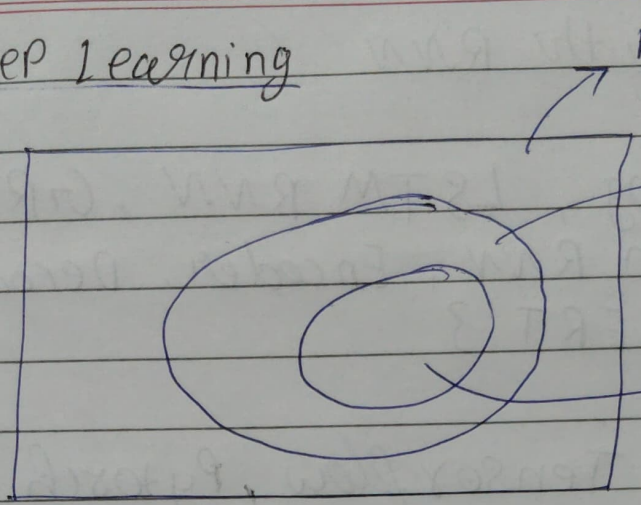
classmate
Date 25/3/25
Page

Deep learning



(1) ANN => Artificial Neural Network → Classification
→ Regration.

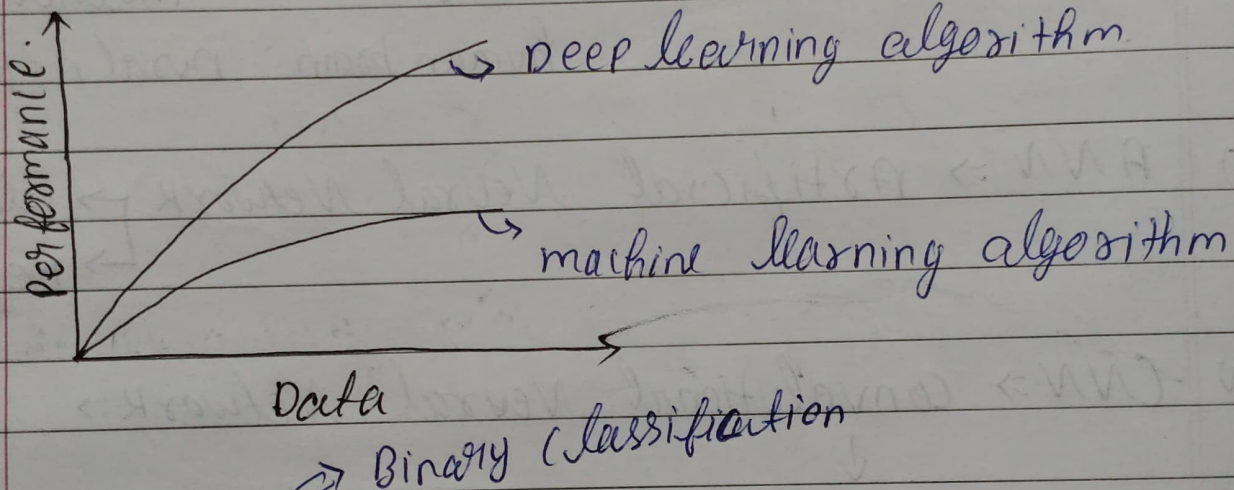
(2) CNN => convolutional Neural Network =>
↓
I/P : Image / video frames → RCNN, masked RCNN, Yolo
⇓
Computer vision and object detection.

(3) RNN => Recorrent Neural Network => N2P, Time-serise
Text ti Time serie ⇐ I/P

components under the RNN

{ Word Embadding, LSTM RNN, GRU RNN
Bi direction LSTM RNN, Encoder, Decoder.
Transformer, BERT }

framework => Tensorflow, Pytorch



=> Binary Classification

Perceptron: => A Single Neural Network. or a Neural network unit.

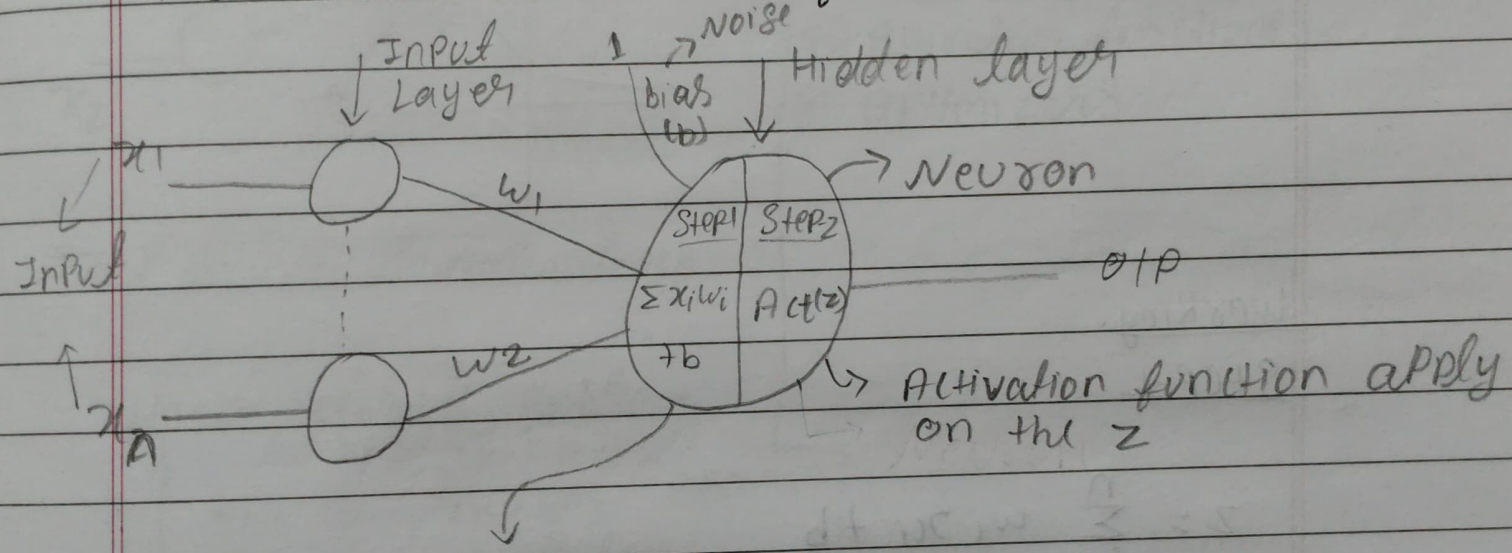
it consist 4 things:

① Input layer
② Hidden layer
③ weights
④ Activation function.

Dataset:

IQ $\rightarrow x_1$	no of Study hr $\rightarrow x_2$	O/P: Pass/fail 0 → fail 1 → Pass
95	3	
110	4	
100	5	1

A Simple architecture of Perceptron.



$$z = w_1 x_1 + w_2 x_2 + b$$

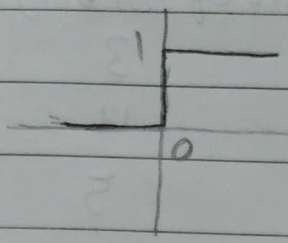
$$z = \sum_{i=1}^{n} w_i x_i + b$$

Activation function: Activation function is basicaly transform the output b/w some value like 0 to 1, -1 to 1

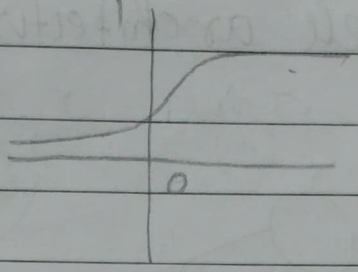
classmate
Date
Page

Activation function.

① Step function:



② Sigmoid function:



Working...

Step-1

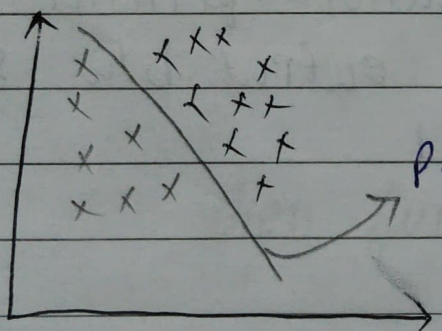
$$z = \sum_{i=1}^{n} w_i x_i + b$$

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

$$y = mx + c$$

$$y = \beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n + \beta_0$$

↳ linear problem statement.

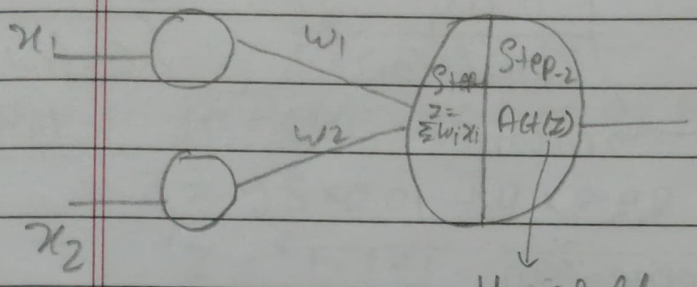


perceptron is a linear classifier.

[ANN]

Perceptron model

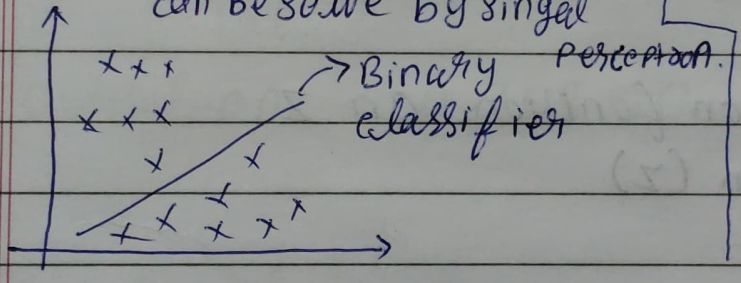
Single layered Perceptron



Threshold

Feed forword Nural Network

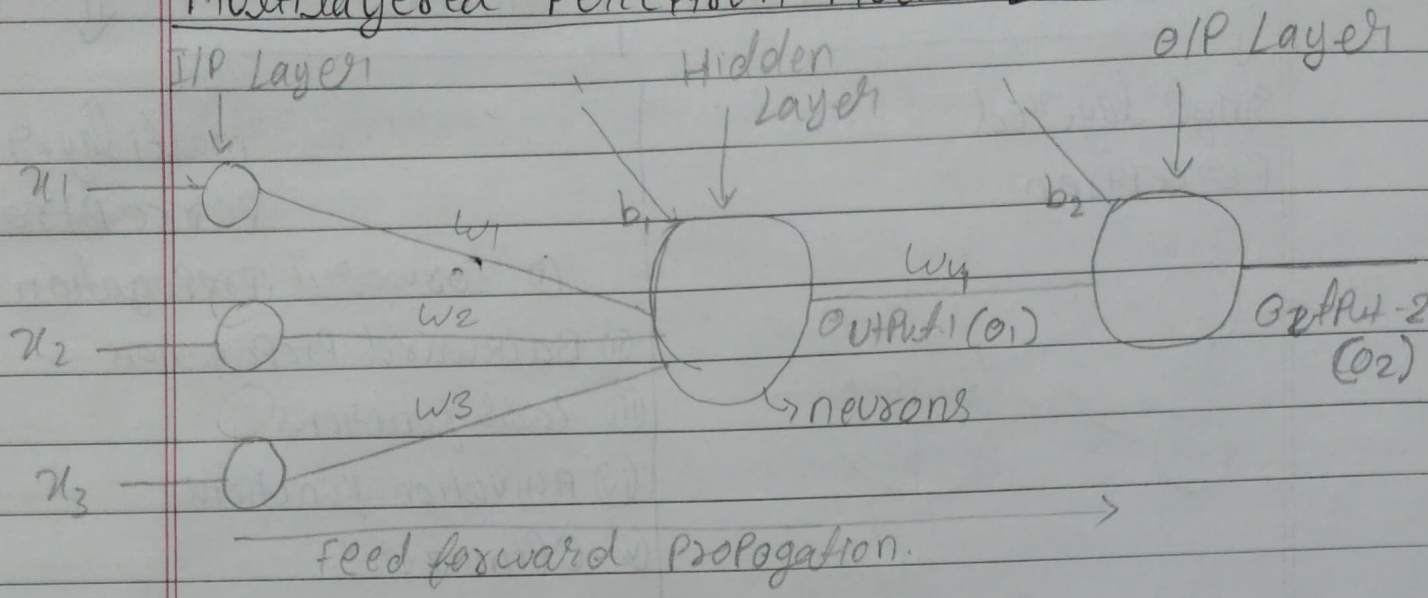
Linear Seperable Problem.
can be solve by singal perceptron.



multi layered perceptron

(i) Forweard propogation
(ii) Backward Propogation
(iii) Loss functions
(iv) Activation functions.
(v) Optimizers.

