Deep learning

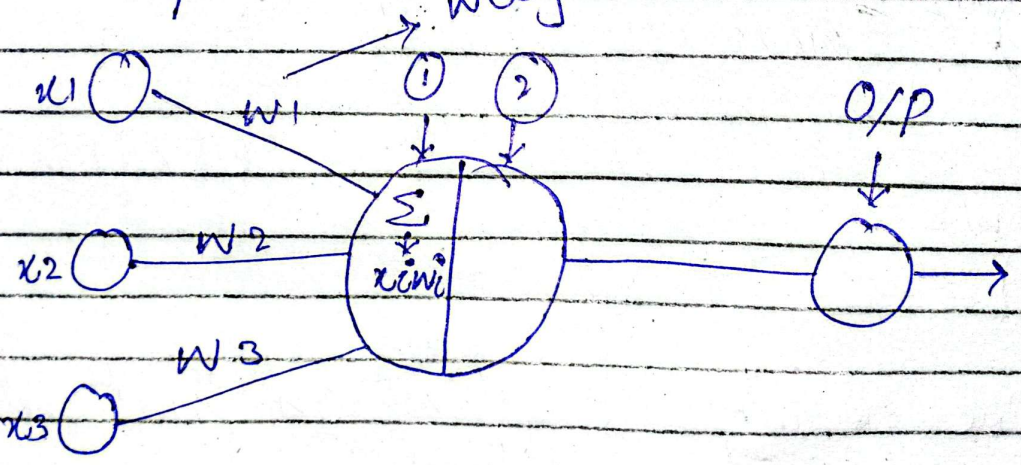
ML Vs DL OR without guidence

ML = Learning with guidance.
DL = Learning without guidance
↓ (using un_SP algo) (learn with guidence
thrives or work on big data. ↓
with SP algo)

Why DL?

↳ Data Explosion.
→ GPU's (Hardware Power)
today GPU's and TPU's can
train huge data in hours
not in decade.
→ Automatic Feature learning.
→ Better performance
→ Works on every type of
data.

Perceptrons: weights

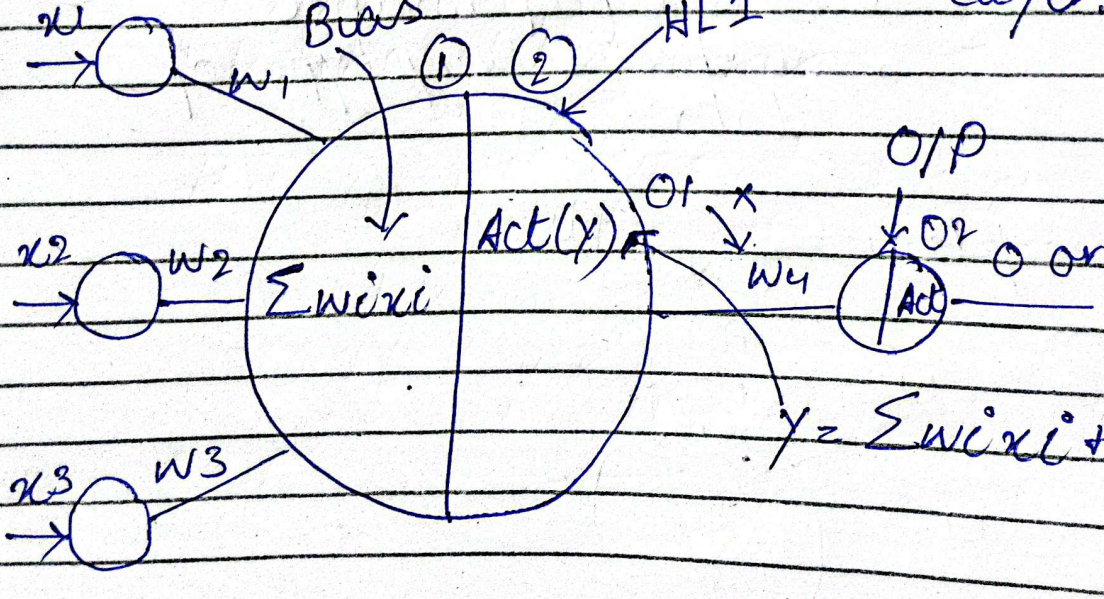


weights will help your neurons to activate at what level.

which weight should have what value so that a neuron is activated till that level or not.

In Simple:
weights says that how much neuron should activated or deactivated.

→ we add Bias to overcome the situation where weight is initialized to 0, and in every situation, and bias is required in every layer.



One example of activation function is sigmoid activation func, for Binary classification.

Sigmoid $= \frac{1}{1+e^{-y}} \Rightarrow \frac{1}{1+e^{-(\Sigma w_i x_i + b)}}$

this was for Binary classification and Binary output.

For Regression we use linear Activation function.

The entire process we have done is called forward propogation

For instance if $\hat{y}$ (Predicted value for 7, 3, 7 is 0 but should come 1) then:

$0 \rightarrow \hat{y}$ $\quad y \Rightarrow$ truth value $= 1$

$y - \hat{y} = 1 - 0 = 1 \rightarrow (1)$

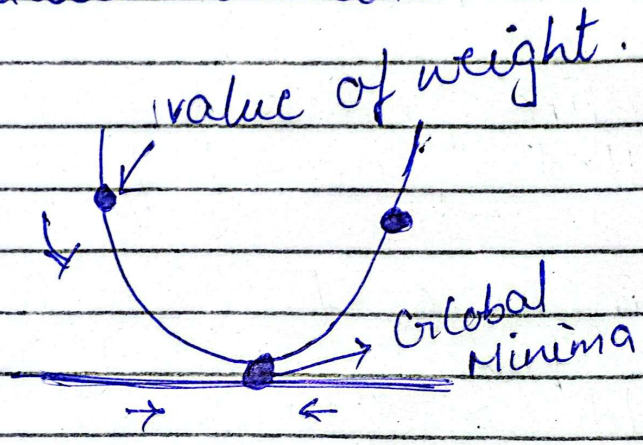
↑ loss function

and our aim is always to minimize the difference, what we'll do if our difference is very huge? $\Rightarrow$ then we'll do back propogation.

In back propogation we update the weights.

To do back propogation we use optimizers, they will ensure that each value of our weight is updated inorder to get satisfactory output.

→ Gradient Descent:



will make sure that the co-efficient is updated to or till to the global minima.

this G.D is type of optimizer.

→ Conclusion:

1. I/P Layer
2. weights will get added.
3. Bias — $y = mx + C$
4. Activision func.
5. output.

Forward Propogation

⑥ Loss function => $\{\hat{y} - y\}$

we need to minimize it to do that we use:

⑦ Optimizers

↳ updates the weights.

↳ By using the concept of Gradient Descent.

Back propagation.

