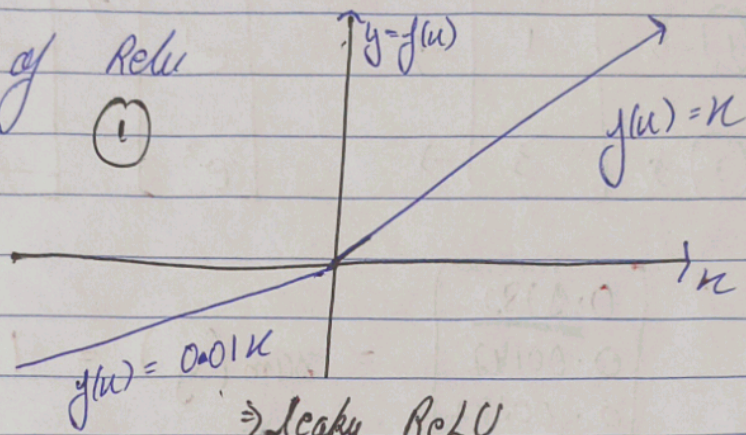
overcome the problem of vanishing gradient

$$\frac{\partial f(x)}{\partial x} = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

- This function is piece wise linear
  - ↳ gives advantage of both linear - non linear functions.

Variants of Relu

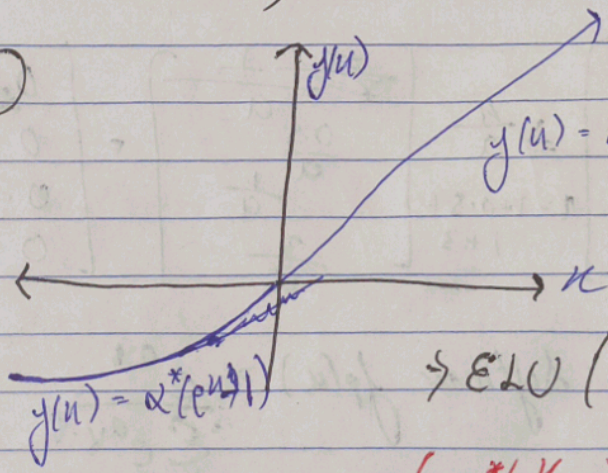
(1)



⇒ Leaky ReLU

$f(x) = \max(0.01x, x)$

(2)



→ ELU (exponential linear unit)

$f(x) = \max(\alpha * (e^x - 1), x)$

Date.
Page No.

Which one to use when

① Binary classification → Sigmoid

② Multiclass classification → ? Here is one more fun.
↳ SoftMax

④ SoftMax function

$a = e^7 + e^{0.5} + e^1 + e^3$

$$\Rightarrow \begin{bmatrix} 7 \\ 0.5 \\ 1 \\ 3 \end{bmatrix} \Rightarrow e^x \begin{bmatrix} e^7 \\ e^{0.5} \\ e^1 \\ e^3 \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{e^7}{e^7+e^{0.5}+e^1+e^3} \\ \frac{e^{0.5}}{a} \\ \frac{e^1}{a} \\ \frac{e^3}{a} \end{bmatrix}$$

$$y = \begin{bmatrix} 0.9782 \\ 0.00147 \\ 0.00247 \\ 0.01791 \end{bmatrix} = \text{sum}(y) = 1$$

$$\begin{bmatrix} 7 \\ 0.5 \\ 1 \\ 3 \end{bmatrix} \Rightarrow \frac{x}{a} \quad a = 7+0.5+1+3 \quad \begin{bmatrix} \frac{7}{a} \\ \frac{0.5}{a} \\ \frac{1}{a} \\ \frac{3}{a} \end{bmatrix} = \begin{bmatrix} 0.6086 \\ 0.0435 \\ 0.0861 \\ 0.2608 \end{bmatrix}$$

$$\text{Softmax } f_i(x) = \frac{e^{a_i x}}{\sum_k e^{a_k}}$$

Date
Page No.