Multilayered perceptron model [ANN]



Step-1: $Z = w_1x_1 + w_2x_2 + w_3x_3 + b_1$

$$Z = \sum_{i=n}^{n} w_i^T x_i + b$$

Step-2: Activation function on Z

Activation (Z)

Data Set

IQ	Study hr	Play hr	O/P
95	4	4	1 → Pass
100	5	2	1
95	2	7	0 → fail

(1) Forward Propegation:

Let $w = [0.01, 0.02, 0.03]$ (w_1, w_2, w_3), $b = [0.001]$

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

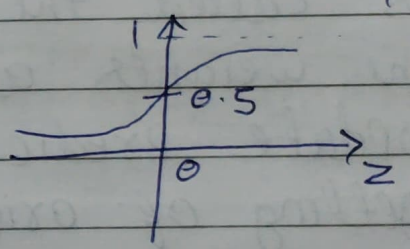
Step-1 in Hidden neuron or Hidden Layer-1

$z = 95 \times 0.01 + 4 \times 0.02 + 4 \times 0.03 + 1 \times 0.01$

$z = 1.151$

Step-2: Activation(z) Sigmoid $= \frac{1}{1+e^{-z}}$

$f(z) = \frac{1}{1+e^{-1.151}} = 0.759$



output-one $= 0.759 = O_1$

Hidden layer-2

$w_4 = 0.02$

$b_2 = 0.03$

Step-1

$z = O_1 * w_4 + b_2$

$= 0.759 \times 0.02 + 0.03$

$= 0.04518$

Step-2 Activation $(z) = \frac{1}{1+e^{-(0.04518)}} = 0.51129 = O_2$

O_2 or Predicted o/p $(\hat{y}) = 0.51129)$

Loss function = error = (original - predicted)

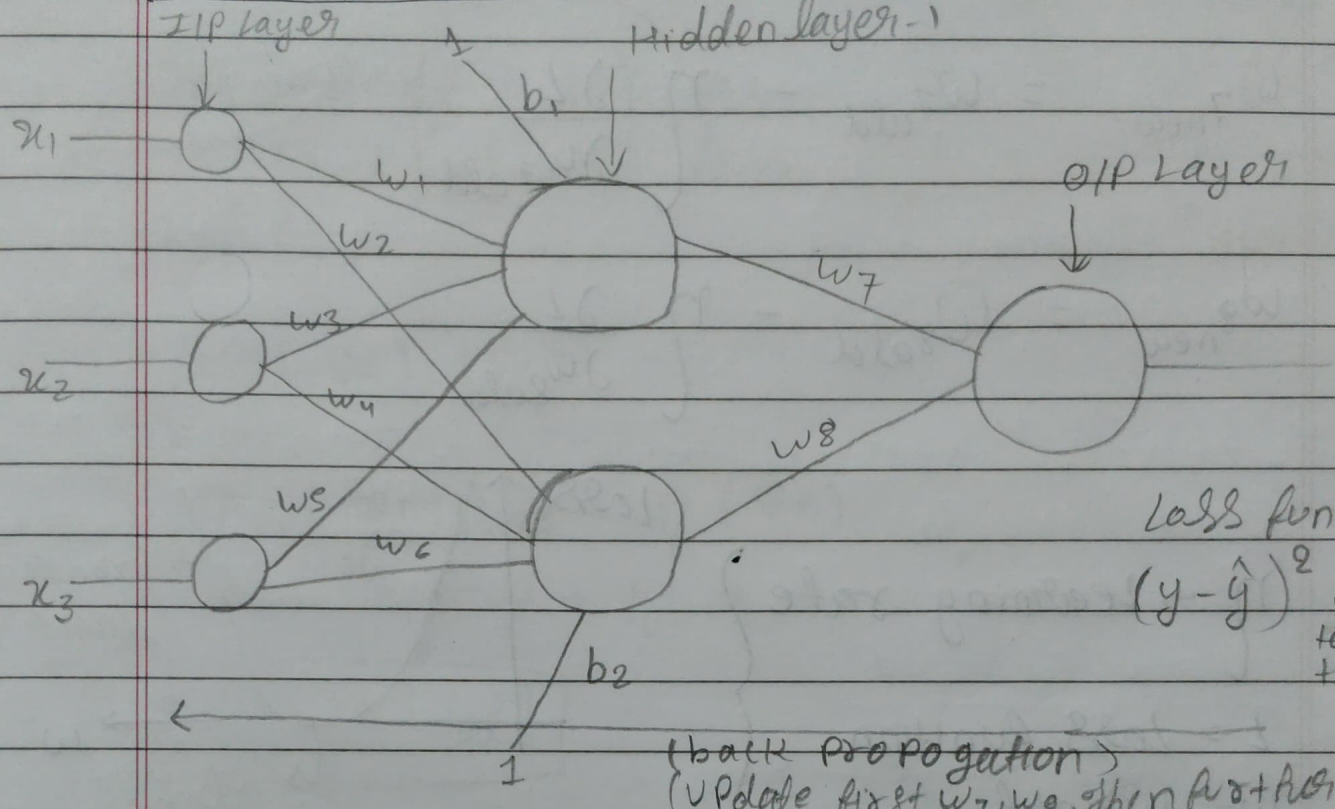
$= (1 - 0.51129)$

$= \approx 0.49$

⇒ So this error is very high so we have to reduce it.
So the way to reduce the loss function we have to update the weights so this is called the backword propogation. because the weights are use predict and again update predict and so on until the getting of original output.

After the first input the weights are rearranged and then the second input is comes to the pictures and then it is again going the forword propogation and calculate the loss function if the value is not minimum the back propogation and then the repeatation is going on.

Back propogation and weight updation formula.



Loss function
$(y - \hat{y})^2$ ↓↓ to reduce the loss

(back propogation)
(Update first w_7, w_8, then b_2 & b_1.

this looks as 3x2 matrix 3 inputs and 2 Hidden layers.

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \end{bmatrix}$$

Some loss functions:

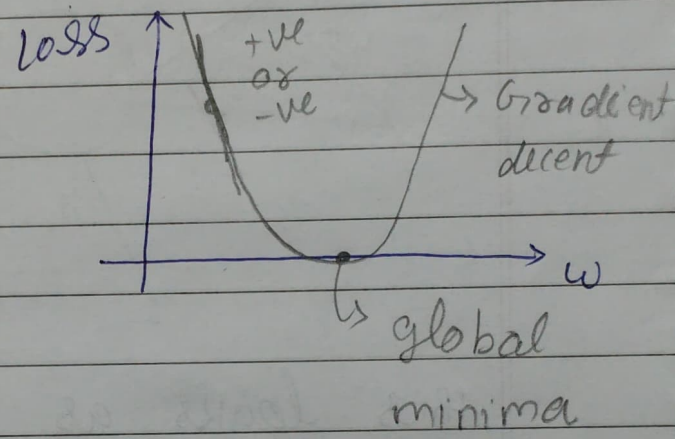
Regression	classification
(i) MSE	(i) Binary cross entropy
(ii) MAE	(ii) categorical cross entropy
(iii) Hubers loss	

weight updation formula:

$$w_{7_{new}} = w_{7_{old}} - \eta \left(\frac{\partial L}{\partial w_{7_{old}}} \right)$$

(η → learning rate; $\frac{\partial L}{\partial w_{7_{old}}}$ → slope)

$$w_{8_{new}} = w_{8_{old}} - \eta \frac{\partial L}{\partial w_{8_{old}}}$$



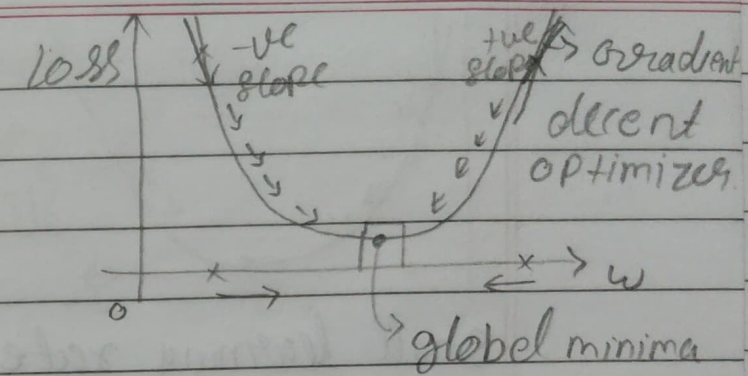
- η → Learning rate
- L = Loss function
- w = weight

$$w_{new} = w_{old} - \eta \left(\frac{\partial L}{\partial w_{old}} \right)$$

↳ weight updation formula.

classmate
Date
Page

Optimizers:



The roll of optimizers is to reduce the loss value.

for (-ve) slope

$$W_{new} = W_{old} - \eta(-ve)$$
$$= W_{old} + \eta(+ve)$$

$$\boxed{W_{new} >> W_{old}}$$

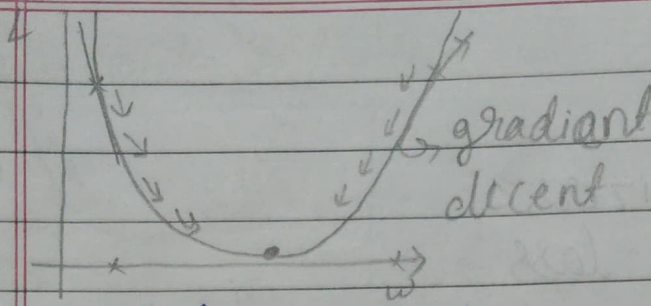
for (+ve) slope

$$W_{new} = W_{old} - \eta(+ve)$$

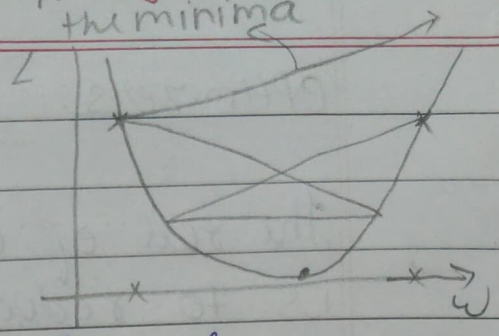
$$\boxed{W_{new} << W_{old}}$$

Learning rate: So basically learning rate is decide the step size how fast it is achive the global minima.

So always take the small learning rate.



small learning rate
$\eta = 0.01$



High learning rate
η = large value.

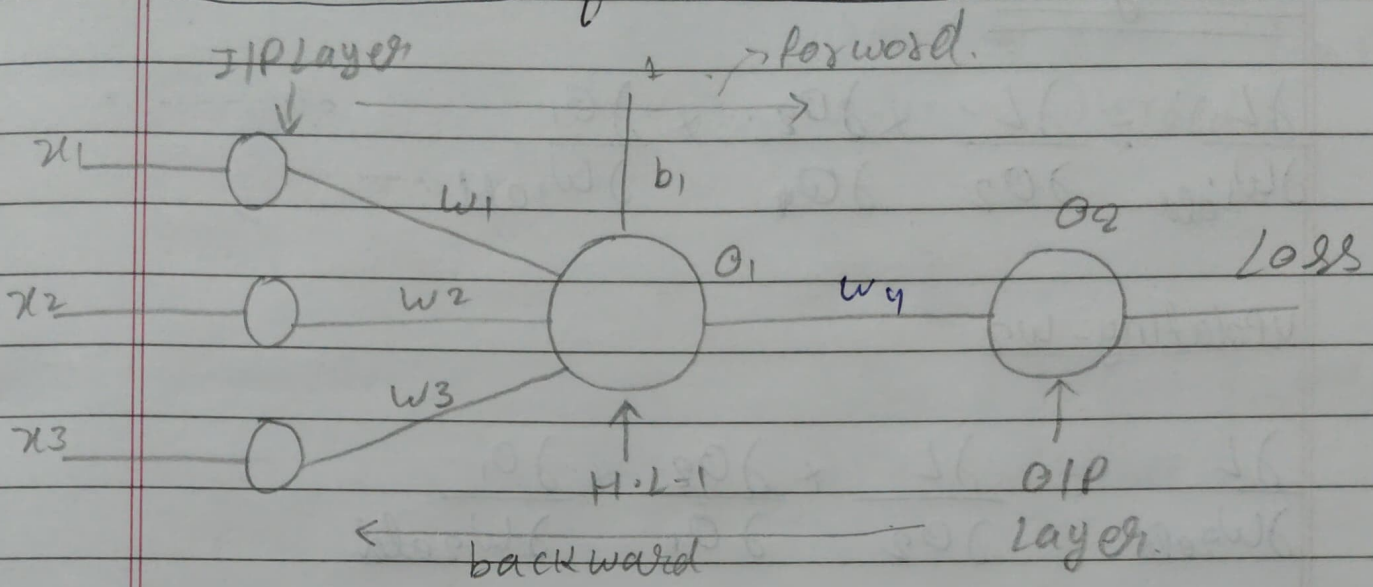
when W reaches the global minima

$$W_{new} = W_{old}$$

no need to update the weight.

it will zost never converge to get the minima and reduce the loss.

Chain Rule of Derivative.



$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

Updating w_4:

$$w_{4new} = w_{4old} - \eta \frac{\partial L}{\partial w_{4old}}$$

$$\frac{\partial Loss}{\partial w_{4old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial w_{4old}}$$

=> chain rule of derivation.

↳ Explanation: Loss is depend on the O_2 (output-2) and the O_2 (output-2) is depend on the w_4.