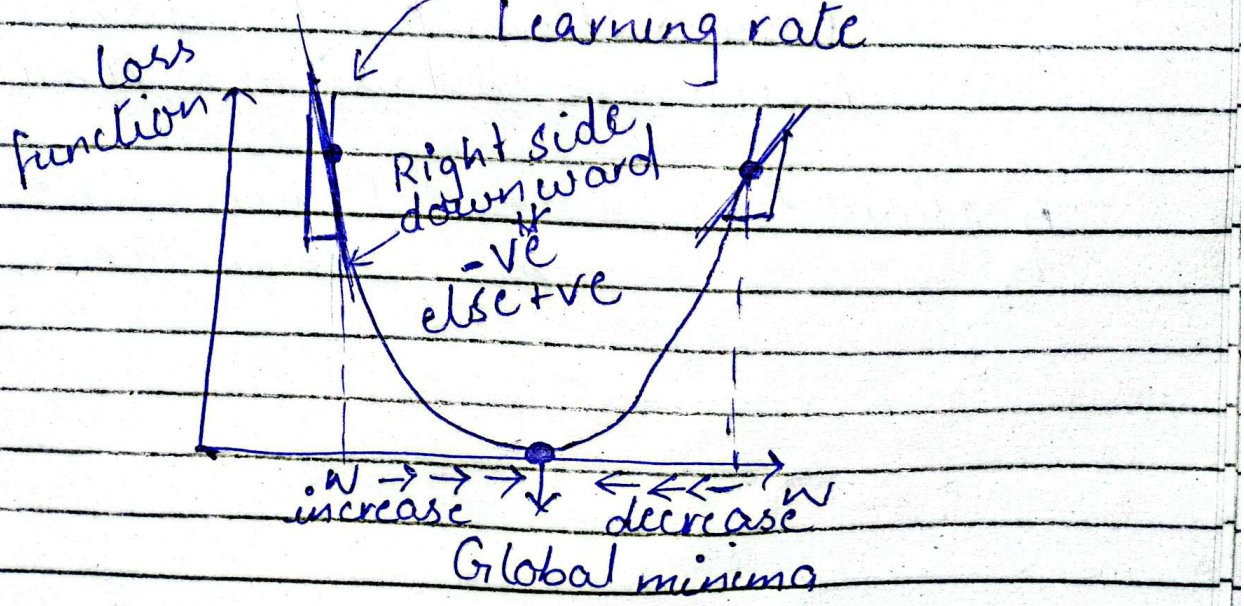
→ Back propagation:

Weight updation formula:

$$W_{new} = W_{old} - \eta \left(\frac{dL}{dW_{old}}\right)$$

Calculating Slope

Learning rate



for (+ve):

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

$W_{new} = W_{old} + \eta$

Now my $W_{new} > W_{old}$

for (-ve):

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

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

$W_{new} < W_{old}$

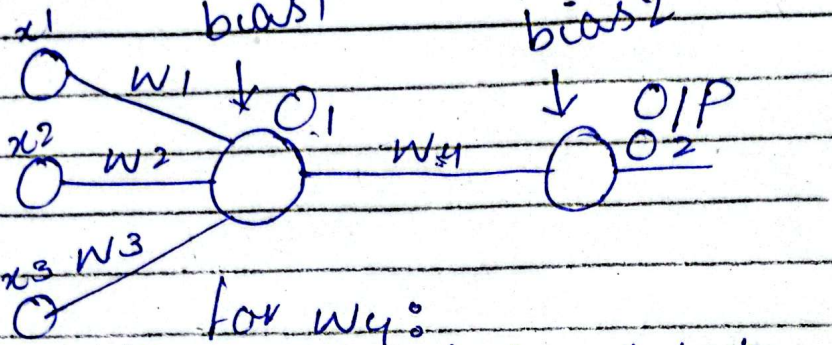
η :=> ≈ Small number
↳ it will slowly converge to global minima (Best).

η => Larger number
↳ it might jump here and there and might not be able to reach global minima (Worst)

η should be = 0.001 or else like this. (Best practice)

② Chain Rule of Derivative:

$$W_{new} = W_{old} - \eta \frac{dL}{dW_{old}} \rightarrow (1)$$



for W_4:

if we want to update value of W_4 we'll take $\frac{dL}{dW_{old}}$ from eq (1)

$$\frac{dL}{dW_{4old}} = \frac{dL}{dO_2} * \frac{dO_2}{dW_{4old}}$$

eq (1) will same formula for bias:

$$bias_{2new} = bias_{2old} - \eta \frac{dL}{dbias_{2old}}$$

for W_2:

$$\frac{dL}{dW_{1old}} = \frac{dL}{dO_2} * \frac{dO_2}{dO_1} * \frac{dO_1}{dW_{1old}}$$

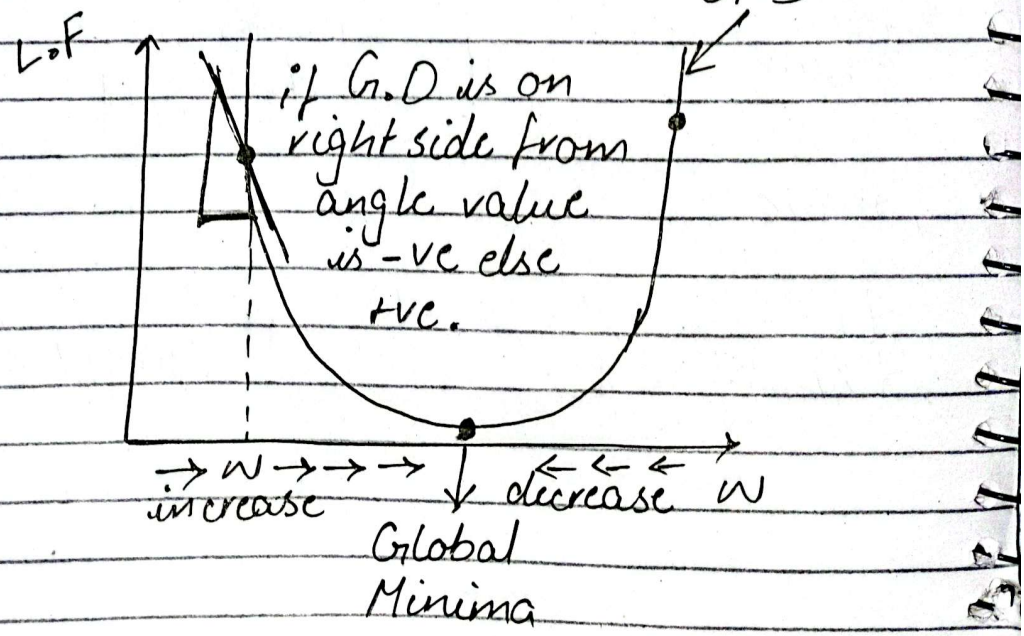
For updating w_2 :