(3) House price pred.

-> Here output can be 10 to 1000000 any number

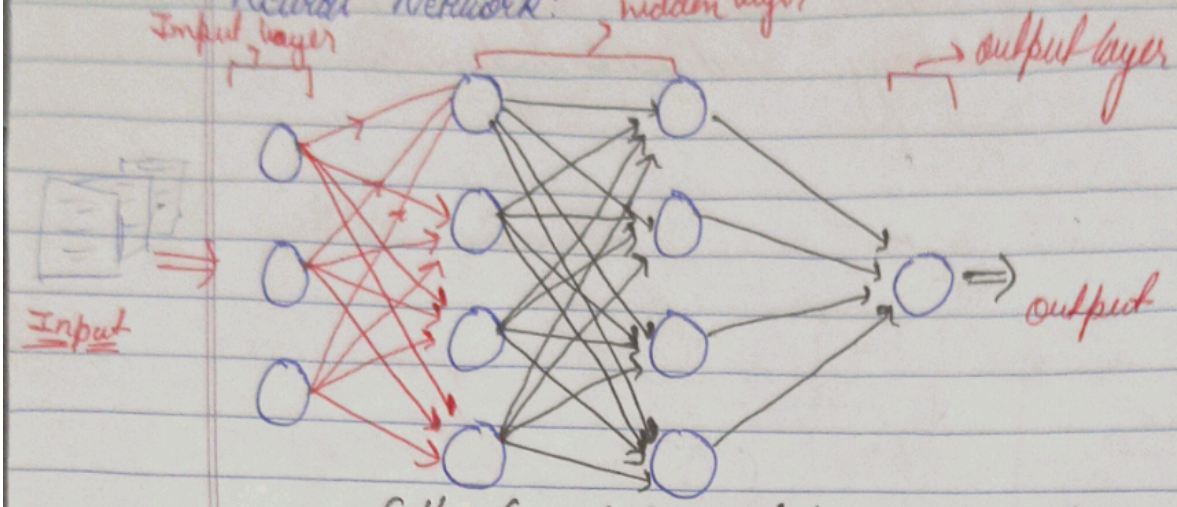
-> So output neuron can not any activation function

In case of linear regression output neuron must not use ~~some~~ any activation function

Date.
Page No.

Multi Layer Neural Network

• Neural Network: Input layer, hidden layer, output layer



Fully Connected multilayer NN looks like

What is neuron? → it is a function that takes input and gives output for that

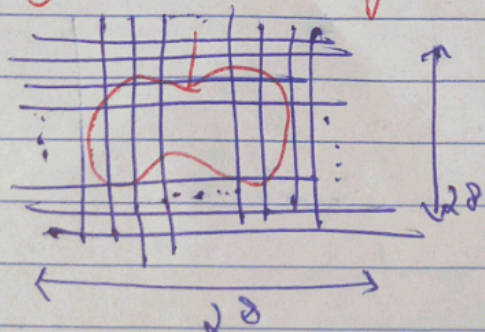
Input → (f(w)) → output

* Neurons are also called as activation functions.

Note → Most of the learning happens in hidden layers

How to know the no. of neurons to be used in input layer.

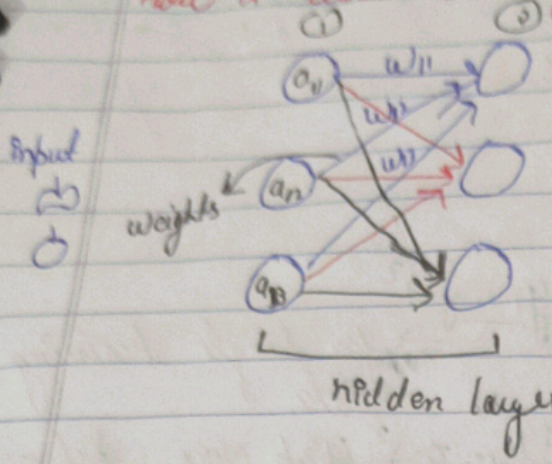
Basic thinking (i) → No. of Neuron = No. of features



- 28 x 28 x 3
= 2352

Date.
Page No.

How & what is happening in hidden layer.



1st layer is responsible for identifying the shape

2nd layer is responsible for identifying the color.

How many layers ? ? { More Neuron = More layer = More Calculation = more time = More Complex problem }

Connection b/w neurons are called as weight

$a_{21} = \{w_{11} \times a_{11} + w_{12} \times a_{12} \times w_{13} \times a_{13} + b\}$ ↳ bias

Here we have two values that model learns { weight, bias }

$a_{21} = func\{w_{11} \times a_{11} + w_{12} \times a_{12} + w_{13} \times a_{13} + b\}$

Loss Function & Cost function

$x \rightarrow y \quad \hat{y} \rightarrow$ model's output

Independent variable

$y \rightarrow$ actual output

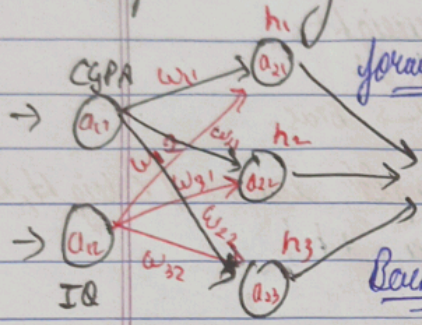
In Linear reg. we have $\Rightarrow$ error / loss $= (y_i - \hat{y})^2$ → for single training data.

$$\text{Cost fun} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y})^2$$

High error = bad model trained

Low error = good model trained.

In deep learning



forward propagation

$h_1 / a_{21} = \omega_{11} \times a_{11} + \omega_{12} \times a_{12} + b$

$h_2 / a_{22} = \omega_{21} \times a_{12} + \omega_{22} \times a_{22} + b$

$h_3 / a_{23} = \omega_{31} \times a_{23} + \omega_{32} \times a_{23} + b$

Backward propagation

These are linear fun.

Motive of training $\Rightarrow$ minimise the cost/cat function.

↳ "We cannot improve what you can't measure"

Types of loss fun. for regression problems

(1) MSE $\Rightarrow \sum_{i=1}^{n} (y_i - \hat{y})^2$ • Not robust to outlier

(1) easy to interpret $= \frac{d(y - \hat{y})^2}{dx} = \left\{ 2 \frac{dy}{dx} - \frac{d\hat{y}}{dx} \right\}$

(2) Differentiable — used in gradient descent

(3) Only one local minima

Date.
Page No.

② MAE $= \sum_{i=1} |y_i - \hat{y}|$

↳ more robust to outlier

↳ Not differentiable

③ Huber Loss → Combo of MAE & MSE $e = y - \hat{y}$

↳ differentiable

↳ robust to outlier

$$L = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \le \delta \\ \delta(|e| - \frac{1}{2}\delta) & \text{if } |e| > \delta \end{cases}$$

δ = threshold

Loss/cost function types for Classification problem.

① for binary classification

① Binary Cross entropy

also called as log loss in logistic reg.

$$\text{Loss fun} = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

→ it can be easily differentiate

② for multiple classes. (we use Softmax as ~~out~~ activation fun in ~~a~~ output neuron

① Categorical Cross entropy

$$\text{Loss} = \sum_{j=1}^{K} y_j \log(\hat{y}_j)$$

③ Hinge Loss ⟶ used for SVM

↳ [0, 1]

$$L = \max(0, 1 - y \cdot f(x)) \qquad w^T \cdot x + b = \text{model out.}$$

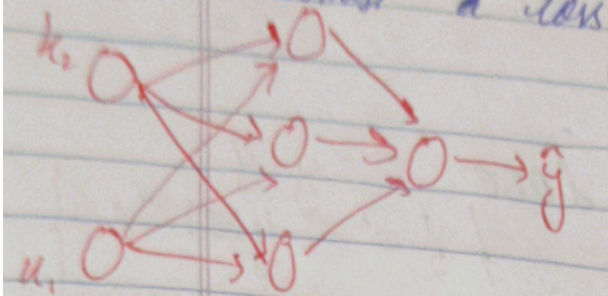
if $(y \cdot f(x) \ge 1)$ → Loss = 0 ⟶ confidently correct

if $(y \cdot f(x) > 0$ & $y \cdot f(x) < 1)$ Loss > 0 → Correct output but need improvement

if $(y \cdot f(x) < 0)$ → wrong pred.

Date
Page No.

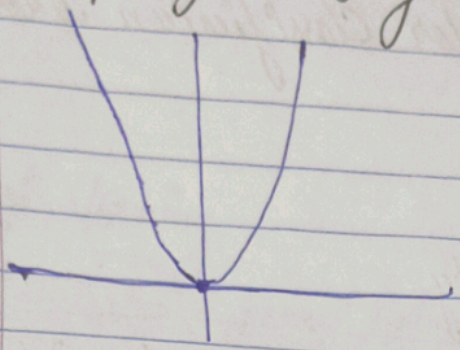
- **Optimizers** → optimizers are algorithm that adjusts model's parameters during training to minimise a loss function and improve models performance.



$$\text{Loss} = (y - \hat{y})^2$$

$$\begin{cases} h_1 = w_1 u_1 + u_2 w_2 + b_1 \\ h_2 = w_3 u_1 + u_2 w_3 + b_2 \end{cases}$$

Optimizers → ① Gradient Descent



$$w_{new} = w_{old} - \delta \frac{\partial loss}{\partial w_{old}}$$

$$b_{new} = b_{old} - \delta \frac{\partial loss}{\partial b_{old}}$$

train →
- 1 epoch ⇒ taking entire data and training for one (w & b)
- 1 epoch = 100 itteration
- itteration ⇒ smaller version of epoch. it happens for one single datapoint

n itteration ⇒ 1 epoch

n epochs ⇒ 1 batch

Gradient Descent ⇒ it take entire dataset & loss as input and gives better (weights and bias)

Concerns ⇒ ① Computationally expensive.
② Higher training time

Date.
Page No.

(2) SGD → Stochastic Gradient Descent

~~1000 Record~~
~~100 = Batch~~
~~↳ 10 Iteration = 1 Epoch.~~

in case of GD ↓
We propagate Back the loss for each iteration of data.

But in SGD → if we have ~~1000~~ SGD ~~but~~ we send only one dp per iteration & we calculate loss for that and update weight vector.
↳ for each iteration we update weight vector
↓
but it not solve the problem of GD

(3) mini-batch Gradient Descent
1000 records & 100 Batch Size
$\frac{1000}{100} \Rightarrow$ 10 iteration = 1 Epoch
after every batch the value of weights updated. ~~not~~ not like GD where it update for entire data set and not like SGD where it updates after single data points
↓
solving problem of GD & SGD

But these GD are also have a problem that they are not very smooth to reach minima
for that ↓ we get exponential average
↓
momentum optimizer
↓
Smothening of descent

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$

$0 < \beta < 1 \hookrightarrow \beta = 0.9$

④ (Adaptive Gradient Boosting) Adagrad

$$w_{new} = w_{old} - \eta \frac{dL}{dw_{old}}$$

$\eta = 0.01$ (fixed before training)

Here $\eta \rightarrow$ adaptive (when far from minima η is large
when close to minima η is small)

⑤ ADAM → most effective in industry.

↳ maximum used in real time in industry

ADAM ↳ momentum + RMS propagation

Better generalizability.

Note: in every optimizer is in the updates of parameters.

↳ aim is same to reach the minima as soon as possible.