Updating - w_1

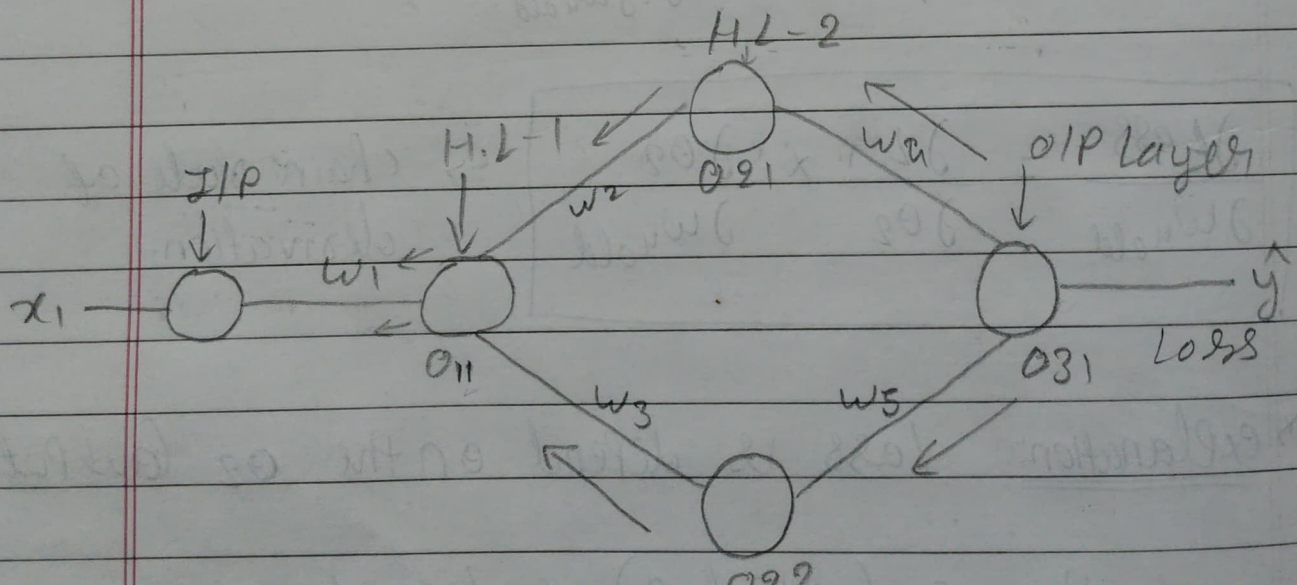
$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial o_2} \times \frac{\partial o_2}{\partial o_1} \times \frac{\partial o_1}{\partial w_{1old}}$$

Updating - w_2

$$\frac{\partial L}{\partial w_{2old}} = \frac{\partial L}{\partial o_2} \times \frac{\partial o_2}{\partial o_1} \times \frac{\partial o_1}{\partial w_{2old}}$$

Updating - w_3

$$\frac{\partial L}{\partial w_{3old}} = \frac{\partial L}{\partial o_2} \times \frac{\partial o_2}{\partial o_1} \times \frac{\partial o_1}{\partial w_{3old}}$$



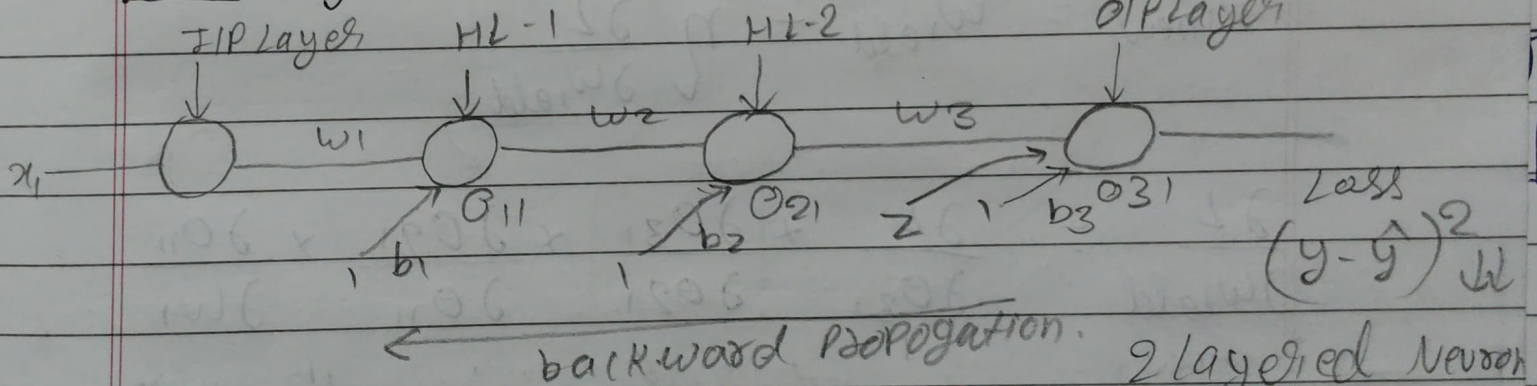
Updating - w_1

$$\frac{\partial L}{\partial w_{1old}} = \left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{1old}}\right] \rightarrow 1^{st} \text{ side}$$

$$+$$

$$\left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{1old}}\right] \rightarrow 2^{nd} \text{ side.}$$

Vanishing Gradient problem And Activation function.



2 layered Neuron

first step in every neuron is

$$\{z = w_3 \times o_{21} + b_3\}$$

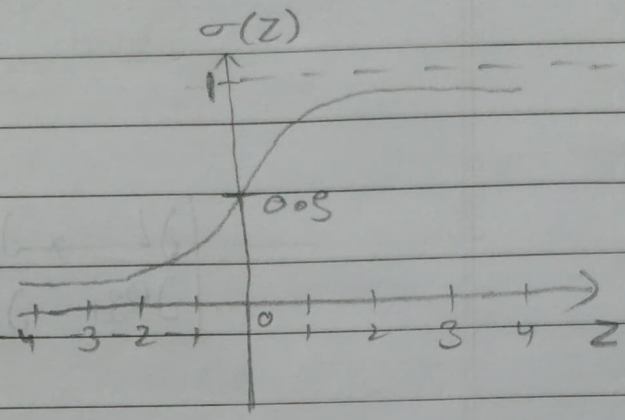
$$z = \sum_{i=1}^{n} w_i x_i + b$$

and second step to apply the activation function.

Sigmoid activation function (σ)

$\sigma(z)$ to convert the value between 0 to 1

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



updating the w_1

$$w_{1new} = w_{1old} - \eta \left[\frac{\partial L}{\partial w_{1old}}\right]$$

values 0.01

$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial o_{31}} \times \left[\frac{\partial o_{31}}{\partial o_{21}}\right] \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_1}$$

for o_{31} $\Rightarrow$

$$o_{31} = \sigma(w_3 \times o_{21} + b_3)$$

$w_3 \times o_{21} + b_3$ = z

$$z = o_{21} \times w_3 + b_3$$

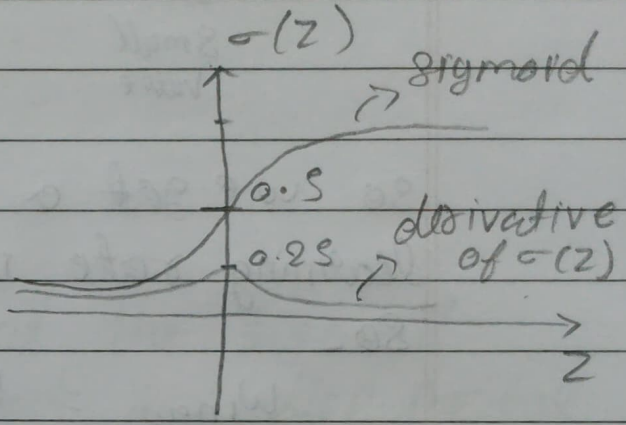
$$o_{31} = \sigma(z)$$

$$\frac{\partial O_{31}}{\partial O_{21}} = \frac{\partial(\sigma(z))}{\partial(z)} \times \frac{\partial(z)}{\partial O_{21}}$$ { chain rule.

$\boxed{0 \leq \sigma(z) \leq 1}$ ⇒ when we applied the ~~sigmoid~~ sigmoid

derivative of $\sigma(z)$ →

$\boxed{0 \leq \sigma(z) \leq 0.25}$



$\partial(\sigma(z)) = 0 \leq \sigma(z) \leq 0.25$ ~~$\times \frac{\partial((w_3 \times O_{21}) + b_3)}{\partial O_{21}}$~~

$$\Rightarrow \frac{\partial O_{31}}{\partial O_{21}} = 0 \leq \sigma(z) \leq 0.25 \times \frac{\partial((w_3 \times O_{21}) + b_3)}{\partial(O_{21})}$$

$$\Rightarrow \boxed{\frac{\partial O_{31}}{\partial O_{21}} = 0 \leq \sigma(z) \leq 0.25 \times w_{3\,old}}$$

So we get the $\frac{\partial O_{31}}{\partial O_{21}}$ in the range of $(0 - 0.25)$ which is small.

So in case of updating the w_1 weight

$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{1old}}$$

$\frac{\partial L}{\partial o_{31}}$ → some small value

$\frac{\partial o_{31}}{\partial o_{21}}$ → $(0-0.25)$

$\frac{\partial o_{21}}{\partial o_{11}}$ → $(0-0.25)$

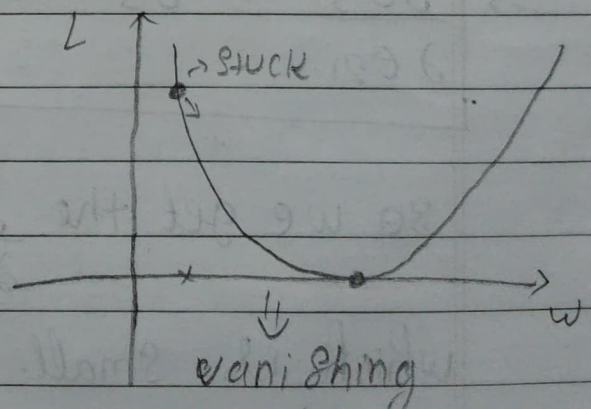
$\frac{\partial o_{11}}{\partial w_{1old}}$ → $(0-0.25)$

so we got a smaller value and our learning rate is also a smaller value so-

$$w_{inew} = w_{iold} - \left(\eta \frac{\partial L}{\partial w_{iold}}\right) \rightarrow \text{very small.}$$

$$\rightarrow \{w_{inew} \approx w_{iold}\}$$

So in this case w_{iold} is like never updated. and stuck to achive the global minima this is called the gradient vanishing problem.

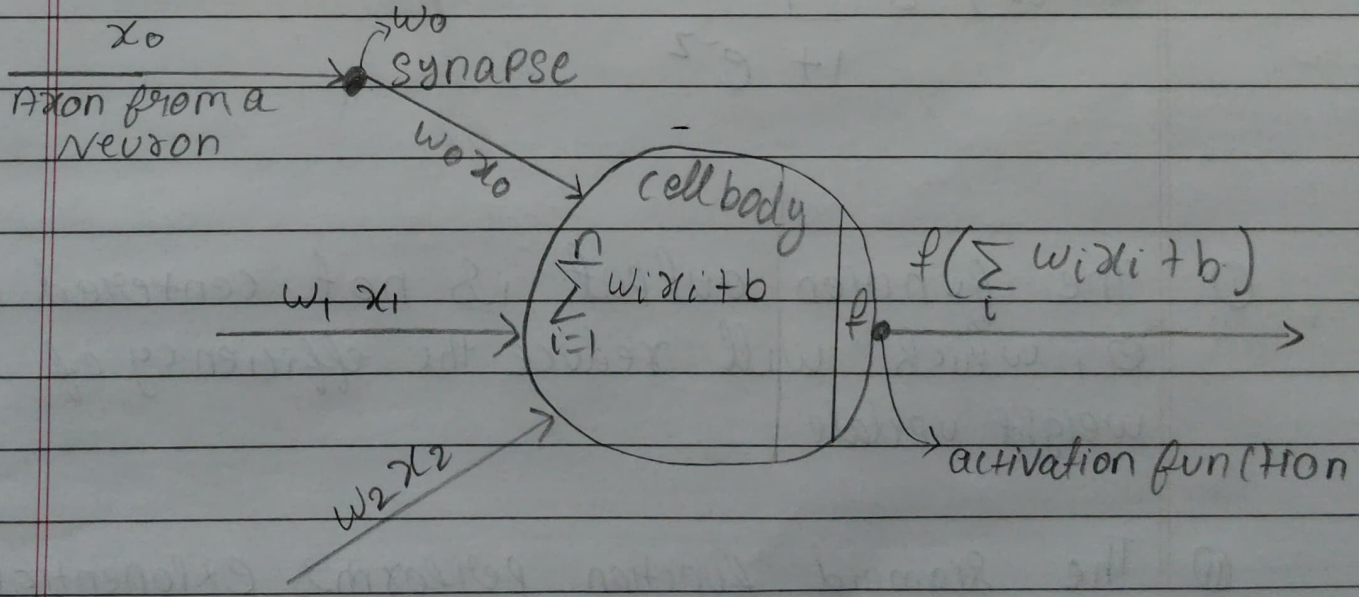


vanishing gradient problem.

To fix the Vanishing gradient Problem the Researchers started to exploring other activation function.

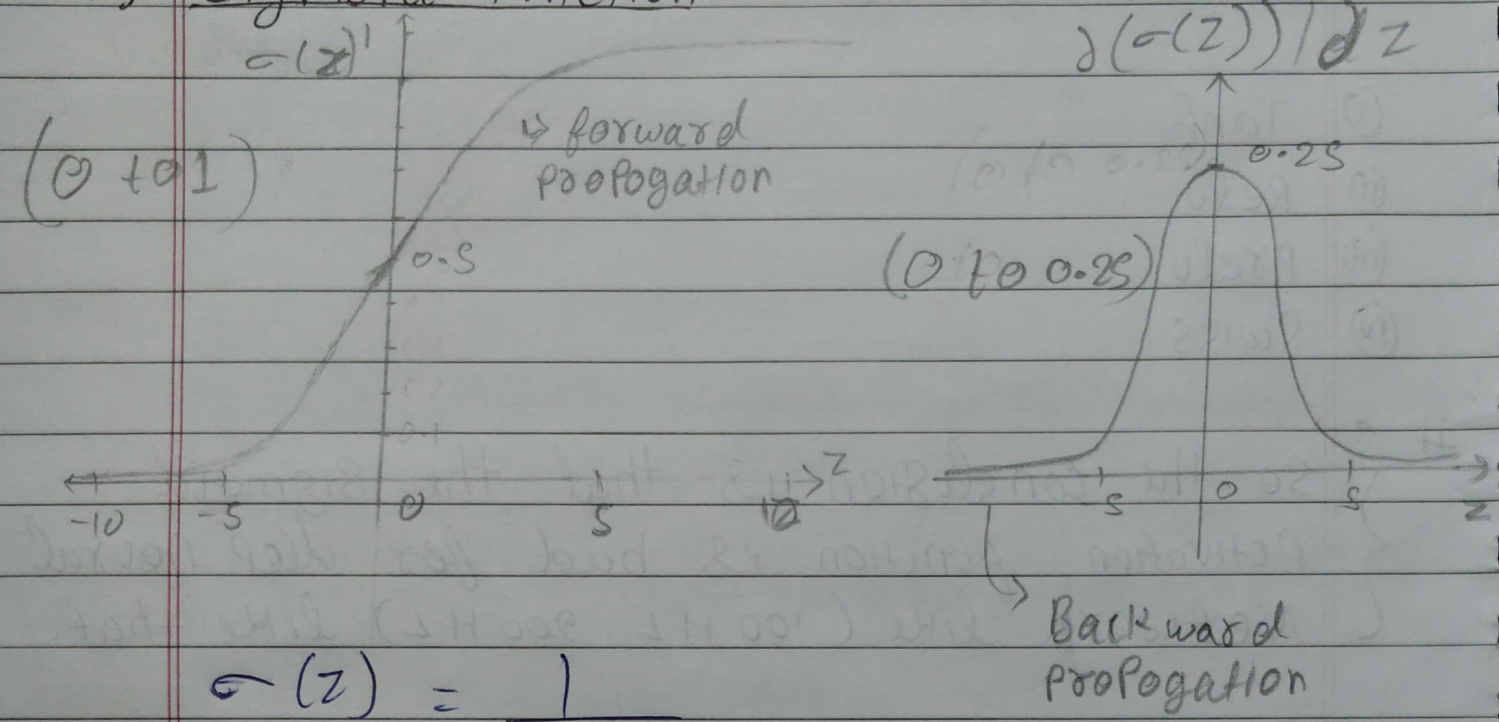
(i) Tanh
(ii) Relu
(iii) Prelu
(iv) Swiss

So the conclusion is that the Sigmoid-Activation function is bad for deep neural Network like (100 H.L, 200 H.L) like that.