$$w_{2new} = w_{2old} - \eta \frac{dL}{dw_{2old}} \rightarrow (1)$$

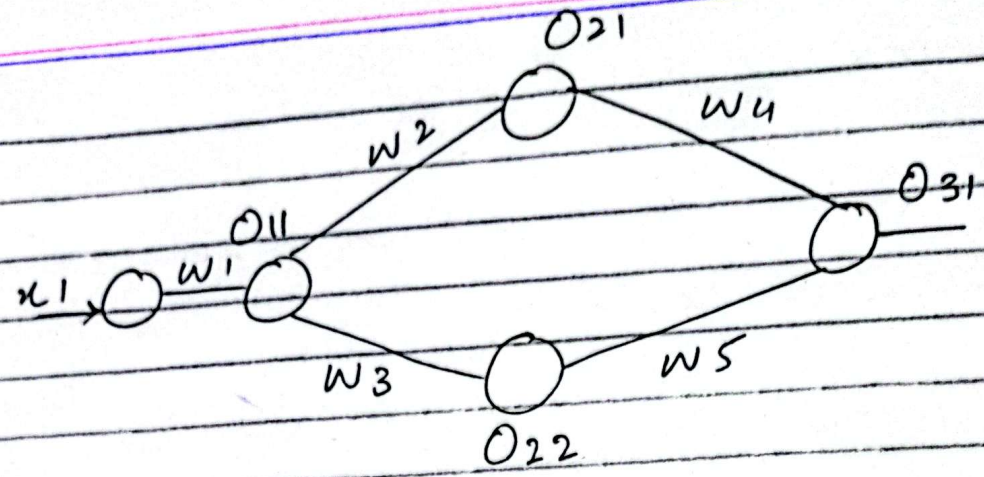
from (1) take $\frac{dL}{dw_{2old}}$.

$$\frac{dL}{dw_{2old}} = \frac{dL}{O_2} * \frac{dO_2}{dO_1} * \frac{dO_1}{dw_{2old}}$$

remember $\frac{dL}{dw_{old}}$ in every weight updation formula will give a slope or a tangent line in graph and from there we'll check wether the value is +ve or -ve if value is +ve then weight will decrease towards global minima else increases.



For 2 routes:



Now for w_1:

$$w_{1,new} = w_{1,old} - \eta \frac{dL}{dw_{1,old}} \rightarrow (1)$$

from (1)

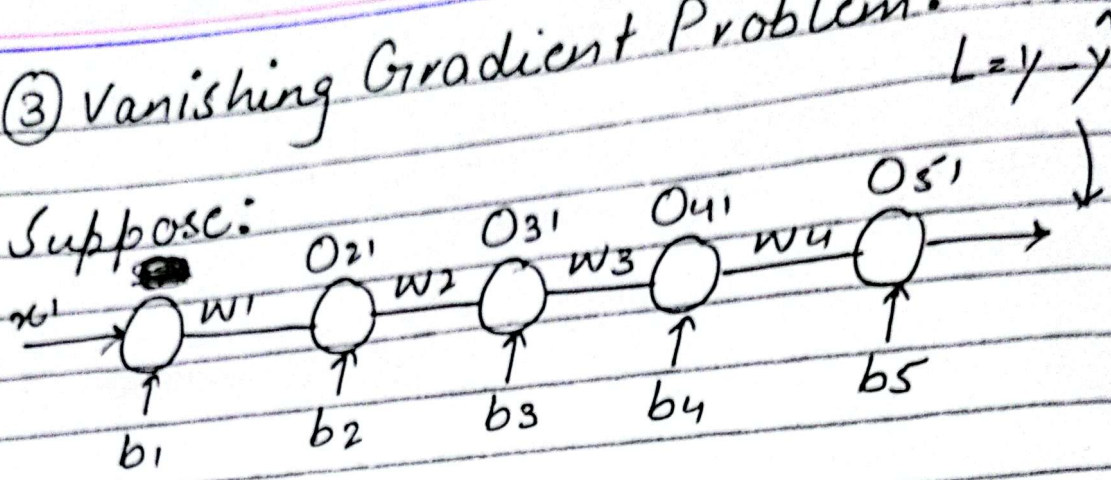
$$\frac{dL}{dw_{1,old}} = \frac{dL}{dO_{31}} * \frac{dO_{31}}{dO_{21}} * \frac{dO_{21}}{dO_{11}} * \frac{dO_{11}}{dw_{1,old}}$$

$$+$$

$$\frac{dL}{dO_{31}} * \frac{dO_{31}}{dO_{22}} * \frac{dO_{22}}{dO_{11}} * \frac{dO_{11}}{dw_{1,old}}$$

③ Vanishing Gradient Problem:

Suppose:



$$w_{1,new} = w_{1,old} - \eta \frac{dL}{w_{1,Old}} \rightarrow (1)$$

from (1) take:

$$\frac{dL}{w_{1,old}} = \frac{dL}{dO_{51}} * \frac{dO_{51}}{dO_{41}} * \frac{dO_{41}}{dO_{31}} * \frac{dO_{31}}{dO_{21}} * \frac{dO_{21}}{dw_{1,old}} \rightarrow (2)$$

Now the question is => Is activation function is implementing in every output?

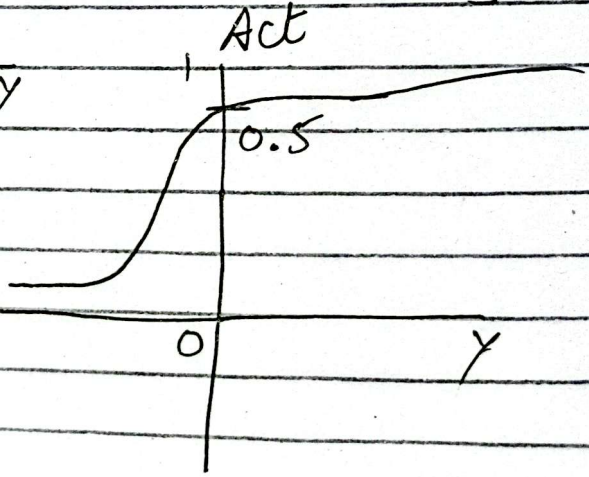
- The Answer is yes.

How? ↓

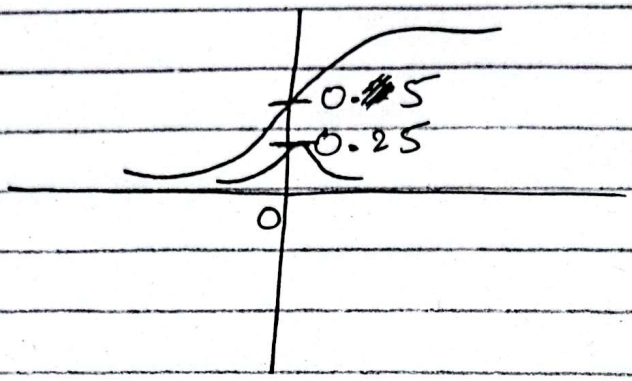
Let take $O_{51} \Rightarrow Act[(O_{41} * W_4) + b]$

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

Act func Graph:



But the derivative of an activision function will always gives value b.w 0 to 0.25.



if values of each derivation in eq ② is like 0.24, 0.21, 0.23, 0.19 then it's fine but what if value decreases consistly like:

$\frac{dL}{w_{old}} = 0.24 * 0.19 * 0.11 * 0.5 * 0.001$

Then:

The value of $w_{i\,new} \simeq$ small value

if it is:

$W_{i\,new} = W_{i\,old} - \eta$ (small value)

η also a small value

$W_{i\,new}$ = Hardly changed value.

This is called gradient descent problem.

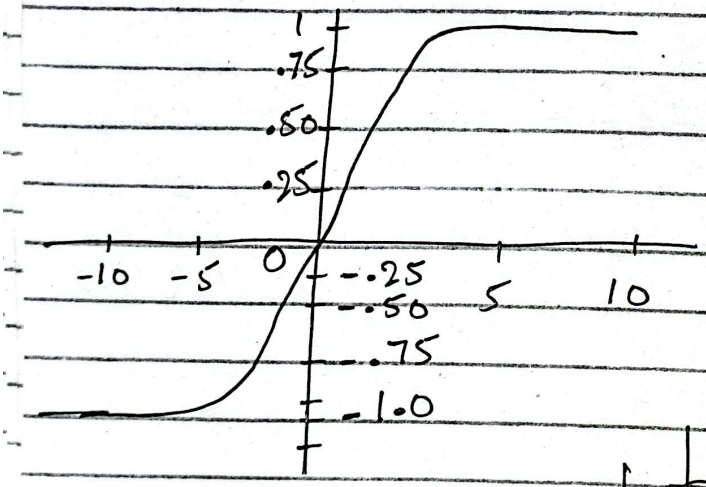
The solution to this is to try other Activation functions like:

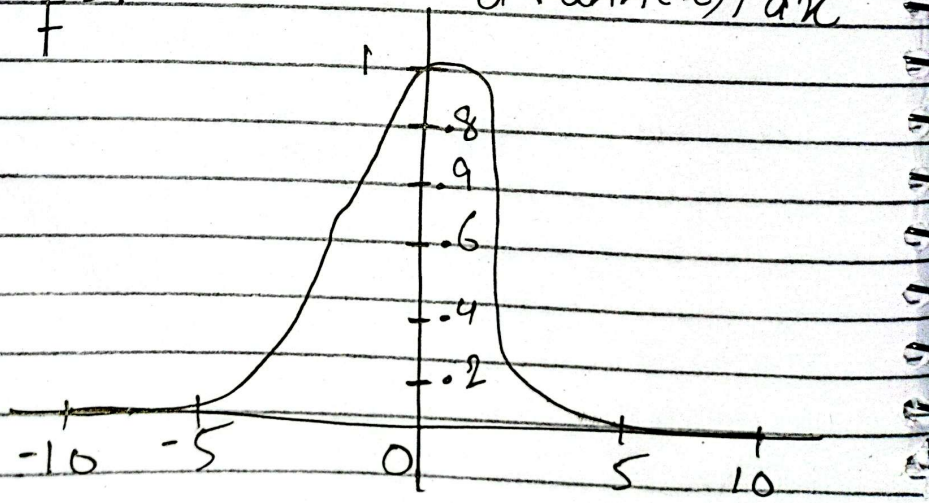
1. Sigmoid.
2. Tanh.
3. Relu.
4. LeakyRelu.
5. PreRelu.

→ Types of Activation Functions:

① Tanh:

$$\text{Tanh} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

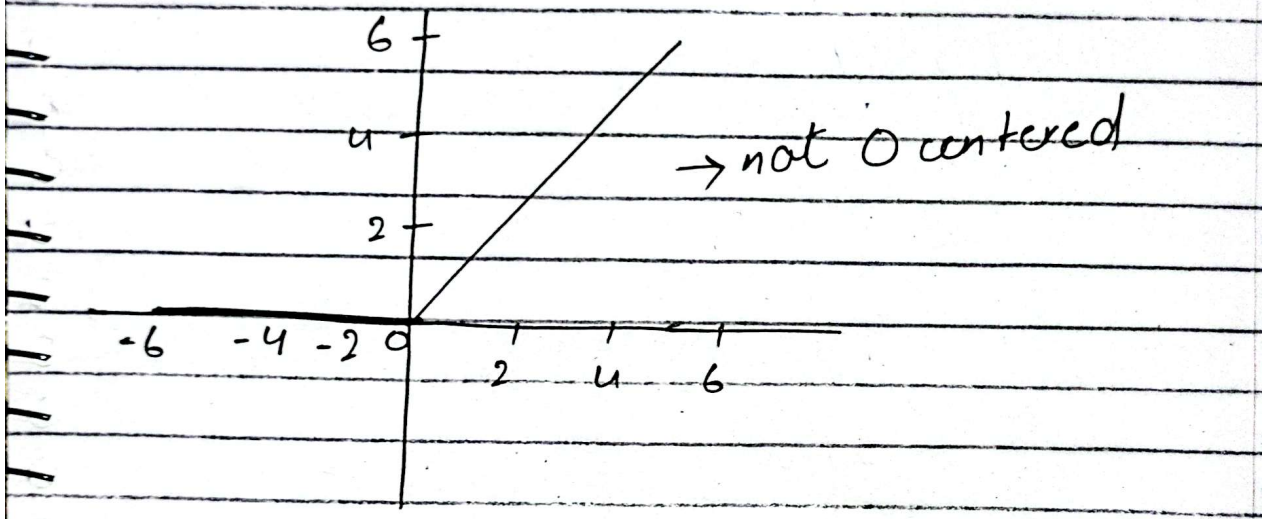




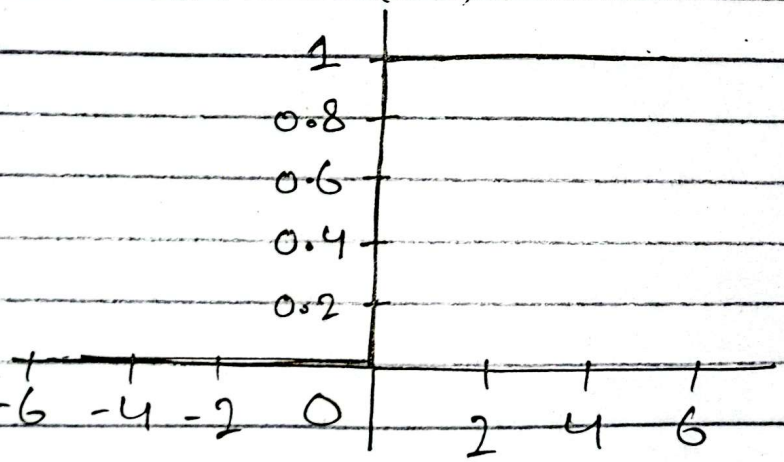
It's activation graph is 0 centered and it's derivative of activation func graph is b.w 0 and 1. Due to which value of weight is efficiently updates, which can help with vanishing gradient problem, but still might not helpful in deep neural Network.