Activation Functions.

1) Sigmoid Function:



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

(i) The function output is not centered on 0, which will reduce the efficiency of weight update.

(ii) The Sigmoid function performs exponential operations, which ~~would~~ is slower the computer.

Advantage of Sigmoid function:-

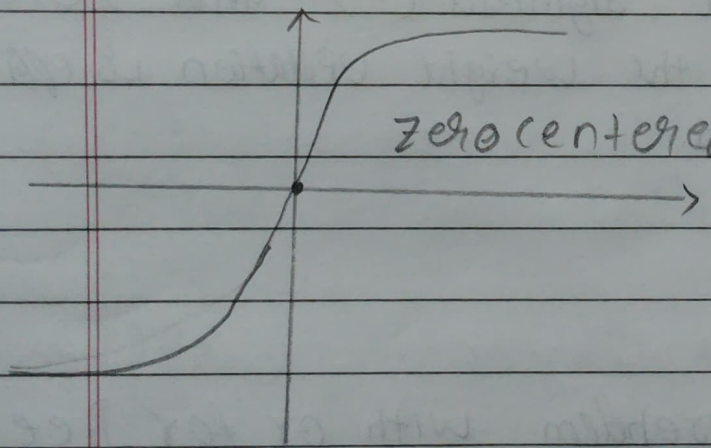
(i) Smooth gradient, preventing "jumps" in output
(ii) output values bound between 0 and 1, normalizing the output of each neuron.
(iii) Clear predictions, i.e. very close to 1 or 0.
(iv) Suitable for binary classification.

Disadvantages:

(i) prone the Gradient vanishing.

(ii) function output is not zero-centerd.

(iii) Power operations are relatively time consuming.



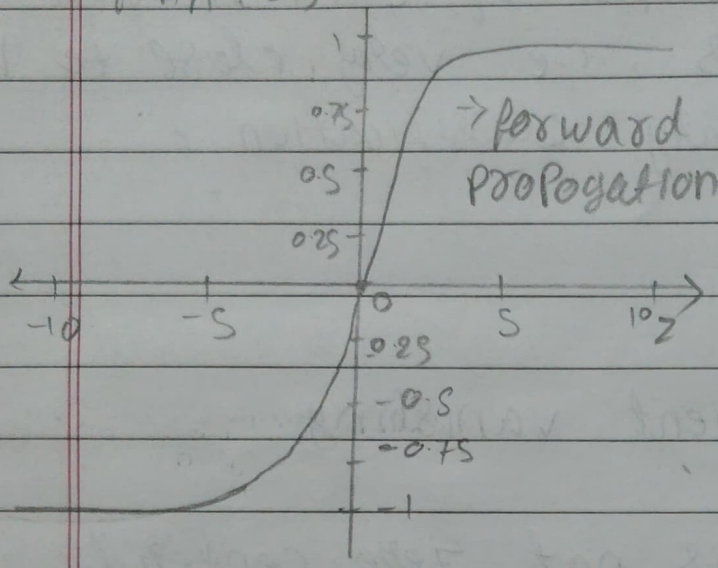
zero centered. => efficeant weight updation.

② tanh function:

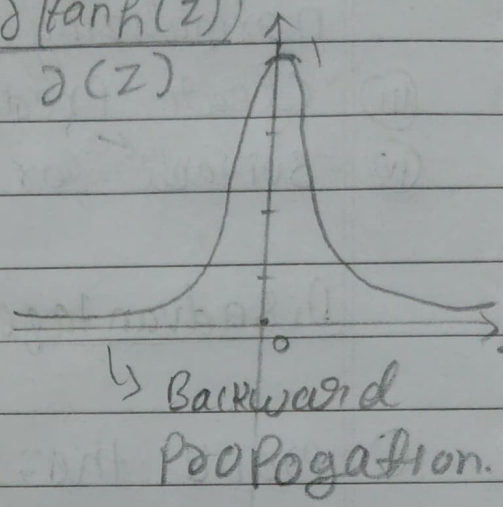
$$\left\{ z = \sum_{i=1}^{n} w_i^T x_i + b \right\}$$

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

tanh (z)



$\frac{\partial(\tanh(z))}{\partial(z)}$



Advantage:

① it is better than sigmoid (σ) and zero-centric that's why the weight updation is efficient

② binary classification.

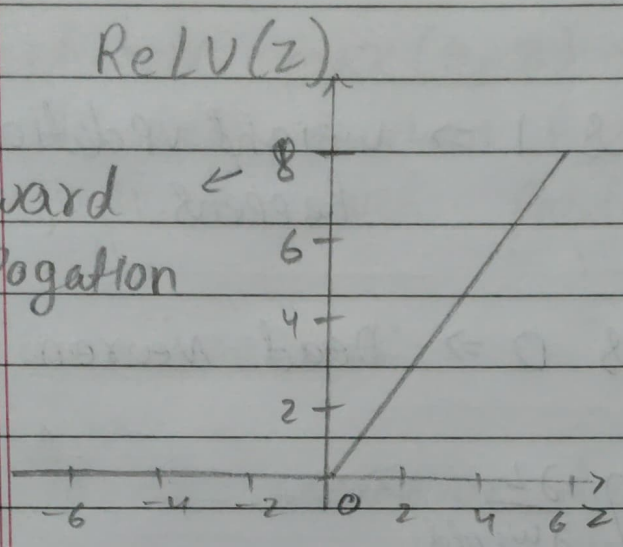
① Dis advantage.

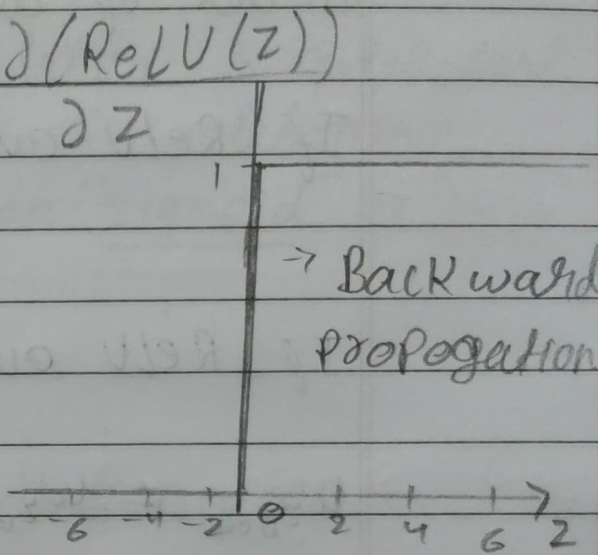
① vanishing gradient problem with or for Deep neural network.

② It take to much time for computation because of exponential power.

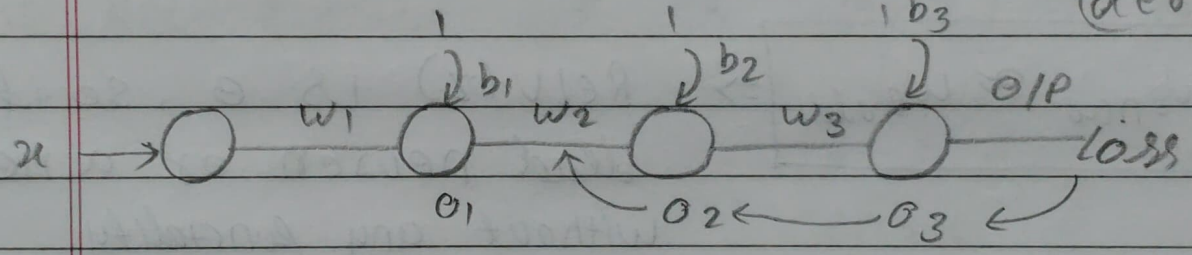
② ReLU Activation function:-

$$ReLU(z) = \max(0, z)$$ → formula.





(derivative)



$$\rightarrow \frac{\partial L}{\partial w_{2\,old}} = \frac{\partial L}{\partial O_3} \times \boxed{\frac{\partial O_3}{\partial O_2}} \times \frac{\partial O_2}{\partial w_2} \quad \left\{ w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}} \right.$$

$$\rightarrow \frac{\partial O_3}{\partial O_2} = \boxed{\frac{\partial (ReLU(z))}{\partial z}} \times \frac{\partial z}{\partial O_2} \quad \{ \because z = O_2 \times w_3 + b_3 \}$$

$$\downarrow \qquad \downarrow$$

$$[0 \text{ or } 1] \times \frac{\partial [(O_2 \times w_3) + b_3]}{\partial O_2}$$

Date
Page

$\Rightarrow$ $[0 \text{ or } 1] \times w_3 \Rightarrow$ (Small value get.)

In case of derivative.

If ReLU output is 1 $\Rightarrow$ weight updation happens.

If ReLU output is 0 $\Rightarrow$ Dead Neuron.

$$w_{2new} = w_{2old} - \eta \boxed{\frac{\partial L}{\partial w_{2\,old}}} \to 0$$

$\boxed{w_{2new} \approx w_{2old}}$ $\Rightarrow$ ReLU(z) is 0, so it is dead neuron or a neuron without any functality.

(i) if $z = +ve$ $\quad \frac{\partial(\text{ReLU}(z))}{\partial z} = 1$

(ii) if $z = -ve$ $\quad \frac{\partial(\text{ReLU}(z))}{\partial z} = 0$

(2) $\Rightarrow$ the dead neuron is just pass the input as it is to the next neuron.

Advantages:

(i) It is Solving the vanishing gradient Problem

(ii) It use max(0, z) → calculation is super-fast
The ReLU function has a linear relationship

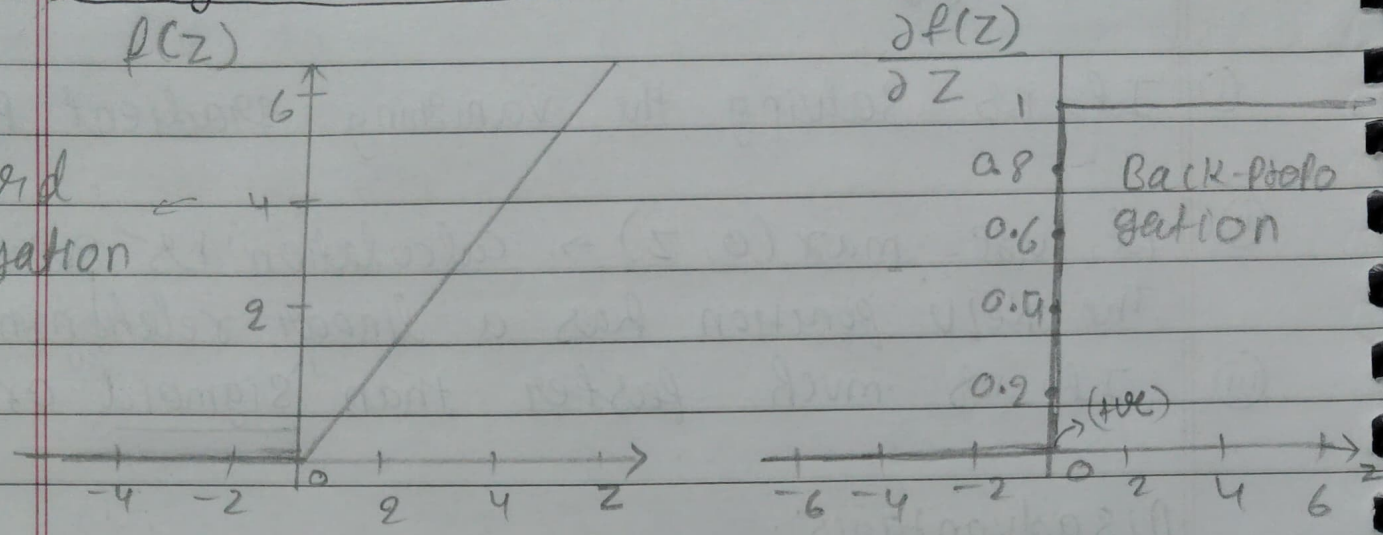
(iii) It is much faster than "sigmoid" or "tanh"

Disadvantages:

(i) It is create or make a dead neuron.

(ii) ReLU function o/p is (0, z) ⇒ 0 or (+ve) number.
⇓
If is not zero centric.