② ReLU (Rectified linear unit):

Relu(x): $ReLU = \max(0, x)$



derative (Relu (x)):



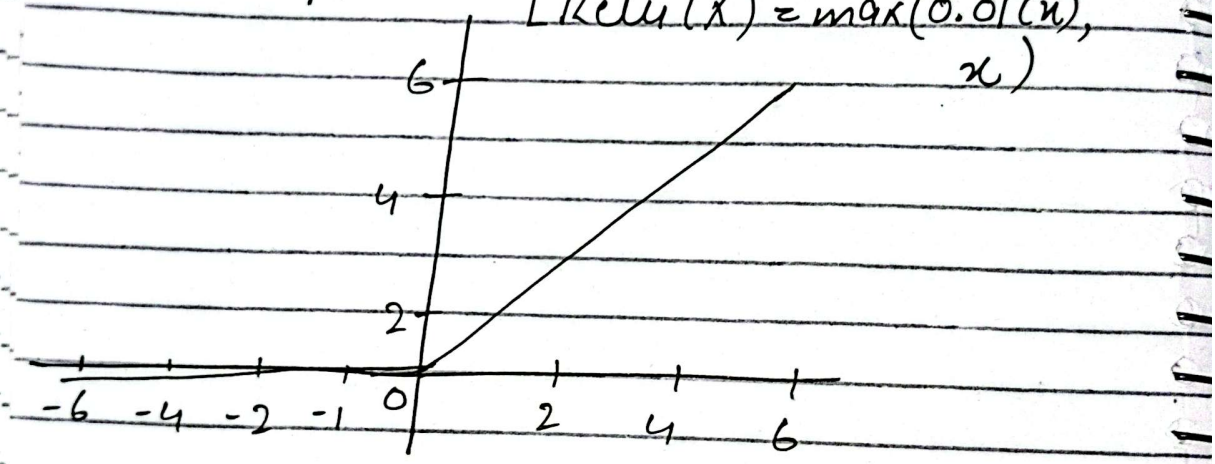
Relu is faster than tanh and sigmoid function as they both need to calculate exponent which is slower.

Disadvantage:

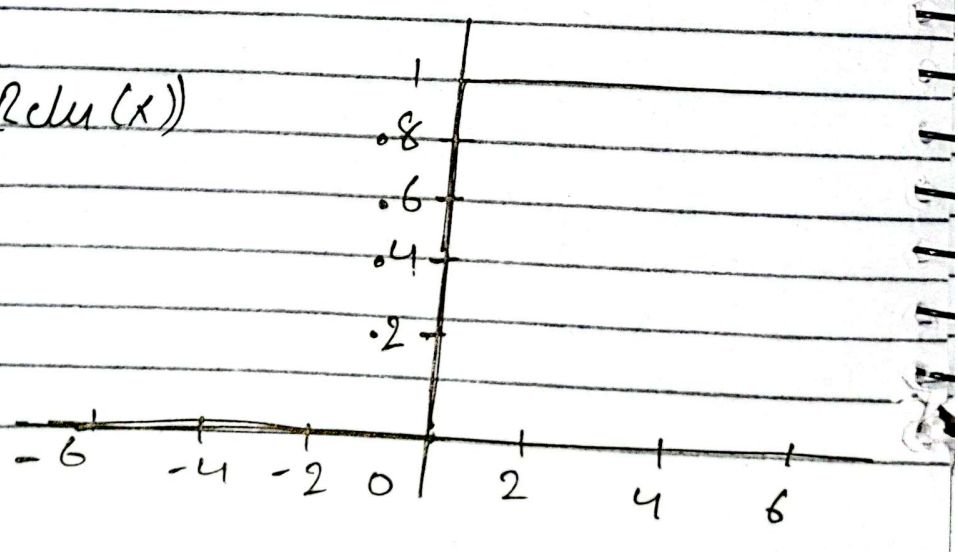
→ Don't handle (-ve) value
↳ means ReLU will die once (-ve) input is entered.

Inorder to solve dead Relu or dead Neuron we'll use Leaky Relu.

(3) Leaky Relu:

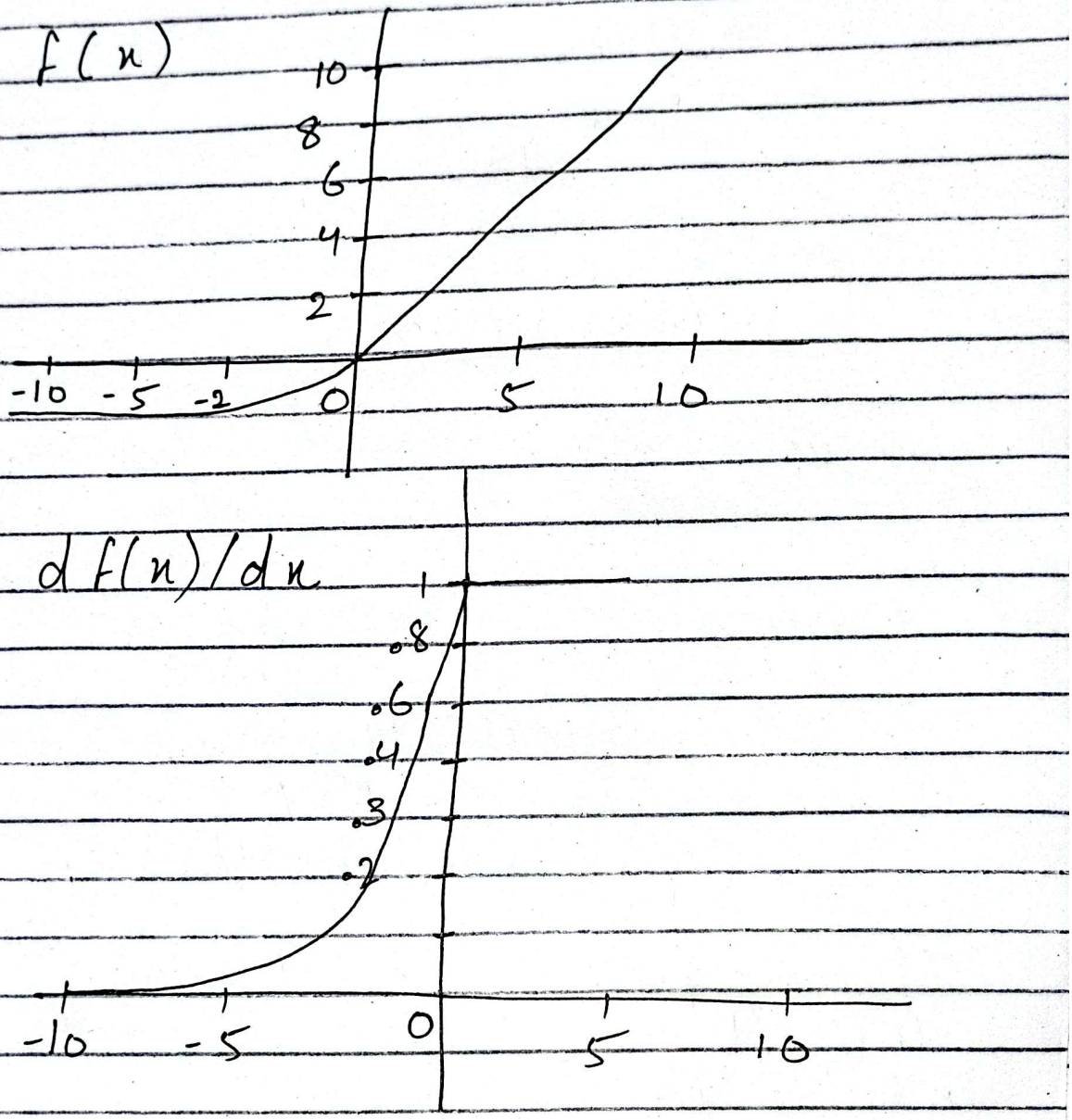


$\frac{d(LRelu(x))}{dx}$

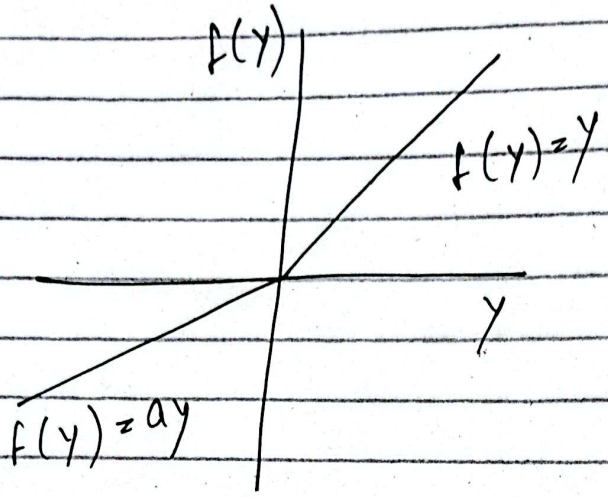


④ ELU (Exponential linear unit)

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



(5) PRelu:



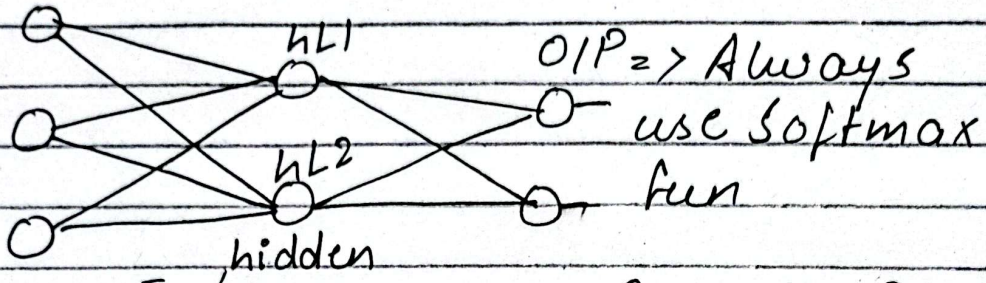
$$f(y) = \begin{cases} y, & \text{if } y > 0 \\ ay, & \text{if } y \leq 0 \end{cases}$$

Technique => which Activation function to use?

→ For Binary Classification:
- ↳ use (ReLU) in hidden layer.
- ↳ In output layer use (Sigmoid.) only.

If convergence is not happening with (ReLU) use (PReLU), or (ELU) in hidden layer.

→ For Multiclass classification:



In ↓ layers use ReLU or PReLU and ELU if convergence is not happening with (ReLU).

But try to use ReLU in hidden layer.

→ For Regression:

Use linear Activation function in output layer and ReLU or any other ReLU in hidden layers.

Loss func: —:

$$L = \frac{1}{n} \sum (\hat{y} - y)^2$$

SVM models:

$$L = \max(0, 1 - y \cdot \hat{y})$$

Binary Cross-Entropy

$$L = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

Multi Class Classification

$$L = -\sum y_i \log(\hat{y}_i)$$

(4) Loss Functions:

Diff b.w loss and Cost

$$\text{loss} = \frac{1}{n}(y - \hat{y})^2 \quad \text{Data} = 100$$

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

→ For Regression:

- → MAE
- → MSE
- → RMSE
- → Hubber loss

MSE:

Disadvantage: Not Robust to Outliers

MAE => Robust to Outliers.

→ For Classification:

Cross Entropy
- → Binary Cross Entropy
  - ↓
  - Binary classifica
- → Categorical Cross Entropy
  - Multi ↓
  - class classificati

→ Binary Cross Entropy:

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

⇓

Logistic Regression

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

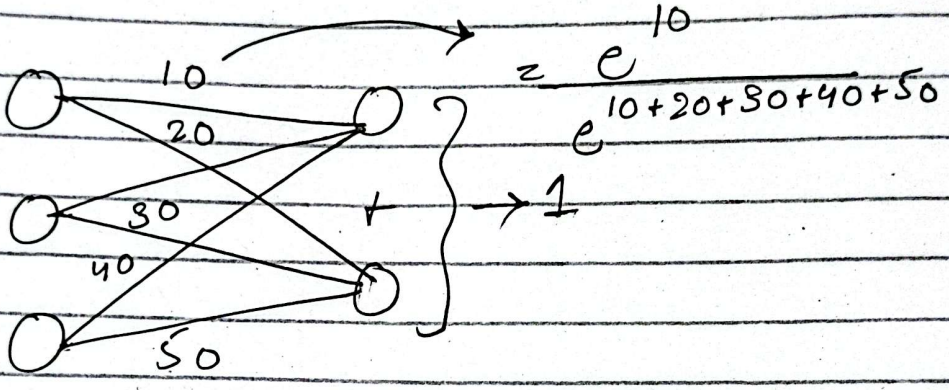
$$\hat{y} = \frac{1}{1+e^{-x}}$$

→ Categorical Cross Entropy:

$$L(x_i, y_i) = -\sum_{j=1}^{c} y_{ij} * \ln(\hat{y_{ij}})$$

$\hat{y}_{ij}$ = Softmax Activation function.

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$ } Softmax Activation



the output will be the one with higher probability in case of 2 or more outputs (Multiclass classification).

→ Conclusion:

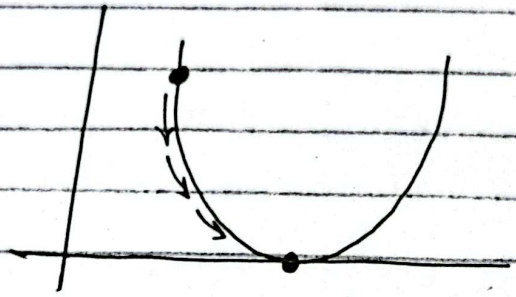
Classification
- Relu(HL), Softmax(OL) => Multiclass ↓ Categorical cross entropy
- Relu(HL), Sigmoid(OL) => Binary ↓ Binary cross entropy

Regression
- Relu(HL), linear act func(OL) ↓
- Loss func => MSE, MAE, Hubber Loss

→ Types of Optimizers

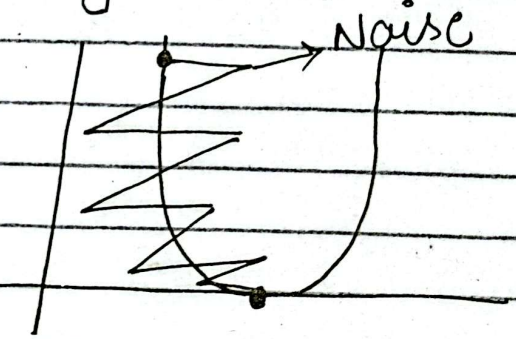
1. Gradient Descent
2. Stochastic Gradient Descent
3. Min Batch SGD
4. SGD with momentum.