④. Leaky ReLU and Parametric ReLU :-



(Leaky ReLU) → formula = $f(z) = \max(0.01z, z)$

alpha → [0.01, 0.02, 0.03]

(Parametric ReLU) → formula ⇒ $f(z) = \max(\alpha z, z)$

↳ Hyper Parameter

So in this Leaky ReLU activation function have some (+ve) value insted of 0 so it is never give the output 0

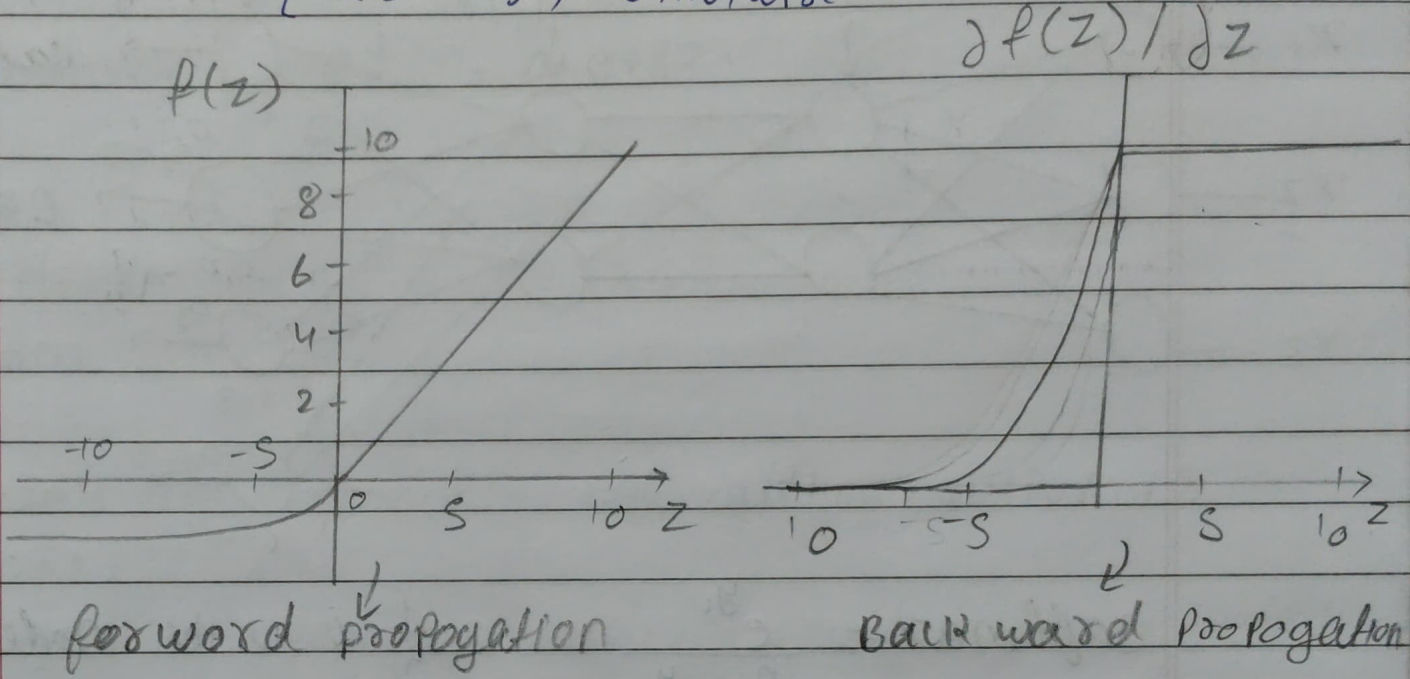
So it remove the probelam of Dead neuron.

ReLU ⇒ Rectified Linear Unit.

it is not a zero-centric.

⑤. ELU (Exponential Linear Units)

$$f(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha(e^z - 1), & \text{otherwise} \end{cases}$$

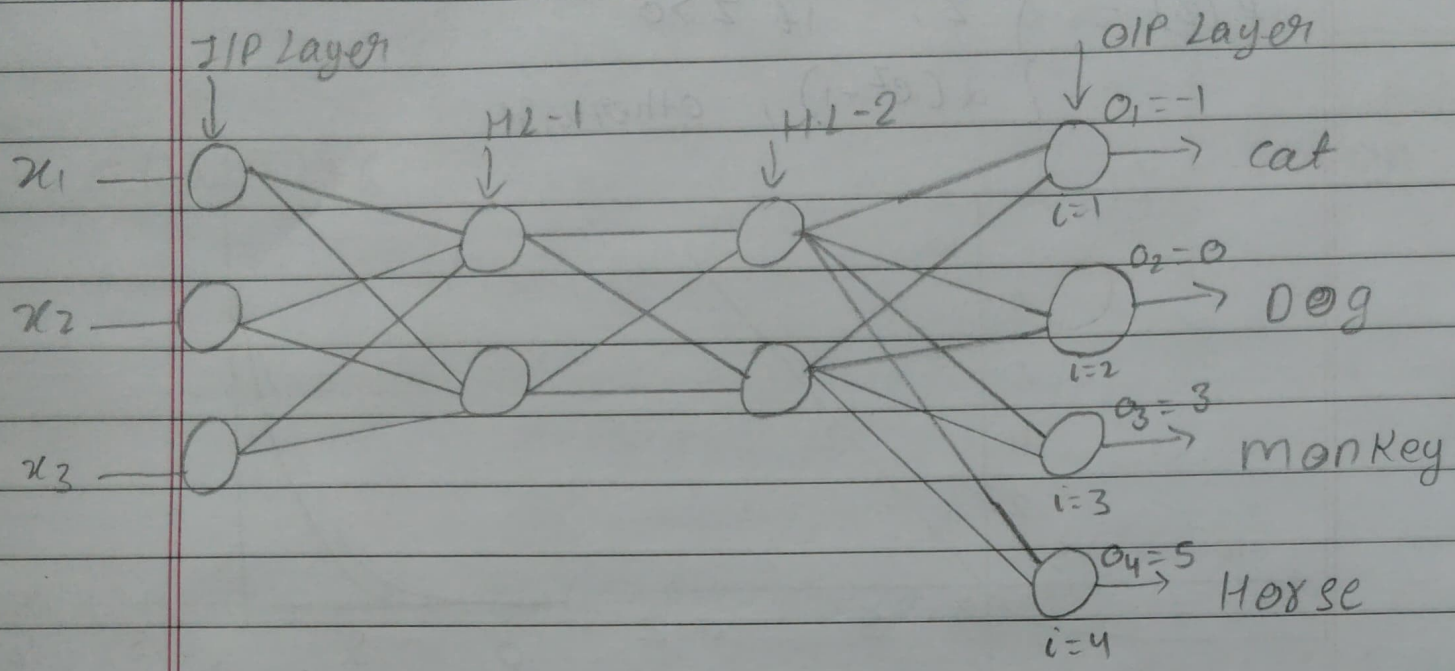


forword propogation ↓ Back ward Propogation ↓

Advantages	Disadvantages.
① No dead ReLU issues	① Slightly more computationally Intensive.
② It is Zero centric	

{ It is solve the ReLU problem. }

⑥ Softmax Activation function ⇒ [multiclass classification Problem].



$$\text{softmax} = \frac{e^{y_i}}{\sum_{K=1}^{n} e^{y_K}}$$

{ i is the node / neuron no.

y_i = output of that neuron.

$\boxed{y_i = \text{output} \times w + b}$

apply the softmax.

$$\text{cat} = \frac{e^{-1}}{e^{-1} + e^{0} + e^{3} + e^{5}} = \frac{e^{-1}}{e^{-1+0+3+5}} = \frac{e^{-1}}{e^{7}} = 0.00033$$

if $i = 1$ $e^{y_i} = e^{-1}$ because y_1 of output is -1 and $y_2 = 0$, $y_3 = 3$, $y_4 = 5$.

$$Dog = \frac{e^0}{e^{-1+0+3+5}} = \frac{e^0}{e^7} = 0.0024$$

$$monkey = \frac{e^3}{e^7} = 0.0183$$

$$Horse = \frac{e^5}{e^7} = 0.1353$$

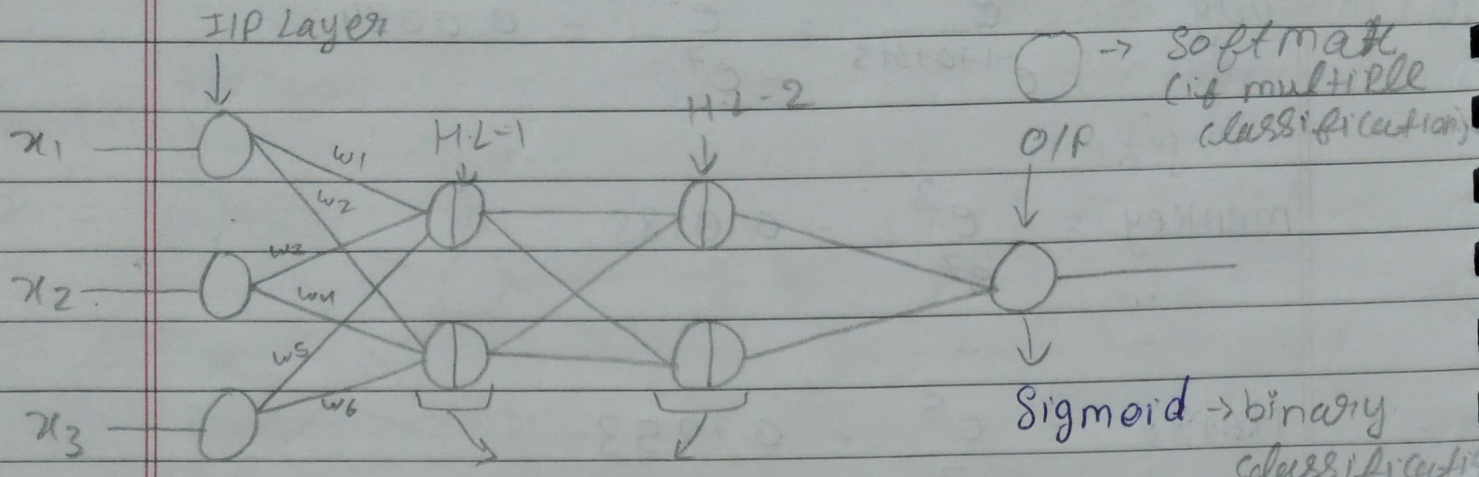
So the probability of getting output is horse is

$$P(Horse) = \frac{0.1353}{0.00033 + 0.0024 + 0.0183 + 0.1353}$$

$$= 0.86 \approx 86\%$$

So there is 86 % chance to the output is horse at the given input, so like wise we can get or findout the others probability.

⑦ Which Activation Function To ~~use~~ / Apply when?



So if your neural network is very Deep then use the ReLU activation function.

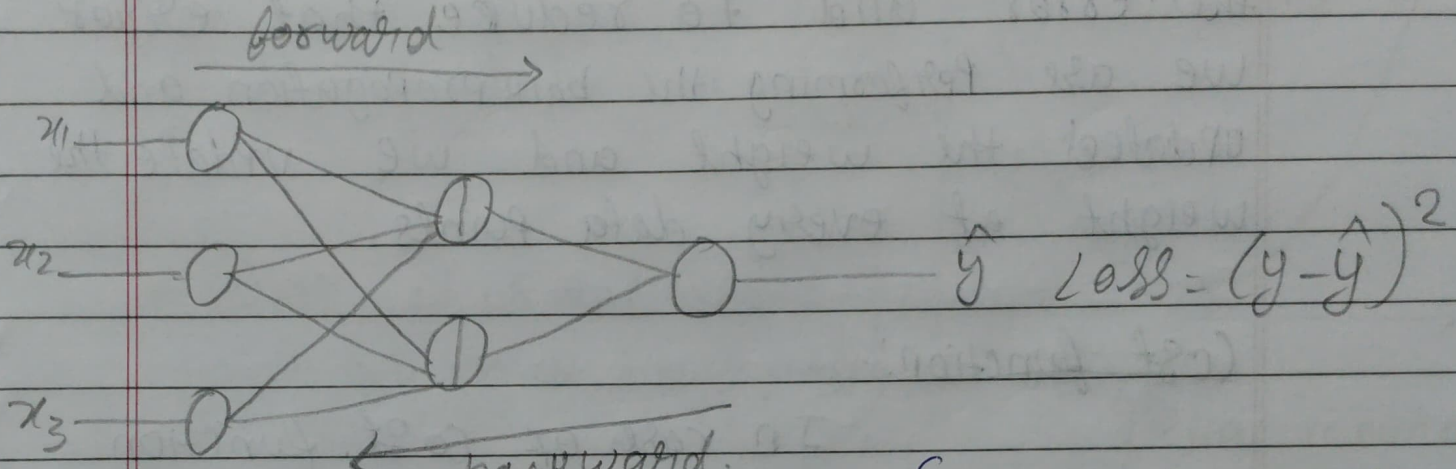
① if we use the Sigmoid activation function or tanh, it leads the vanishing gradient problem So to Randle these type of problem we use the ReLU activation function or it's varients.

② but in the output layer if we require the binary output the inside the O/P layer we are using the Sigmoid activation function.

classmate
Date 27/3/25
Page

=> If we require the multiple classification like dog, cat, horse e.t.c. we are using the Softmax activation function in the O/P layer.

Lost function And Cost function.



x_1	x_2	x_3	O/P
–	–	–	0
–	–	–	1
–	–	–	0
–	–	–	1

loss function

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

Update at every point.

cost function

$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$

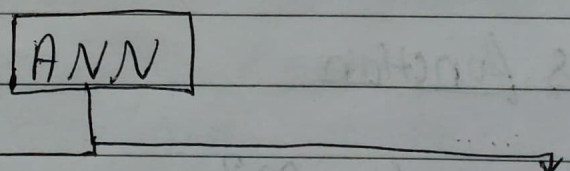
→ take the batch and update once.

Loss function:-

In case of loss function. we give the input and Forward Propogation happens then it is give the $\hat{y}$ the predicted value and we calculate the error and to reduce that error we are performing the back propogation and updated the weight and we update the weight at every data points.

Cost function:-

In case of cost function we give the inputs at once and after performing the operations. and get the $\hat{y}$ value after this it is calculate the errors and update the weight once only. in the back propogation.



(I) mean squarl Error (MSE)
(II) mean Absolute Error (MAE)
(III) Huber loss
(IV) RMSE (Root mean squarl Error)

Different Loss and cost function in the Regression Problem

classmate
Date
Page

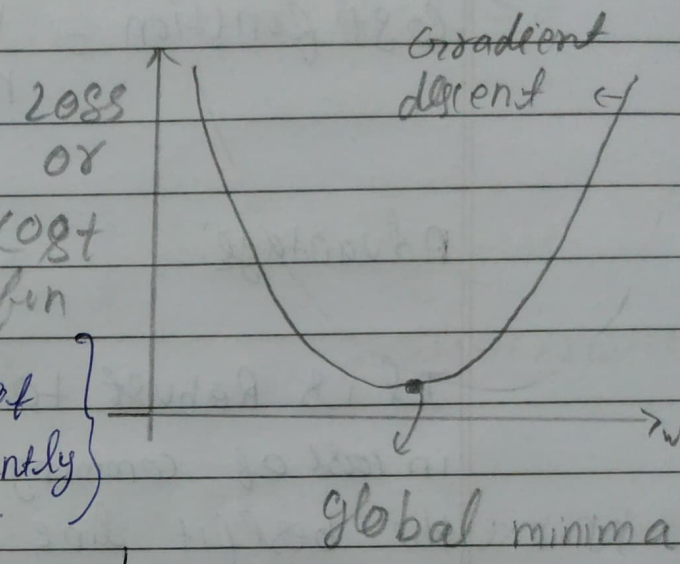
① Mean Squared Error (MSE) ⇒

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

↳ Quadratic equation.

Cost function.

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



outliers: it is that type of values that deviate significantly from other values in a data set.

Advantage.

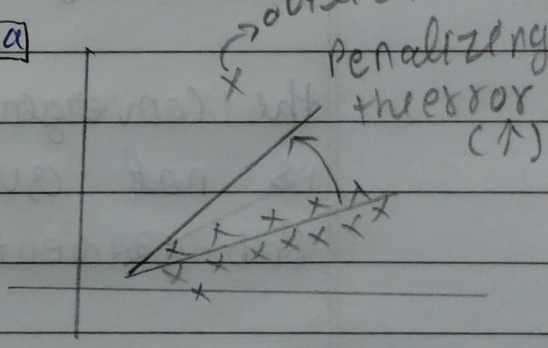
(i) MSE is Differentiable at any Point.

(ii) It has only 1 local minima or global minima.

(iii) It converges faster because of Smooth gradient descent

Disadvantage.

(i) Not robust to outliers.



in case of outlier the best fit line shift much towards the outlier

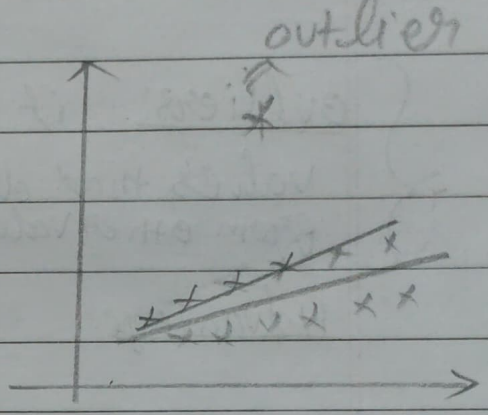
② Mean Absolute Error (MAE)

Loss function = $|y - \hat{y}|$

Cost function = $\frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$

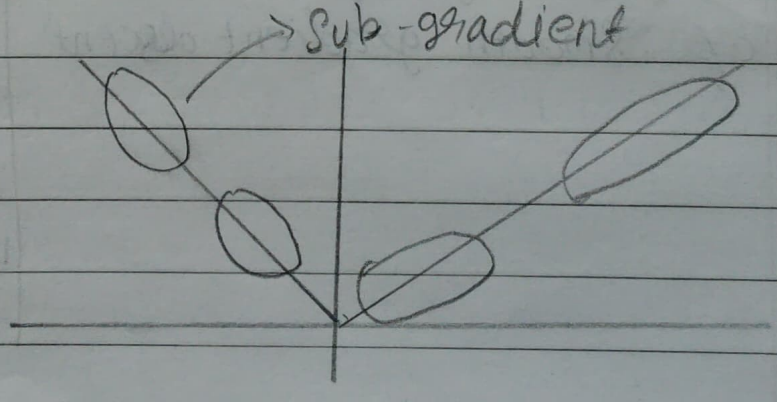
Advantage:

It is Robust to outlier.
in case of comming of outlier
the bestfit line shift very
less toward the outlier.



Disadvantage:

the convergence take time in MAE → because the curve is not quadratic so it is create a Sub-gradient descent and calculate the slope.



③ Huber loss: It is a combination of MSE and MAE with the ~~same~~ outliers and hyperparameter. (MSE) no outliers present hyperparameter

$$\text{Cost function} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2 ; & \text{if } (y-\hat{y}) \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 ; & \text{otherwise.} \end{cases}$$

(MAE) outlier present

④ Root mean squared Error (RMSE)

$$\text{Cost function} := \sqrt{\frac{\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}{N}}$$

N = No. of Data points.

Advantages:	Disadvantages.
Differentiable at any point	Not Robust to outliers.
Interpretability of Different models.	scale dependence.

easy

Outliers: It is a data points that behaves differently from all other data points in the dataset.

Loss or cost function for classification problems.

Classification

CROSS Entropy.

- Binary cross entropy → (Binary class)
- Categorical cross entropy → (multiclass)
- Sparse categorical cross entropy. → (multiclass)

① Binary Cross Entropy

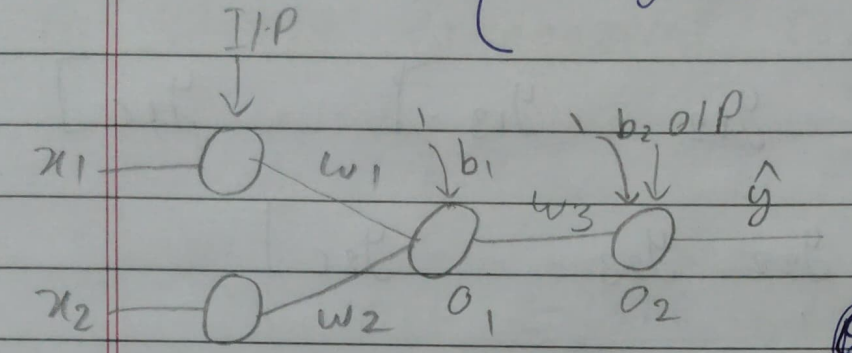
Log loss function :-

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

{ y => actual value
$\hat{y}$ => predicted value. }

classmate
Date 28/3/25
Page

$$loss = \begin{cases} -\log(1-\hat{y}) & ; \text{ if } y=0 \\ -\log(\hat{y}) & ; \text{ if } y=1 \end{cases}$$



binary classification
use sigmoid

$$o_2 = \sigma(w_3 \times o_2 + b_2)$$

$$\hat{y} = \frac{1}{1+e^{-z}} \rightarrow \text{Sigmoid.}$$

② Categorical Cross Entropy (multiclassification)

→ OHE => (one Hot encoding)

f_1	f_2	f_3	o/p
2	3	4	Good
5	6	7	Bad
8	9	10	Neutral

C = No of catories.

on the output feature.

$j=1$	$j=2$	$j=3$
Good	Bad	Neutral
1	0	0
0	1	0
0	0	1

⇒ when output is Good the Good is 1 and others are 0.

$$Loss\ (x_i, y_i) = -\sum_{j=1}^{C} y_{ij} * \ln(\hat{y}_{ij})$$

1st data Point → $y_{ij} = [\ y_{11} \quad y_{12} \quad y_{13} \quad \ldots \quad y_{1C}\ ]$

2nd data Point → $= [\ y_{21} \quad y_{22} \quad y_{23} \quad \ldots \quad y_{2C}\ ]$

Actual value ⇒ $y_{ij} = \begin{cases} 1 ; & \text{if the element is in class} \\ 0 ; & \text{otherwise.} \end{cases}$

Prediction → $\hat{y}_{ij}$ ⇒ apply softmax activation function on the o/p layer.

$$\Rightarrow \boxed{Sof(z) = \frac{e^{z}}{\sum_{j=1}^{K} e^{z_j}}}$$

o/p ⇒ $\hat{y}_{ij}$ is in form of probabilites.

⇒ $[0.2, 0.5, 0.3]$ ⇒ if we have three categories.

↳ Categorical cross Entropy.

The categorical cross entropy is also give the probability of other categories.

(3) Sparse Categorical Cross Entropy:

$\hat{y} = [0.2, 0.3, 0.5]$ → categories (0, 1st, 2nd)

⇓

O/P ⇐ {2nd index have highest probability}

⇓

category

So it is not going to give the others probability.

Suppose you have 5 categories and their

$\hat{y} = [0.2, 0.3, 0.1, 0.2, 0.2]$ → category (0th, 1st, 2nd, 3rd, 4th)

So your final output is 1st index category directly.

Disadvantage ⇒ in case of sparse we loose the info about the probability of other categories.

Which loss function is use when.

	Hidden layer	activation function → O/P layer	Problem	Loss function
(i)	ReLU or (variant)	Sigmoid	Binary classification	Binary cross entropy
(ii)	ReLU ,, ,,	Softmax	multi class classification	Categorical /Spars cross Entropy.
(3)	ReLU	Linear	Regression	MSE, MAE, Rubor loss RMSE

(i) Grad
(ii) Stoc
(iii) mini
(iv) SGD
(v) Adag
(vi) Adar

OPtir
Los

(1) Gra

x_1
x_2
x_3

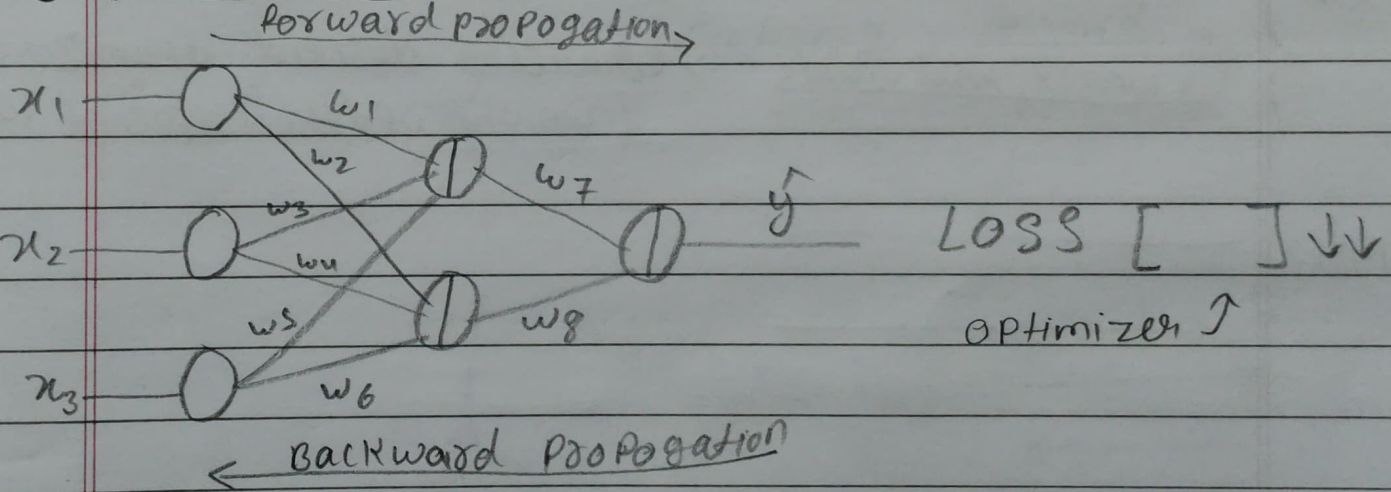
W_{ne}

Optimizers.

(i) Gradient Descent
(ii) Stochastic Gradient Descent (SGD)
(iii) mini batch SGD
(iv) SGD with momentum
(v) Adagard and RMSPROP
(vi) Adam optimizer.

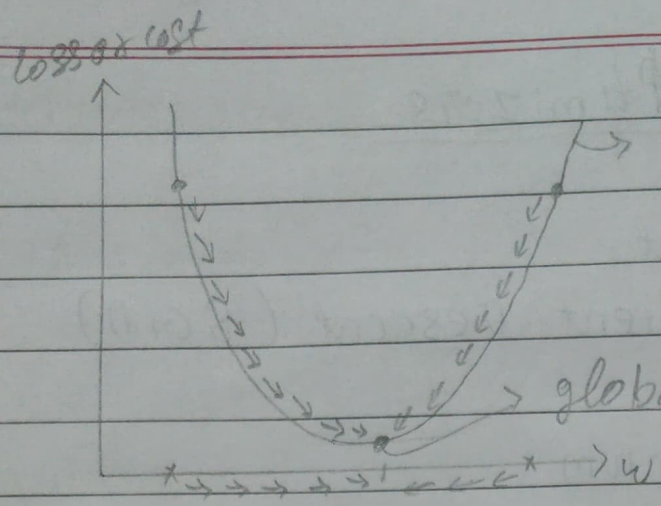
Optimizers are responsible for reducing the Loss function in the backpropogation.

(i) Gradient Descent Optimizer:-



$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$ → weight updation

η → Learning rate



MSE

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

Cost function

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

Epochs, iteration

Data set ⇒ 1000 Data Point

1-Epoch

1000 data point → $\hat{y}$ ⇒ cost function ↓↓

weight will update → gradient descent optimizer → [weight updation applied]

2-Epoch

100 Epoch

Advantages:	Disadvantage:
① convergence will happen.	① It is require huge resource like RAM, GPU ⇓ it is Resource intansive.

② Stochastic Gradient Descent (SGD)

1000 datapoint

Epoch-1

Iteration-1 → 1 datapoint → ŷ → loss ↓↓ → optimizer (SGD) → ~~weight update~~

Iteration-2 → forward → backward

Iteration-1000 → ⟶ ⟵

=> Advantage:

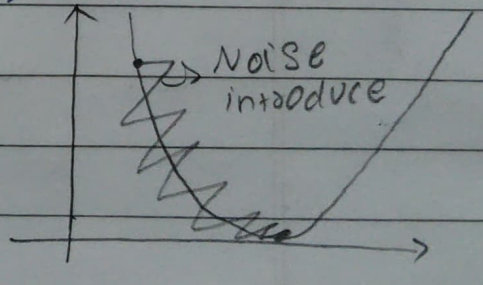
=> it is solve the resource issues.

=> Disadvantages:

=> Huge Time complexity (↑↑)

=> Convergence take more time

=> Noise get introduce.



③ Mini batch SGD:

In mini batch SGD we introduce the Batch-Size.

{Epoch, Iteration, Batch-Size}

Data Points = 100 K

⇒ batch-Size = 1000 ,

No of iteration = $\frac{100000}{1000}$ = 100 iteration

Epoch-1

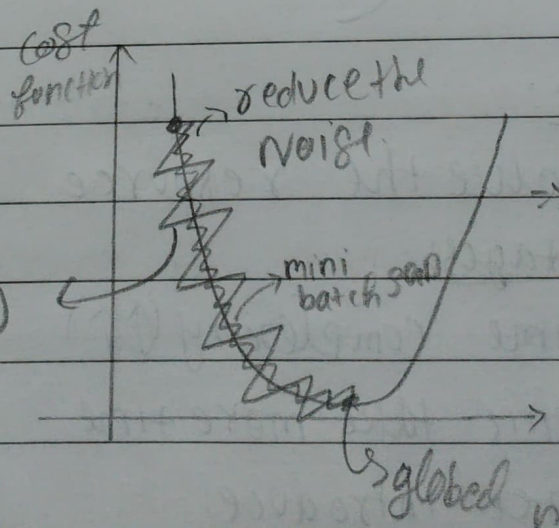
Iteration-1 → 1000 Point → cost = $\frac{1}{1000}\sum_{i=1}^{1000}(y_i - \hat{y}_i)^2$

← (Update weight) Optimizer (mini b SGD)

iteration-2 → ←

⋮

iteration 100 → ←

① 

Advantage:

(i) Convergence speed will increase

(ii) Noise will be less compaired with SGD

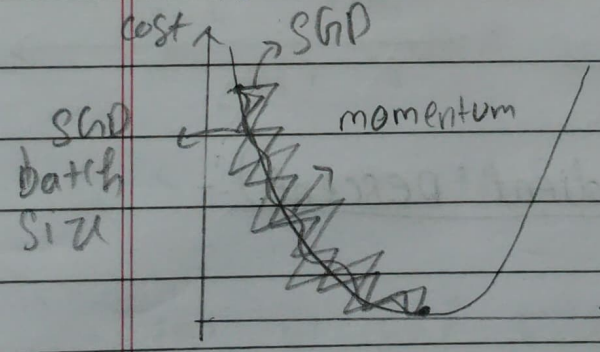
(iii) Efficient resource usage (RAM, GPU)

Dis advantage:

(i) Noise still exists.

(4) SGD with momentum =>

In this optimizer we ~~age~~ are going to smooth the noise of SGD.



$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

update w.r.t time.

$$W_t = W_{t-1} - \eta \frac{\partial L}{\partial W_{t-1}}$$

Exponetial weight Average (Smoothening)

ARIMA, SARIMA

Time series

time	t_1	t_2	t_3	t_4 - - -	t_n
value	a_1	a_2	a_3	a_4 - - -	a_n

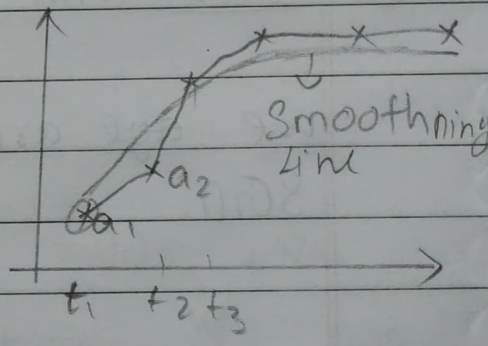
$\Rightarrow v_{t_1} = a$

$\Rightarrow v_{t_2} = \beta * v_{t_1} + (1-\beta) * a_2$

$\Rightarrow v_{t_3} = \beta * v_{t_2} + (1-\beta) * a_3$

{ β = control the smoothening function }

$\{0 \leq \beta \leq 1\}$



$\Rightarrow$ Advantage:-

(i) Reduce the noise.

(ii) Quick convergence.

(5) Adagrad: (Adaptive Gradient Descent):-

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$\Rightarrow$ We know that the learning rate is fixed we are going to make it dynamic.

Dynamic learning rate

$$w_t = w_{t-1} - \eta' \frac{\partial L}{\partial w_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$ → very small value

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_t}\right)^2$$

→	→	→
$t = 1$	$t = 2$	$t = 3$
$\eta = 0.01$	$\eta = 0.005$	$\eta = 0.003$

Disadvantage:-

① η' → Possibly to become a very small value ≈ 0

⇒ so in this case $w_t \approx w_{t-1}$

the weight updation is not happen!

⑥ Adadelta and RMSPROP

$$\eta' = \frac{\eta}{\sqrt{Sdw_t + \varepsilon_0}}$$

we use the Exponential weight Average to restrict the η' to reduce to much.

$$Sdw_t = \beta * Sdw_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

[Dynamic learning rate + Smoothening EWA]

$$w_t = w_{t-1} - \eta'\left(\frac{\partial L}{\partial w_{t-1}}\right)$$

w_t ↓ New w_{t-1} ↓ old

⑦ Adam optimizer → {Best optimizer}

↳ {SGD with momentum + RMSPROP [Dynamic LR + Smoothening]}

$$w_t = w_{t-1} - \eta' V_{dw} \quad \rightarrow \text{weight updation}$$

$$b_t = b_{t-1} - \eta V_{db} \quad \rightarrow \text{bias updation.}$$

$$\eta' = \frac{\eta}{\sqrt{Sdw_t + \varepsilon_0}}$$

$$Sdw_t = \beta * Sdw_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

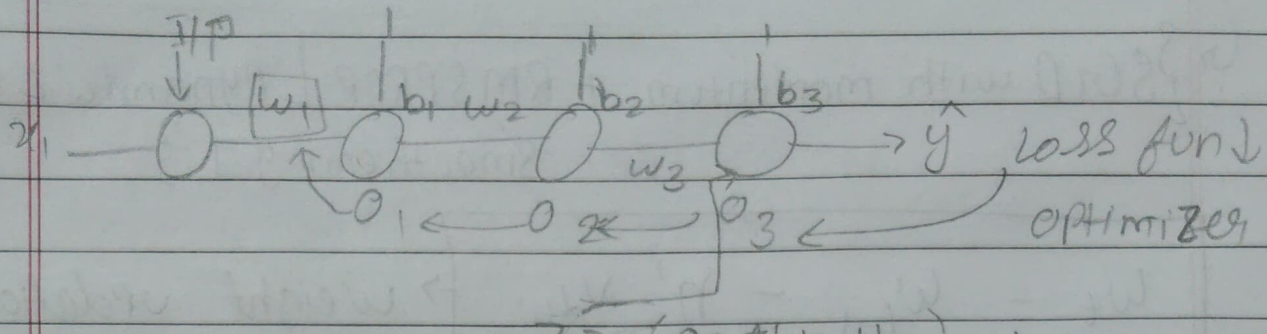
initialy $\Rightarrow Sdw_t = 0$

$$V_{dw_t} = \beta * V_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)$$

$$V_{db_t} = \beta * V_{db_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial b_{t-1}}\right)$$

↳ {momentum Smoothening}

Exploding Gradient problem.



$\rightarrow z = (o_2 * w_3 + b_3) \Rightarrow$

updating the w_1

$$w_{i,new} = w_{i,old} + \eta\left(\frac{\partial L}{\partial_{old} w_{i,old}}\right)$$

$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial o_3} \times \boxed{\frac{\partial o_3}{\partial o_2}} \times \frac{\partial o_2}{\partial o_1} \times \frac{\partial o_1}{\partial w_{1old}}$$

↳ this part

$$\frac{\partial o_3}{\partial o_2} = \frac{\partial(\sigma(z))}{\partial z} \times \frac{\partial z}{\partial o_2}$$

$$\Downarrow$$

$$\Rightarrow [o - 0.25] \times \frac{\partial (o_2 \times w_3 + b_3)}{\partial o_2}$$

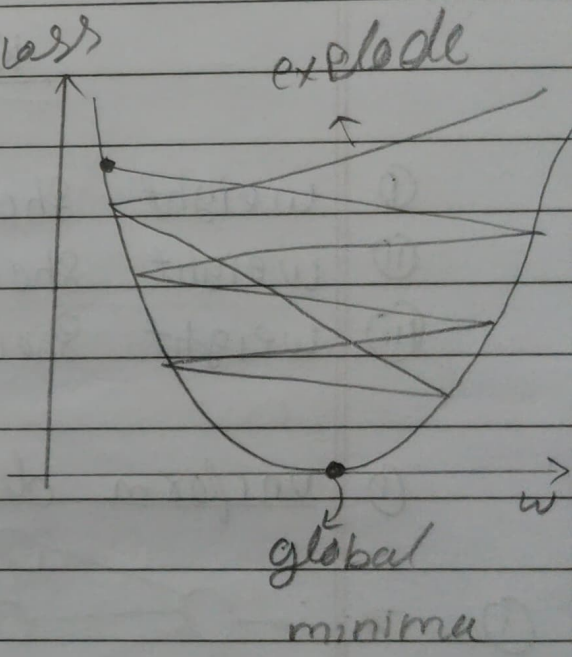
$$\left\{ \frac{\partial o_3}{\partial o_2} \Rightarrow [o - o.2S] \times w_3 \right\}$$

if weight initialization is very high value.

then the $\left(\frac{\partial L}{\partial w_{i\,old}}\right)$ is very high it may be (+ve) or (- ve).

it this case

$$\left\{ \begin{array}{c} w_{i\,new} >>>> w_{i\,old} \\ \text{or} \\ w_{i\,new} <<<< w_{i\,old} \end{array} \right\} \Rightarrow$$



Weight Initializing Techniques.

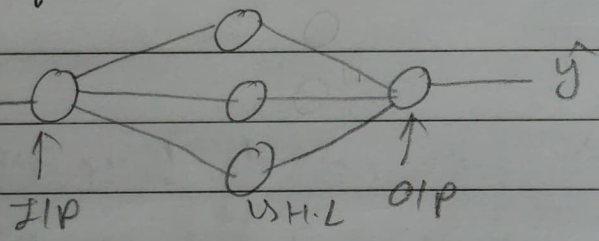
(I) Uniform distribution
(II) Xavier / Glorot Initialization
(III) Kaiming He Initialization.

Key points

(I) weight Should be small.
(II) weight Should not be same.
(III) weight Should have good variance.

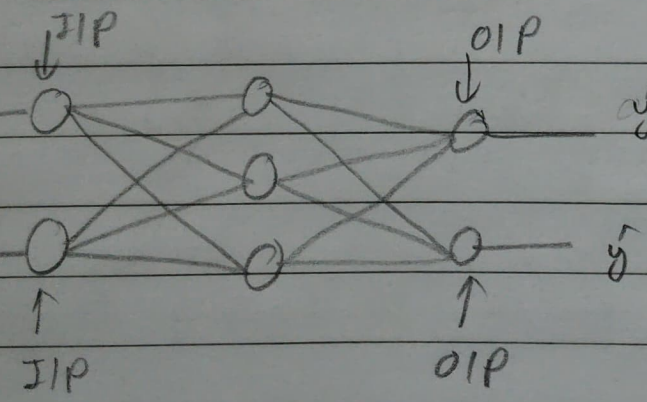
(I) Uniform distribution:

(I) x_1 → $\hat{y}$



input = 1
output = 1

(II) x_1, x_2 → $\hat{y}$



input = 2
output = 2

weights

$W_{ij} \approx$ uniform distribution $\left[\frac{-1}{\sqrt{input}}, \frac{1}{\sqrt{input}}\right]$

in case of 2nd $\Rightarrow \left[\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$

(2) Xavier / Glorot Initialization:-

two types =>

(i) Xavier normal initialization.
(ii) Xavier uniform initialization.

(i) Xavier normal

$W_{ij} \approx N(0, \sigma)$ (mean, standard deviation)

$$\sigma = \sqrt{\frac{2}{(input + output)}}$$

(ii) Xavier uniform:

$W_{ij} \approx$ uniform distribution $\left[\frac{-\sqrt{6}}{\sqrt{(input + out-put)}}, \frac{\sqrt{6}}{\sqrt{(input + output)}}\right]$

③ Kaiming He Initialization.

① He Normal

weights ← $W_{ij} \approx N(0, \sigma)$

$$\sigma = \sqrt{\frac{2}{\text{input}}}$$

② He uniform.

W_{ij} uniform $\left[-\sqrt{\frac{6}{I/P}}, \sqrt{\frac{6}{I/P}}\right]$

Dropout layer.

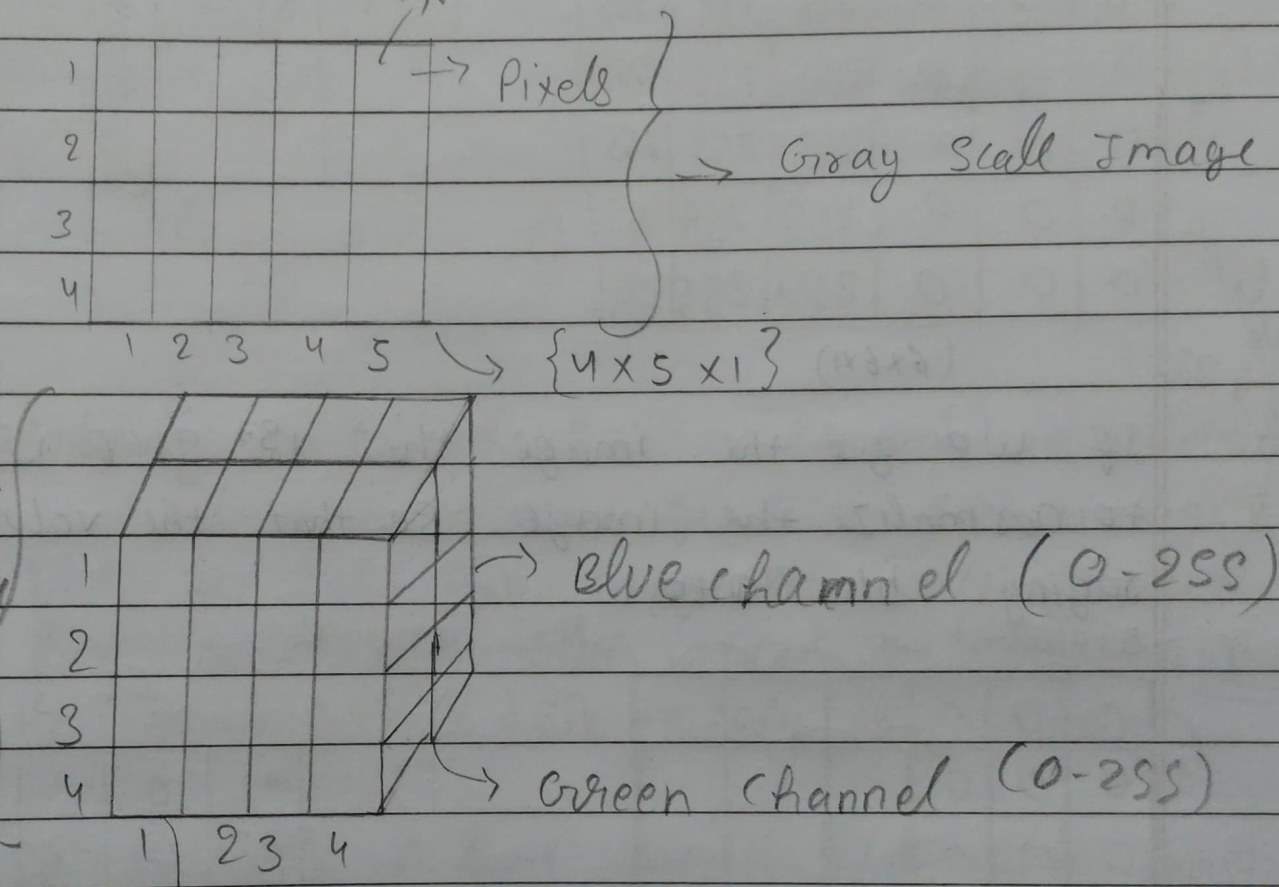
overfitting in neural network when they perform well on the data they have seen during training but fail to generalize to new data.

=> Dropout is a technique that randomly disable the neurons during the training iteration that prevent the dependent of learning pattern and then it perform well on new / unseen data.

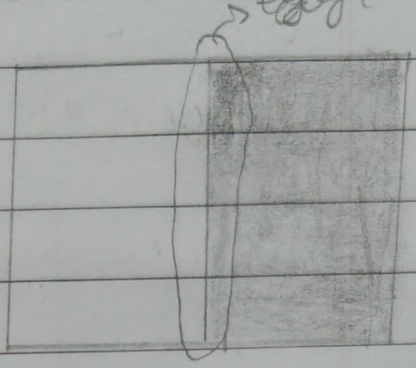
classmate
Date 29/3/25
Page

CNN ⇒ Convolutional Neural Network.

Image:



Convolution operation In CNN:



0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255

(6x6x1)

Normalize ÷255

If we get the image the 1st step is to normalize the image so that the value is raging b/w 0 to 1.

slide Stride=1 → (3+3) move

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

(6x6x1)

n=6

f=3

+1	0	-1
+2	0	-2
+1	0	-1

3+3 filters.

O/P = 4

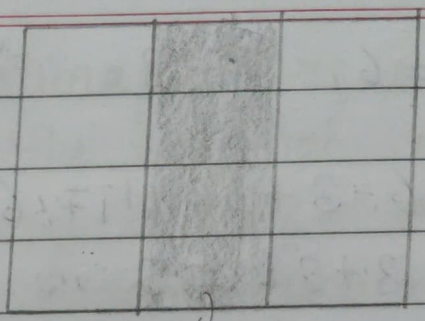
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

4x4

⇒ denormalize

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255

=>



edge detected -> vertical.

{ formula = O/P = n - f + 1 }

Padding in CNN => So in convolution operation we give a 6x6 image size but we get 4x4 image size O/P. So we lost some information about the original image. to prevent this we apply the padding.

So Basically Padding means here we construct or add the lines around the 6x6 matrix and it will become the 8x8 matrix and after apply the filter of 3x3 matrix we get the 6x6 in the out-put.

So formula for finding the Padding. is

{ O/P size require = original size - filter size + 2P + 1

$n = 6, \; f = 3 \Rightarrow \{n - f + 2P + 1 = o/p\}$

$6 - 3 + 2P + 1 = 6$

$3 + 2P = 5$

$2P = 2$

$P = 2/2$

$P = 1$

So we use only 1 Padding on the original given image.

Padding layer

	0	0	0	1	1	1	
	0	0	0	1	1	1	
	0	0	0	1	1	1	
	0	0	0	1	1	1	
	0	0	0	1	1	1	
	0	0	0	1	1	1	

$6 \times 6 \rightarrow 8 \times 8$

×

+1	0	-1
+2	0	-2
+1	0	-1

filter

↓

	0	-4	-4	0	
	0	-4	-4	0	
	0	-4	-4	0	
	0	-4	-4	0	

$4 \times 4 \rightarrow 6 \times 6$

→ This is called the convolutional layer.

Q. If you have 7×7 image size and you apply 3×3 filter on the original image and if you have require 7×7 output. Then find How much padding you require.

7×7	×	3×3	=	7×7
n=7		f=3		O/P=7

Padding require ⇒

⇒ $n - f + 2P + 1 = O/P$

⇒ $7 - 3 + 2P + 1 = 7$

⇒ $4 + 2P + 1 = 7$

⇒ $2P = 7 - 5$

⇒ $P = \frac{2}{2} = 1$

⇒ So padding is two types to initialize.

(i) Zero Padding → fill all the value = 0

(ii) Neighbouring Padding ⇒ fill the value of as the value of Neighbours contain.

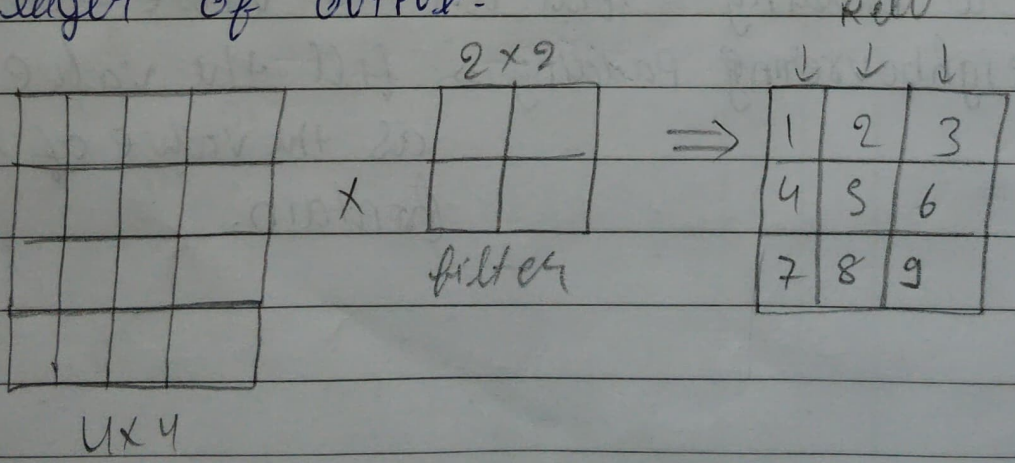
#=> In CNN the filters are going to be updated so in the output matrix we use the ReLU activation function for the back propogation.

The need of ReLU or it's variant:-

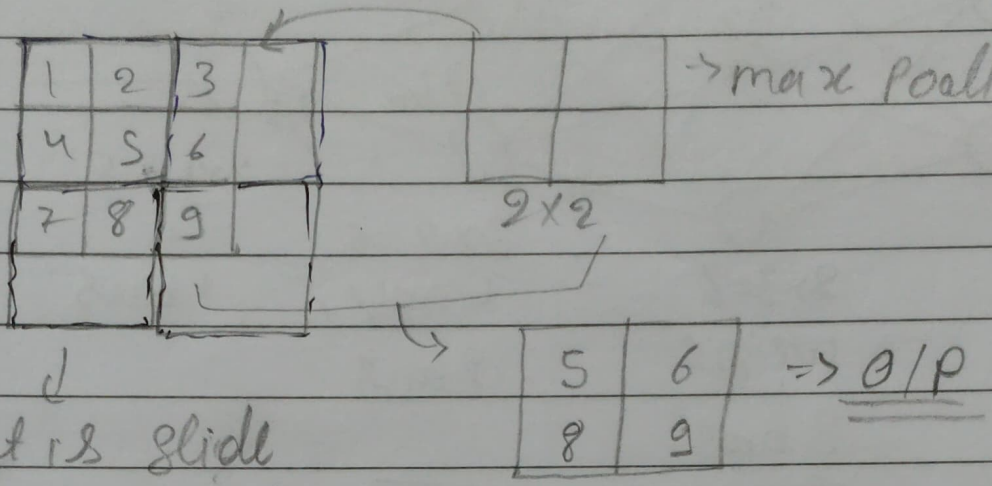
We use the ReLU for finding the derivative and using that derivative with applying the learning rate, we are going to update the values of filters in case of CNN for backward propogation.

Max Pooling, Min Pooling and Mean Pooling.

So basically we use the max pooling to extracat the imp features of the given image. and it is apply on the convolutional layer of output.



Apply the max pooling on O/P.

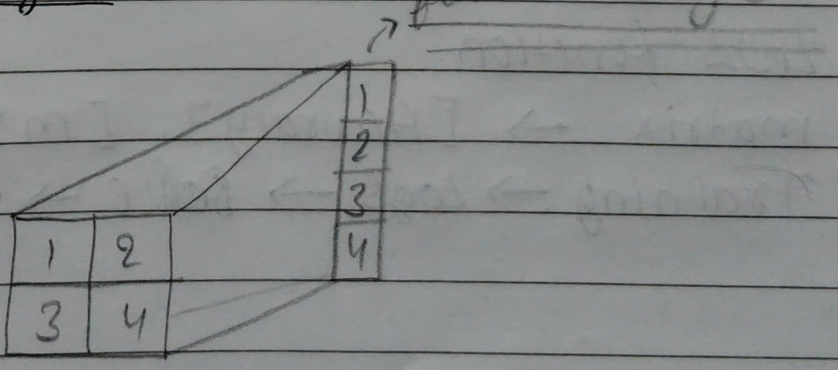


=> In max pooling we extract the max value

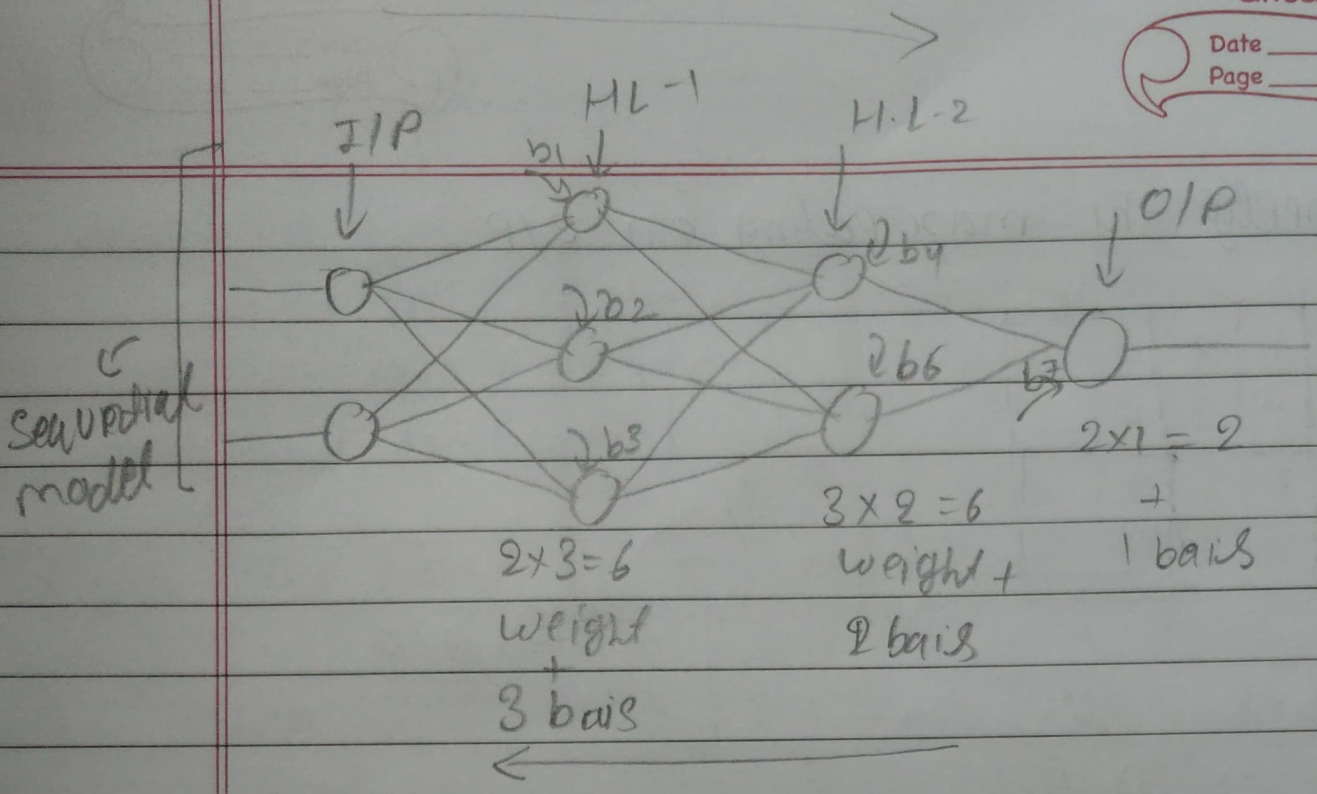
=> In min pooling we extract the min value

=> In mean pooling we extract the mean value.

Flatten layer:



TRaining of ANN



total trainable Parameter = (6+3) + (6+2) + (2+1) = 20

Steps

(I) created a Sequential Network

(II) Dense = 64 => [0, 0, ., 3] => 64 Neurons

(III) APPly the activation function on each node.

(IV) Optimizer

(V) Loss function.

(VI) matrix → [Accuracy], [MSE, MAE].

(VII) Training → Logs → ~~Sell~~ e → Tensorboard ↓ visualization.