GD =>



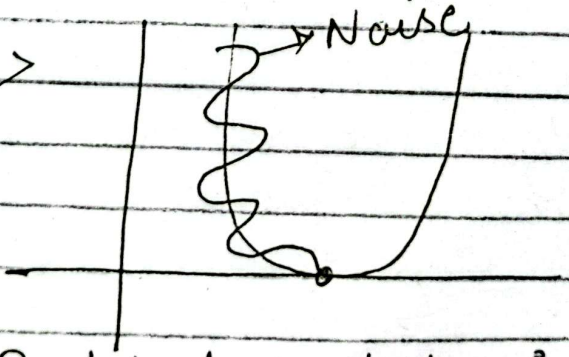
→ Requires Intense or Extensive Resource (Huge RAM).

SGD =>



Send individual value from data in 1 Epoch. → Much slower

MBSGD =>



Sends a Batch from data in 1 epoch. → a little faster but still

have some noise.

How to Reduce Noise?

↳ SGD with momentum.

As we know

$$W_t = W_{t-1} - \eta \frac{dL}{dW_{t-1}} \rightarrow (1)$$

We use Exponential weighted Average

$t_1, t_2, t_3, t_4 \ldots\ldots t_n$
$a_1 \quad a_2 \quad a_3 \quad a_4 \quad\quad a_n$

$V_{t1} = a_1$

$V_{t2} = \beta * V_{t1} + (1-\beta) * a_2 \rightarrow (2)$

$\beta \Rightarrow$ Hyperparameter
↳ ranges from 0 to 1.

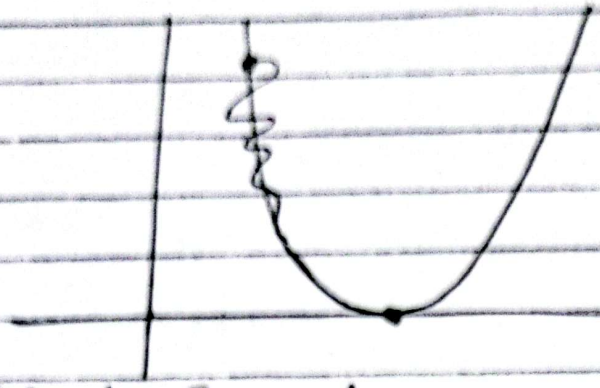
$V_{t3} = \beta * V_{t2} + (1-\beta) * a_2$

By using β as hyperparameter we will decide which value to choose for updating weight as it will reduce Noise.

$\beta = 0.95 \Rightarrow$ Put in (2)

$V_{t2} = (0.95) * V_{t1} + (1-\beta(0.95)) * a_2$

we'll choose this $\quad$ 0.05



Final E.N.A:

$$W_t = W_{t-1} - \eta \, Vdw_t$$

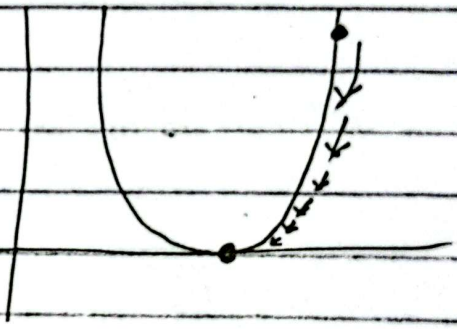
$$Vdw_t = \beta * Vdw_{t-1} + (1-\beta) * \frac{dL}{dw_{t-1}}$$

(5) Adagrd => Adaptive Gradient Descent

$$W_t = W_{t-1} - \eta \frac{dL}{dt-1}$$

$$\downarrow$$

$$Wt = Wt-1 - \eta' \frac{dL}{dt-1}$$



$$\eta' = \frac{\eta}{\sqrt{\alpha t + \epsilon}}$$

=> here value is decreasing as we reach Global minima, due to which our learning rate is adaptive. So that the learning rate is not fixed

$$\alpha t = \sum_{i=1}^{t} \left(\frac{dL}{dwt}\right)^2$$

what if αt increase greatly?
↳ $W_t \approx W_{t-1}$
vanishing GrD Problem

Then :-

(6) Ada delta and RMSProp

$$\eta' = \frac{\eta}{\sqrt{Sdw_t + t}}$$

$t \to$ now it will increase by a smaller value.

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

(7) Adam Optimizer: (Best Optimizer)

Momentum + RMSprop.

So Finally best option or formula to update weight value:

$$W_t = W_{t-1} - \eta' Vdw$$

$$bt = W_{bt-1} - \eta' Vdb \rightarrow$$ To update bias value.

So what we did:

→ Smoothing (Reducing Noise).
→ Learning Rate (Making it Adaptive).

ANN Implementation

→ Tensorflow important importing code

```
from Tensorflow.Keras.models import Sequential

from Tensorflow.Keras.layers import Dense, ReLU, Dropout → helps with overfitting
                                    ↳ uses it inside neurons or hidden layers.
```

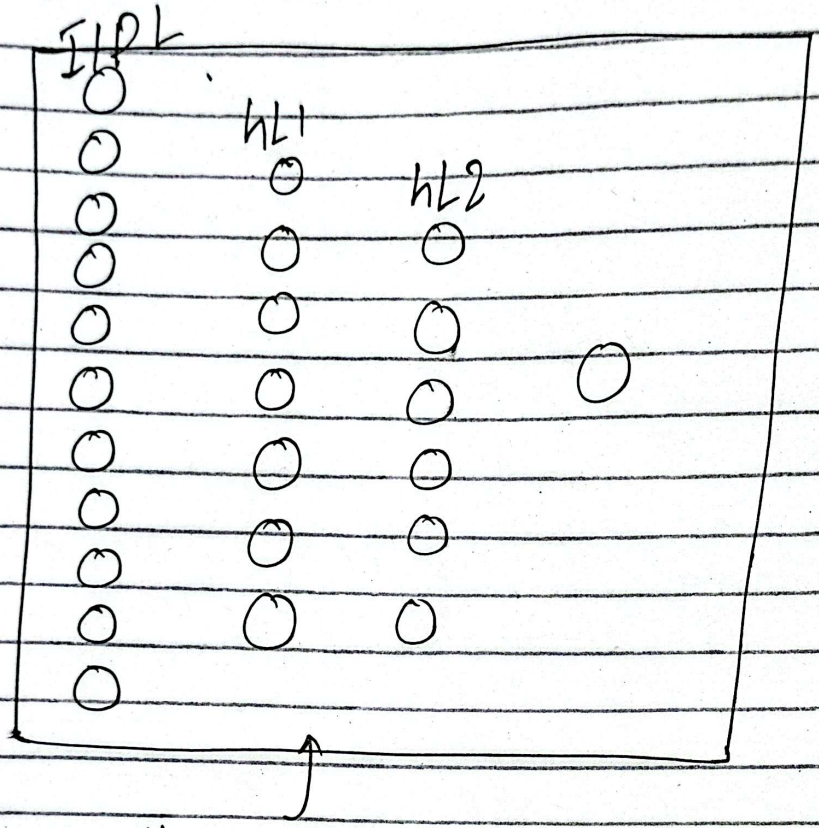
helps to create neurons or hidden layer and also I/P or O/P layers

```
classifier = Sequential()
```

Creates a space to make neural network

Now to add input, hidden and Output layer we'll use Dense

Dense
classifier.add(Dense(unit=11, activation='relu')).



classifier.add(Dense(units=7, activation=RELU))

Creates 1 hidden layer with 7 units if also we make another layer with 6 units then

CNN (Convolution Neural Network)

CNN is a type of DL model that is mainly used to analyze images.

→ It works by automatically detecting important features (like edges, shapes and textures) from images using convolutional layers, so it can recognize patterns and objects without manual feature extraction.

Images:

each pixel ranges from 0-255

RGB

	40	0	255	
	100	32	60	

(5 × 5) pixel image

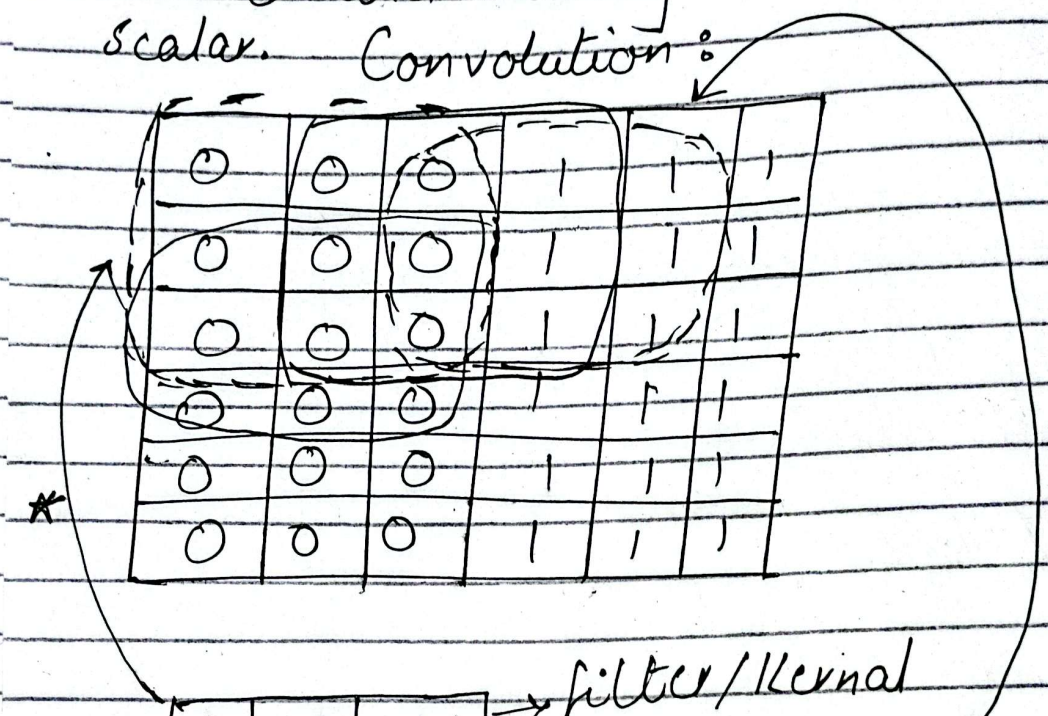
row × column

0 → Black

255 → white

① Convolutional Operations

In first step of CNN we try to convert the pixels of an image into 0 and 1 using min, max scalar. Convolution:



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

→ filter / Kernal

→ Now if => Stride = 1 and then 2

(Horizontal filter edge)
(Sobel)

$1(0) + 2(0) + (1)(0) + 0(0) + 0(0) + 0(0) + (-1)(0) + (-2)(0) + (-1)(0) = 0$

$\Rightarrow 0$

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

$\Rightarrow 0$

→ Why we multiply filter with image pixel

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

=> vertical edge filter (Sobel)

→ Filters / Kernel helps to detect features like: edges, textures and patterns.

if n = 6 ; f = 3

what will be the size of O/P image

= n - f + 1

~~n - f + 1~~ => 3 + 1 = 4 => (4 x 4)

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

→ 4 x 4

As we see that the size of our image is decreased due to which it means that some of the information is lost to prevent that we use a concept => Padding.

Padding => Building a compound around images.

New formula to find size of O/P image after using padding:

$$= n + 2p - f + 1$$
$$= 6 + 2(1) - 3 + 1$$
$$= 8 - 3 + 1$$
$$= 9 - 3$$
$$= 6$$

$n = 6$
$p = 1$
$f = 3$

We need to update the values of filter on the base of input image by using backpropogation.

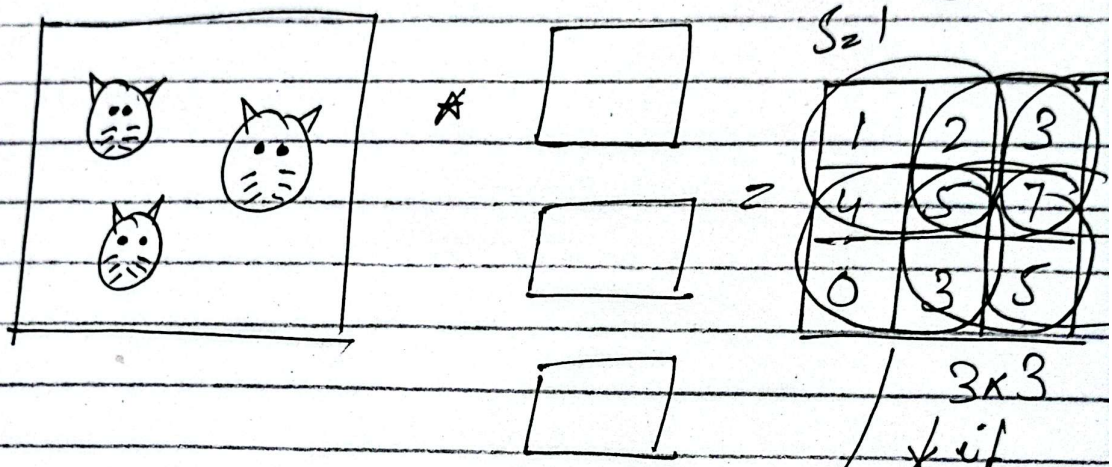
After getting output image we apply relu activation function on each value of image's pixel.

if we give the value of Stride > 1 then O/P image size formula will be:

$$= \frac{n + 2p - f + 1}{s}$$

(2) Max pooling:

Other 2 types of Pooling:
- Avg pooling
- Min pooling

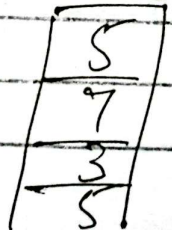


Max Pooling:

Max Pool will contain the highest values of pixels in order have clear data.

(3) Flattening layer:

Just Converting the Pooling into 1D Data.



or combine all O/P. then we can throw this as I/P in NN

Conclusion: