

<https://www.youtube.com/watch?v=d2kxUVwWWwU>

Motive : { Clear their Basics, Maths, Interview Preparation }

Agenda

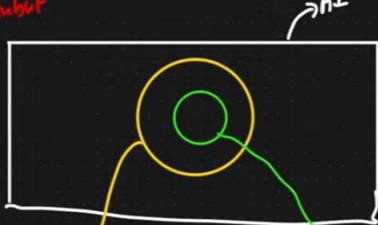
- (1) Deep Learning → Perceptrons { AI VS ML VS DL VS DS }
- (2) Forward Propagation
- (3) Backward Propagation
- (4) Loss function
- (5) Activation functions
- (6) Optimizers. what is exactly deep learning why are these terms coming up

Prerequisite

- (1) Python
- (2) ML
- (3) Stats ^

AI VS ML VS DL VS DS

{ ML is a subset
of AI }



{ application which can do its own task
without any human Intervention }

- ① Netflix App
- ② Self Driving Cars
- ③ Amazon Application
- ④ Sofia

RTX
3090

{
ML → Statistical tools to analyze
the data, visualize the data,
predictions, forecasting, clustering

Researchers (1958)



→ Researchers (1958)
=

Multi Layered

{ Mimic the human brain }

Neural Net

Perceptron }



① 2005 → ORKUT, FACEBOOK, Instagram, WhatsApp, LinkedIn, Twitter

DATA → Exponentially ↑↑↑

2008 → { Big DATA } → Efficiently

2013 → Company had huge amount of Data
↓ {AI → popular}

Scanners Products

```

graph TD
    A[Panasonic : AC's, TV's, Refrigerator {Data}] --> B[Model → Reduce the Electricity Bills]
    A --> C[Subscription Basis]
    B --> D["{Curate Revenue, Better Decisions}"]
    C --> D
  
```

The diagram illustrates a flow from Panasonic's product range to a model for reducing electricity bills, which in turn leads to a subscription basis and ultimately to curated revenue and better decisions.

② Hardware Advancement (Nvidia) → GPU's → TRAINING THE MODEL

Cpu's → cost ↓↓↓

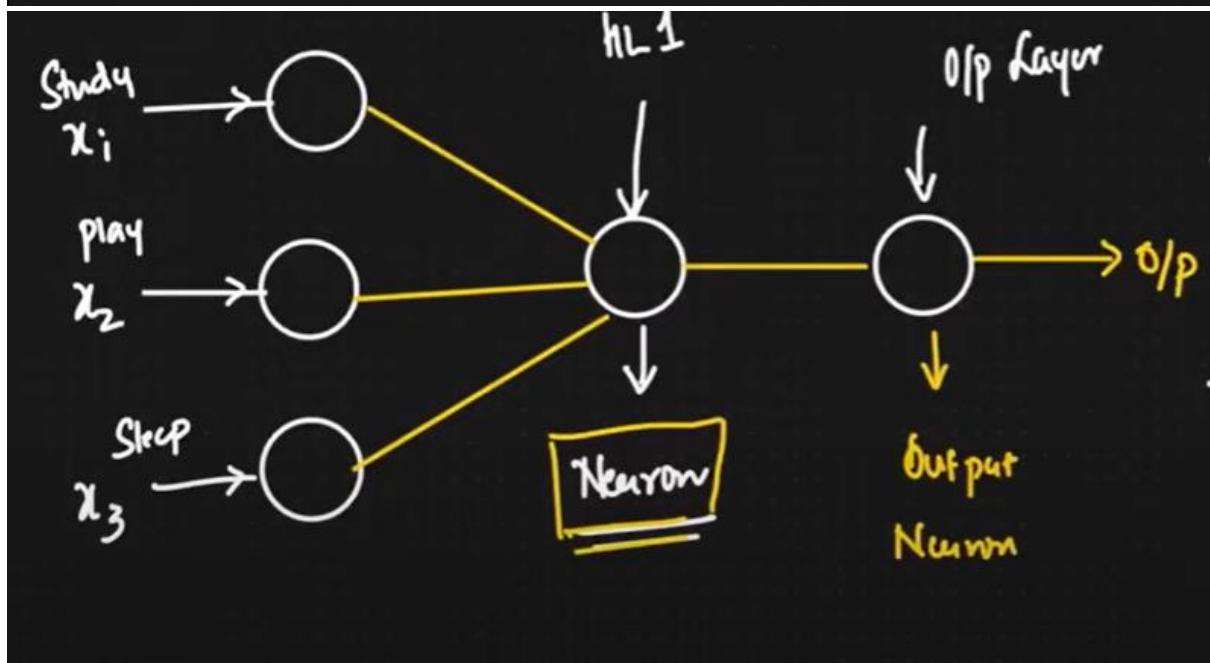
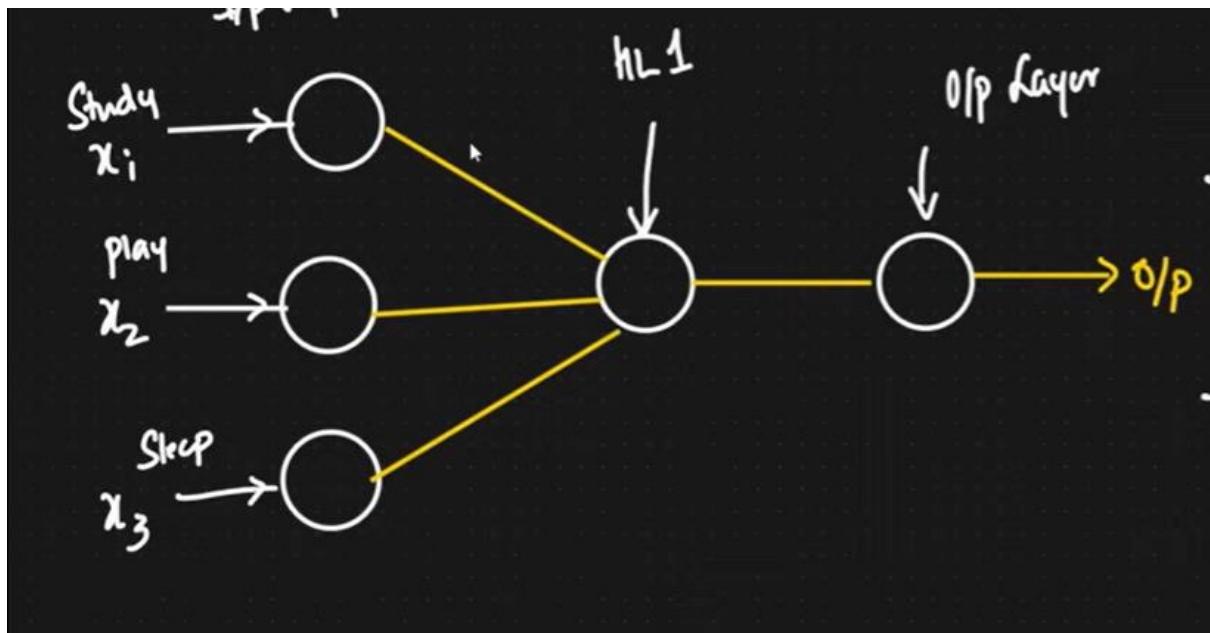
④ Perceptron { Single Layered Neural N/w }.

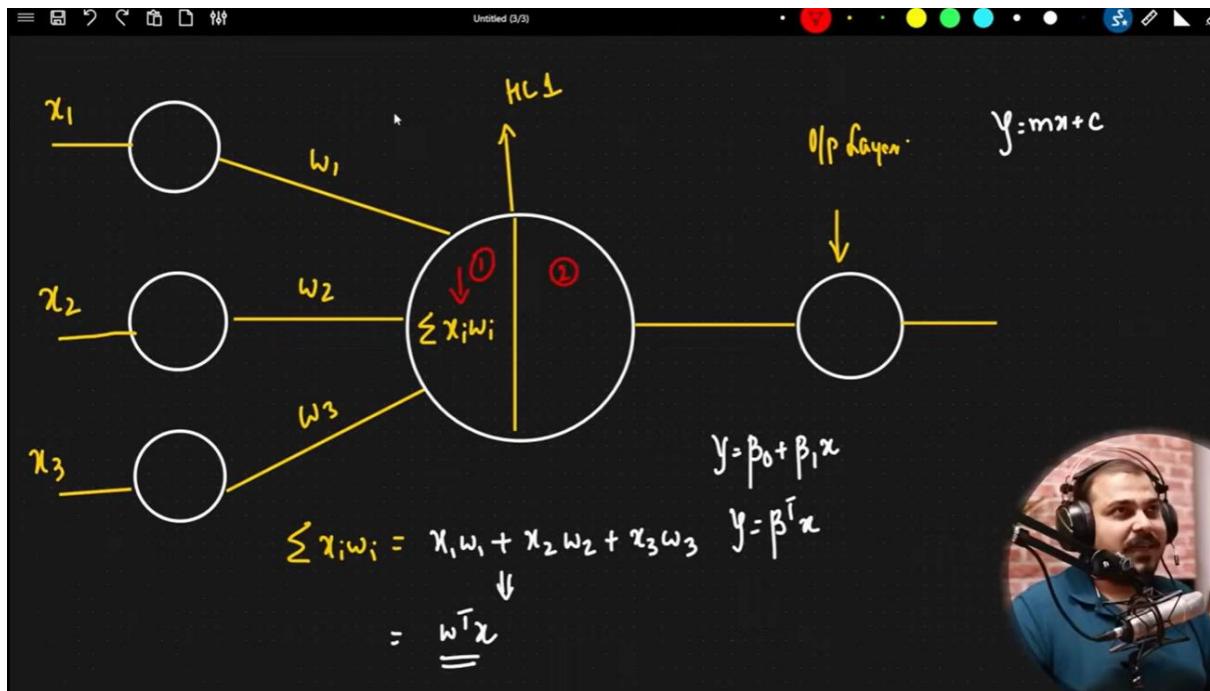
Input factors

Dataset

Binary classification

	Study	Play	Sleep	Pass/Fail
→ 7	3	7	1	
→ 2	5	8	0	
→ 4	3	7	1	

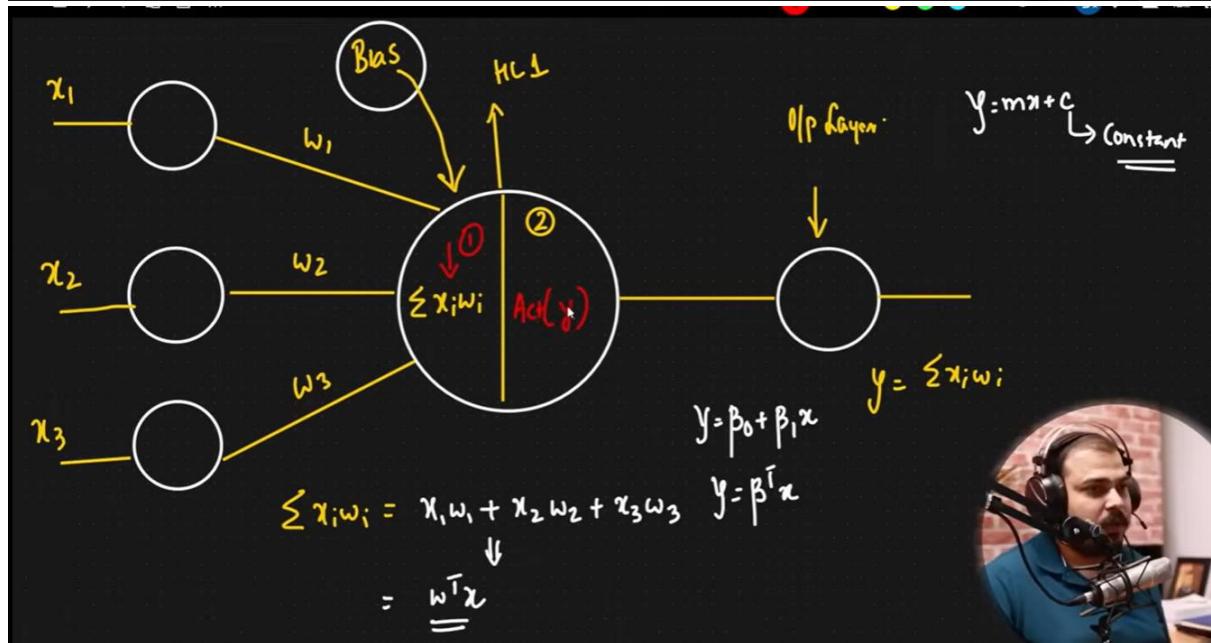
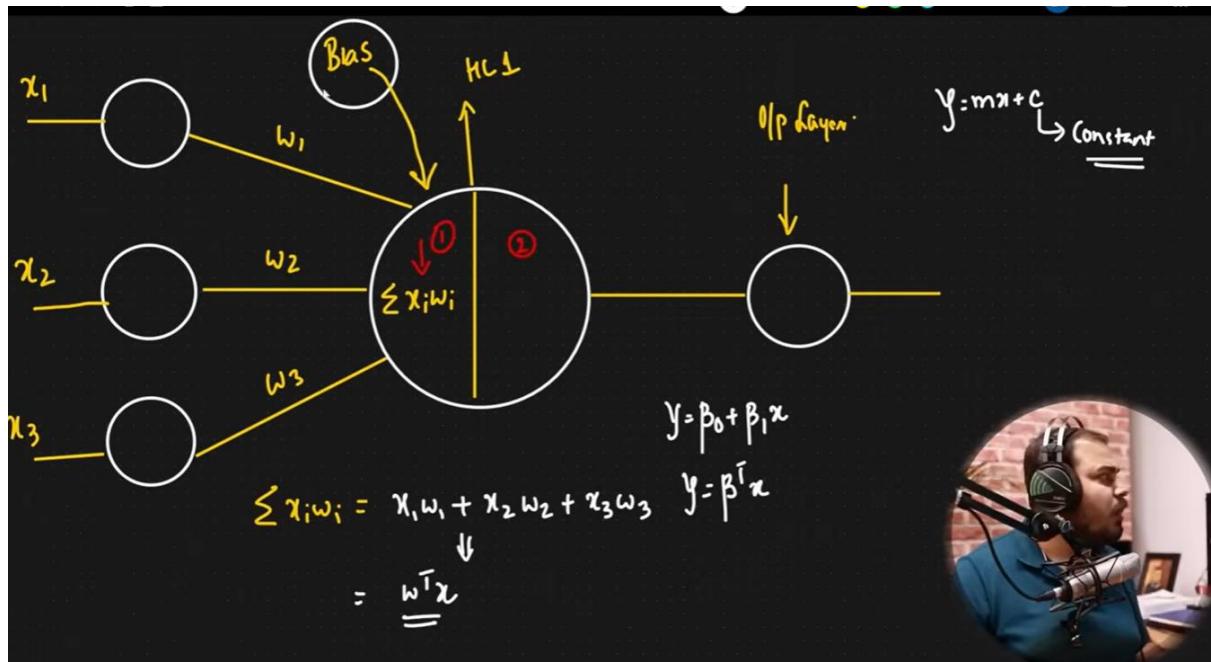




Neurons will tell or give signal at which level should get activated eg when you place a hot object in hand neurons will give signal to the brain to get activate to remove the object

During training of nn we need to decide which values to assign as weight to the neurons, so neurons will signal based on value to which level it should get activated

Weight should not be zero so in order to overcome let's add parameter called bias



We will pass on activation function on top of y

Sigmoid activation function

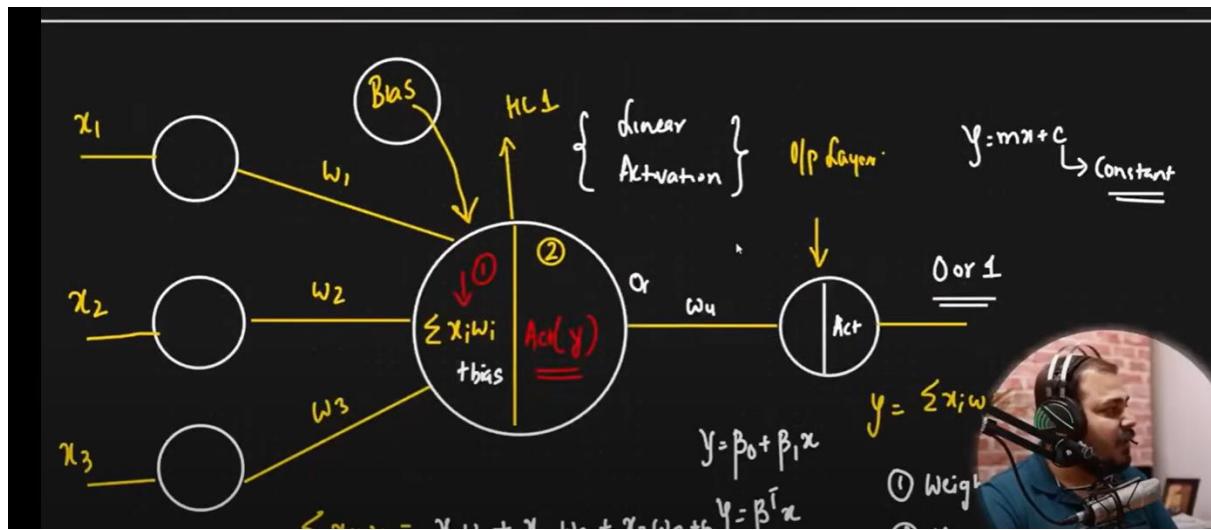
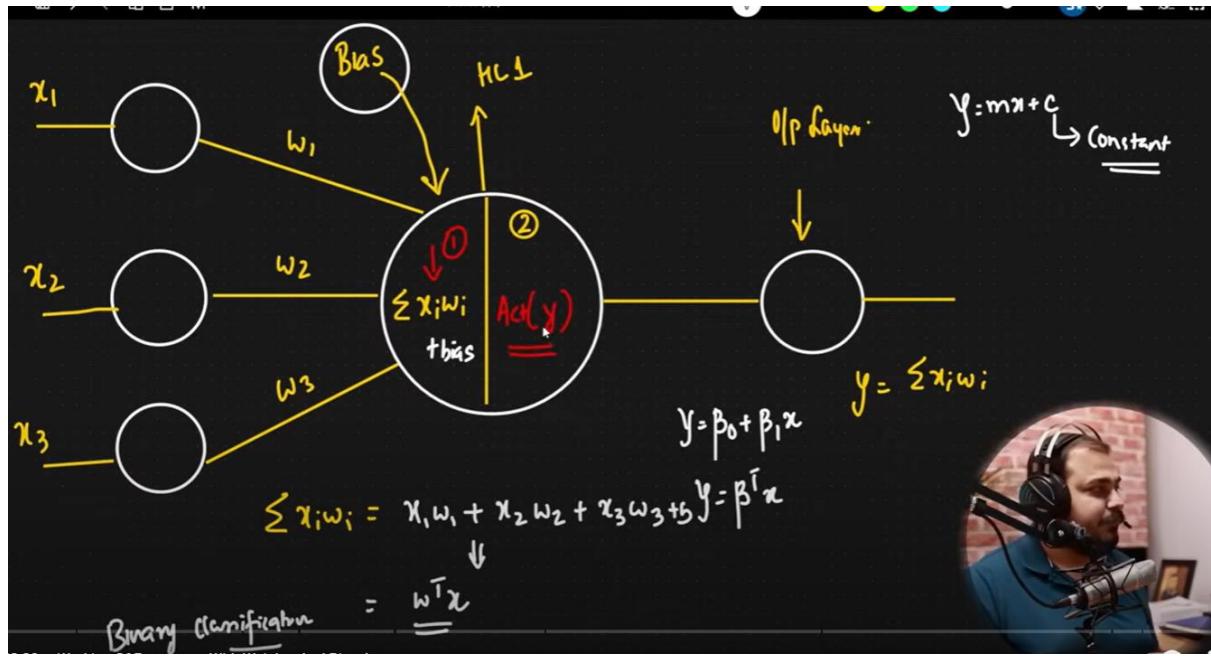


Diagram illustrating the working of a perceptron with weights and bias:

$$\sum x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 + b \quad y = \beta^T x$$

Binary classification = $\underline{w^T x}$

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

0 to 1

$$\begin{cases} \geq 0.5 \rightarrow 1 \\ < 0.5 \rightarrow 0 \end{cases}$$

0 or 1

Working Of Perceptron With Weights And Bias

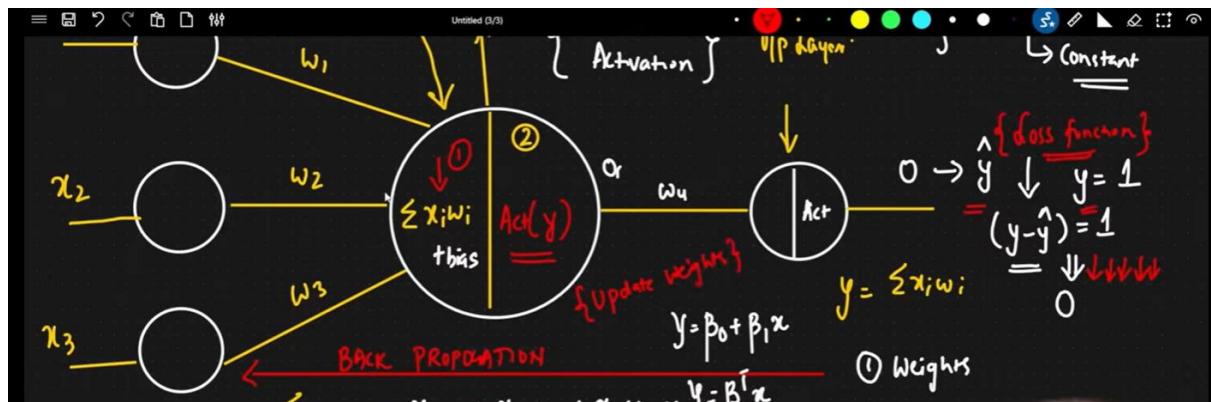
Legend:
 ① Weights
 ② Neurons
 ③ Activations

Forward propagation

We can take a input we take a input multiplied by weights add a bias and then activate neurons get the outputs it is called forward propagation

Backward propagation

In order to minimize the error (loss function) we need to update the weights then it is back propagation

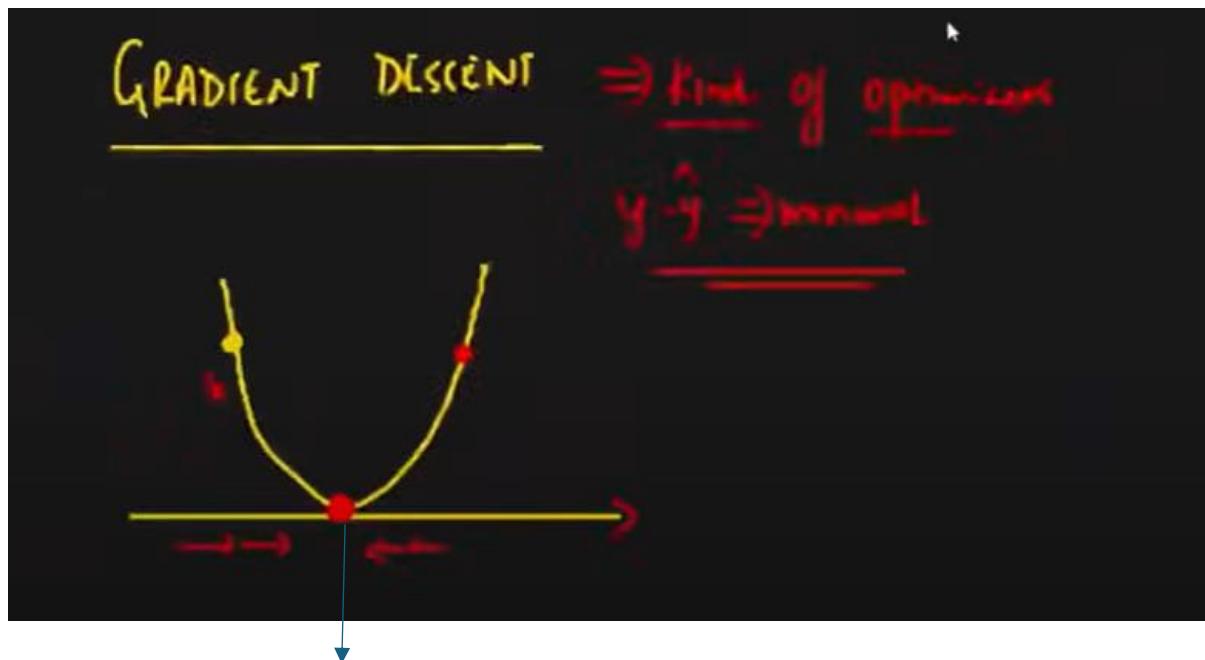


Optimizers

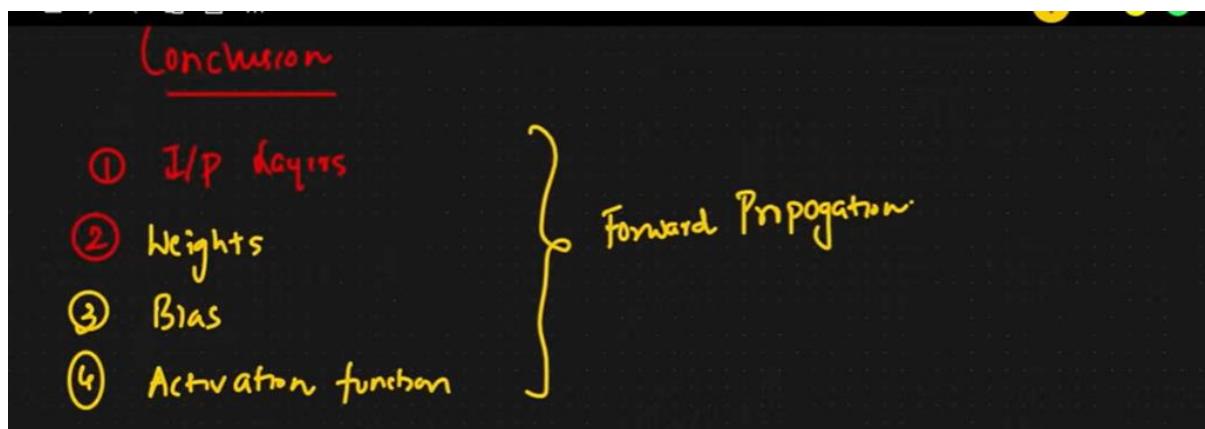
How we update the weights that is done by optimizers

Gradient descent is a optimizers

It is combination of both forward and backward in linear regression backwards weights will get updated and in forward weights will get multiplied with the coefficients



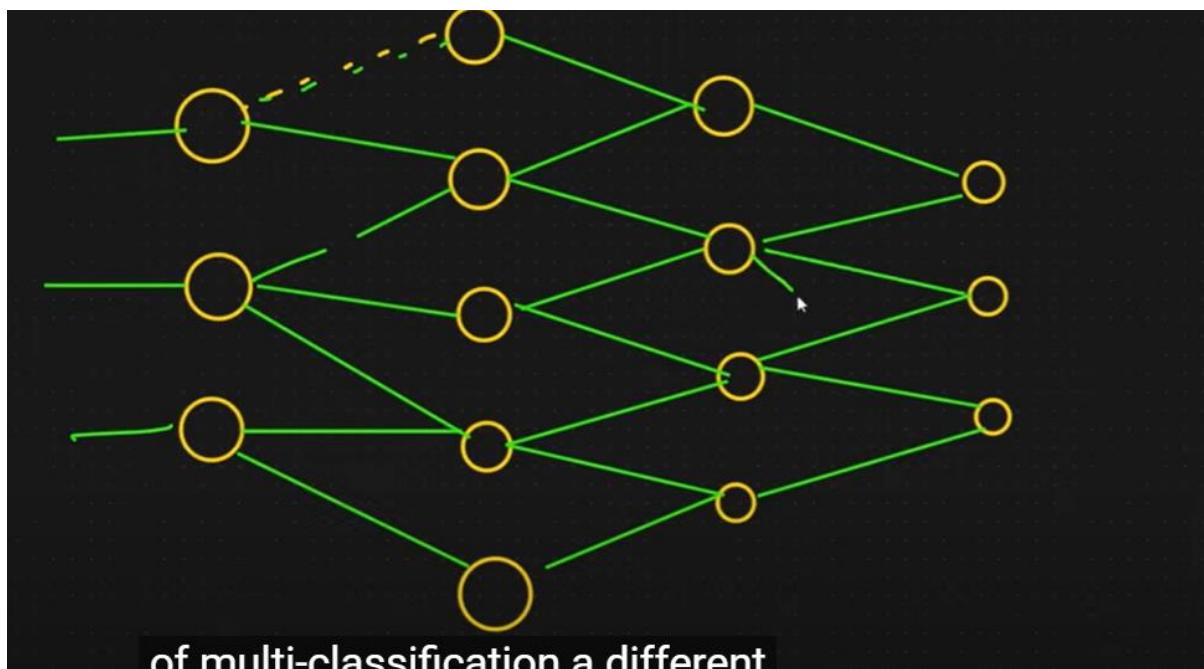
Global minima



- ⑤ Loss function $\{ \hat{y} - y \}$ w.r.t
 - ⑥ Optimizer
 - ⑦ Update the weight
- Backward
Propagation

- ① KNN
- ② CNN
- ③ RNN
- ④ Object Detection

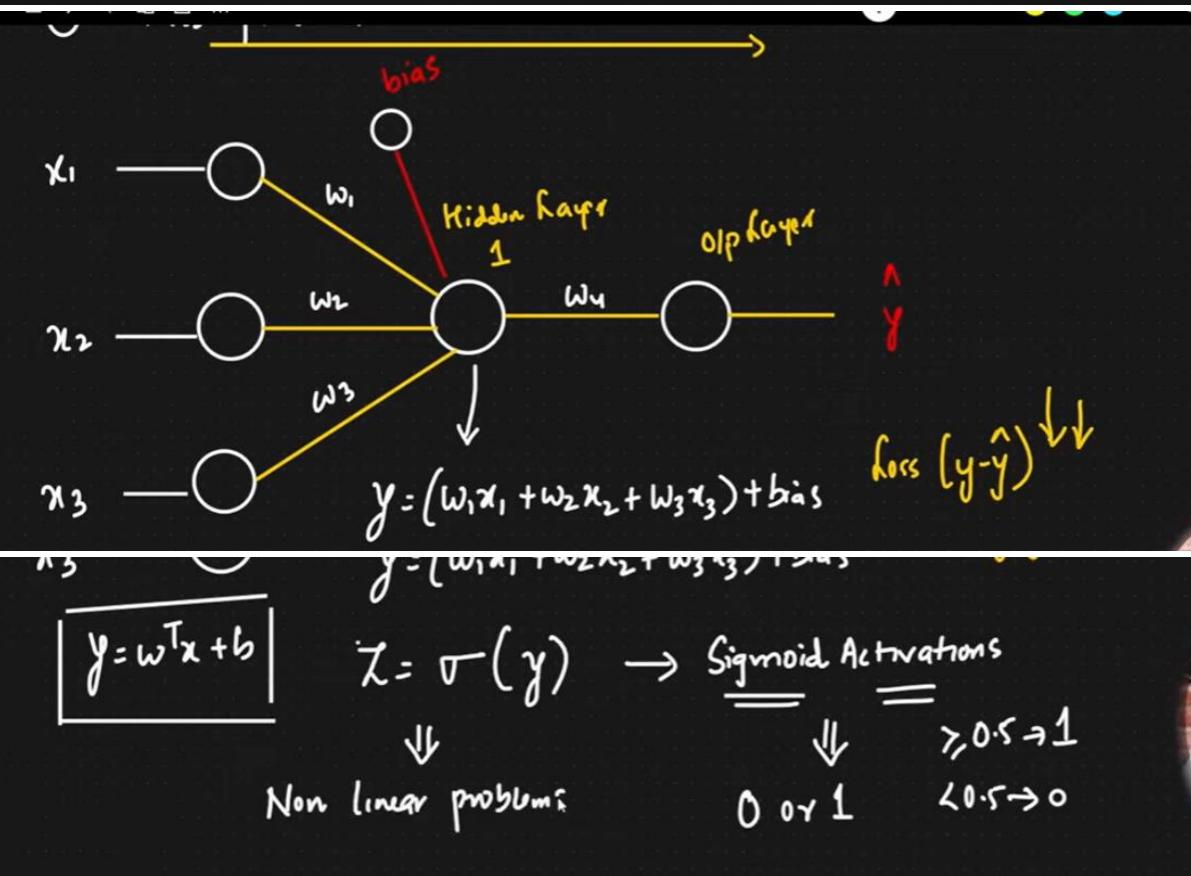
Multilayer neural network



Day 2 - Deep Learning.

Agenda

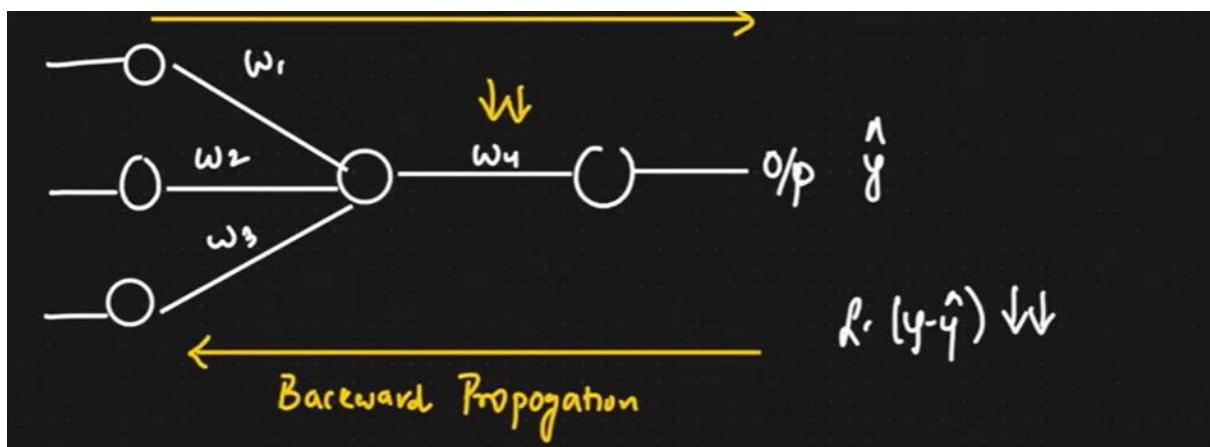
- ① Forward Propagation
- ② Chain Rule of Derivation
- ③ Vanishing Gradient Problem
- ④ Loss functions



Backpropagation

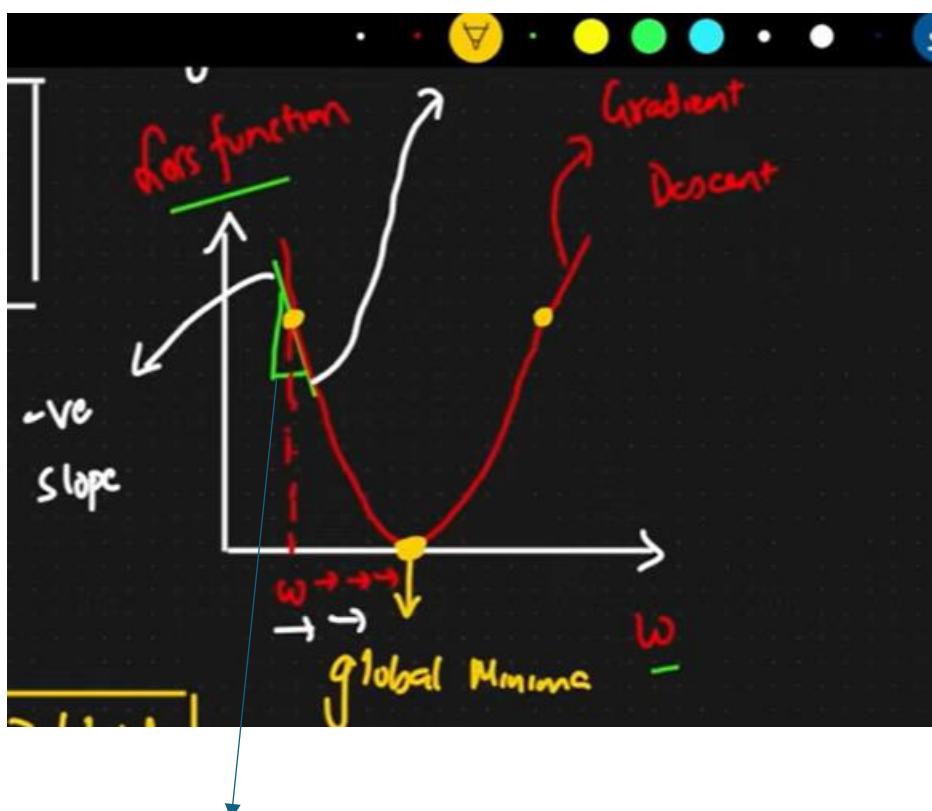
Updation of weights happens in backpropagation

Chain rule of differentiations



Weight update formula \rightarrow Learning Rate

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



Tangent line to find slope if the line going downwards then it is negative slope we need to increase the weights to achieve global minima (i.e., minimal loss function)

$$\frac{\partial L}{\partial w_{old}} \quad \boxed{-ve \text{ slope}}$$

$$w_{new} = w_{old} - \eta (-ve) \quad \boxed{w_{new} >> w_{old}}$$

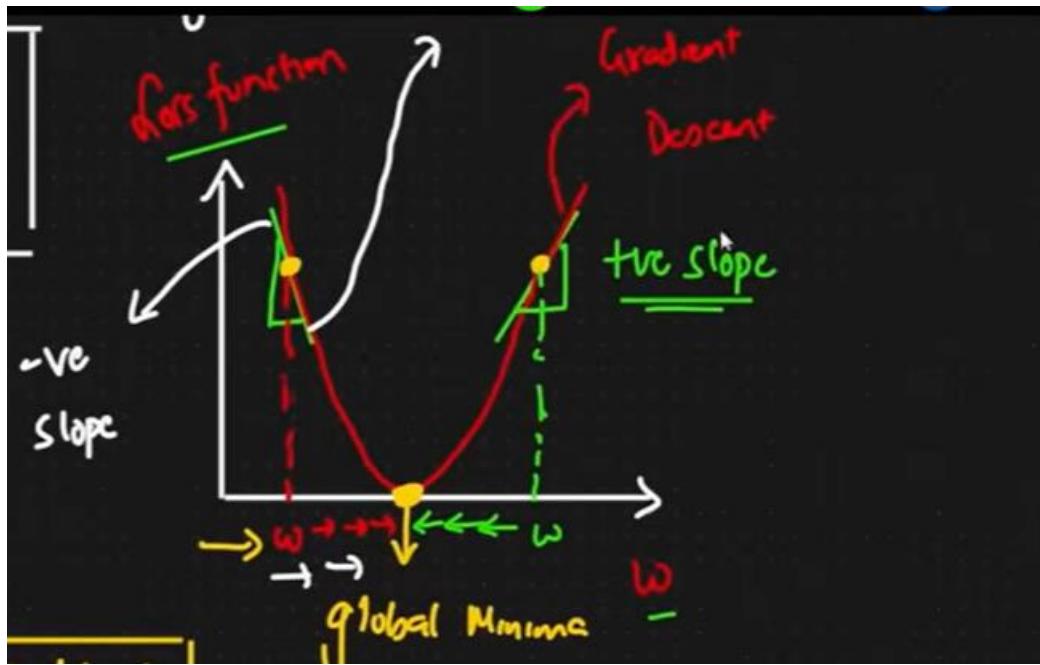
$$= w_{old} + \eta (+ve)$$

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

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

$$\Downarrow \quad \uparrow$$

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

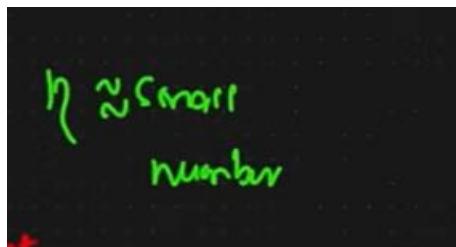


We need to decrease the weight to achieve global minima when slope is positive

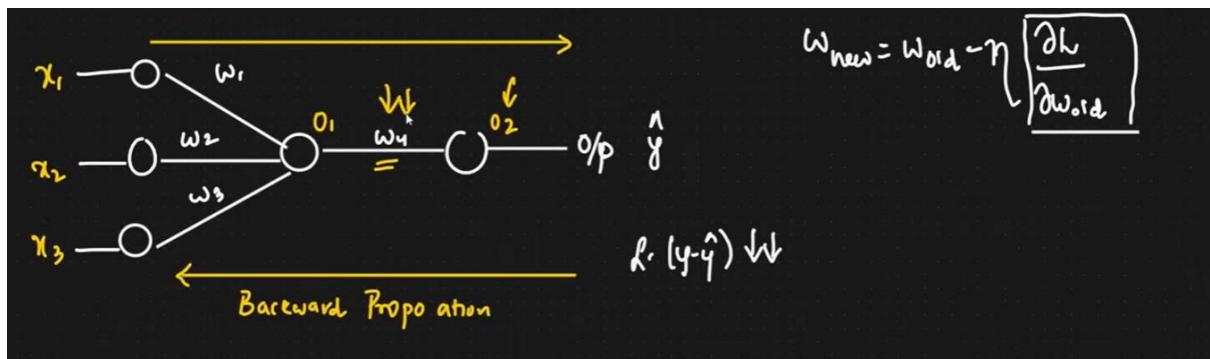
Learning rate should always be a small number to slowly achieve global minima

Learning rate can be round =0.01

Bigger number can take higher steps and vary here and after



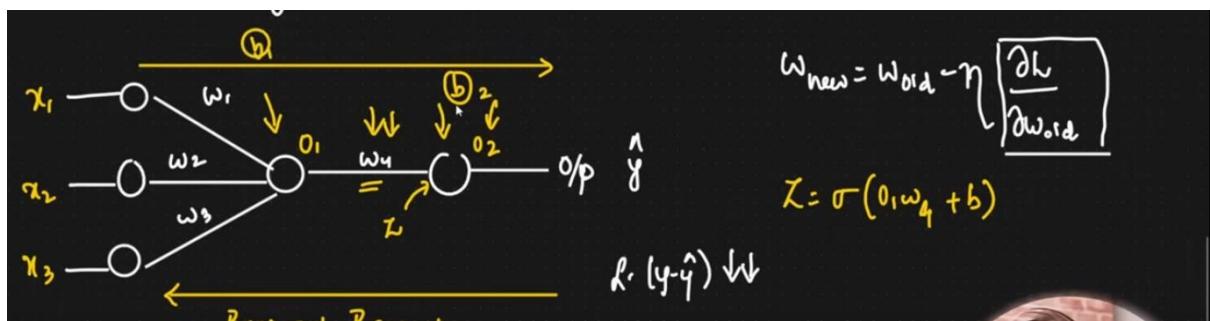
Chain rule of differentiation



$$\boxed{w_{4,\text{new}} = w_{4,\text{old}} - \eta \left[\frac{\partial L}{\partial w_{4,\text{old}}} \right]}$$

{ Chain Rule of Derivative }

$$\frac{\partial L}{\partial w_{4,\text{old}}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial w_4}$$



Derivative of loss function is dependent on o_2 and o_2 is dependent on o_1 and o_1 is dependent on w_1 etc, it is a chain kind of thing

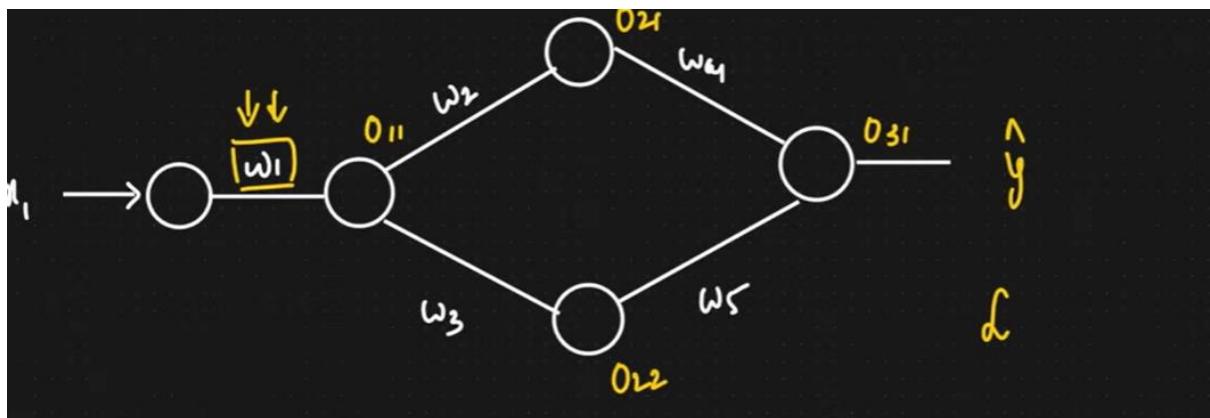
$$\Rightarrow \boxed{b_{2,\text{new}} = b_{2,\text{old}} - \eta \left[\frac{\partial L}{\partial b_{2,\text{old}}} \right]}$$

Loss

$$\frac{\partial L}{\partial w_{1, \text{old}}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial o_1} * \frac{\partial o_1}{\partial w_{1, \text{old}}}$$

\downarrow

$$\frac{\partial o_2}{\partial w_4} * \frac{\partial w_4}{\partial o_1}$$

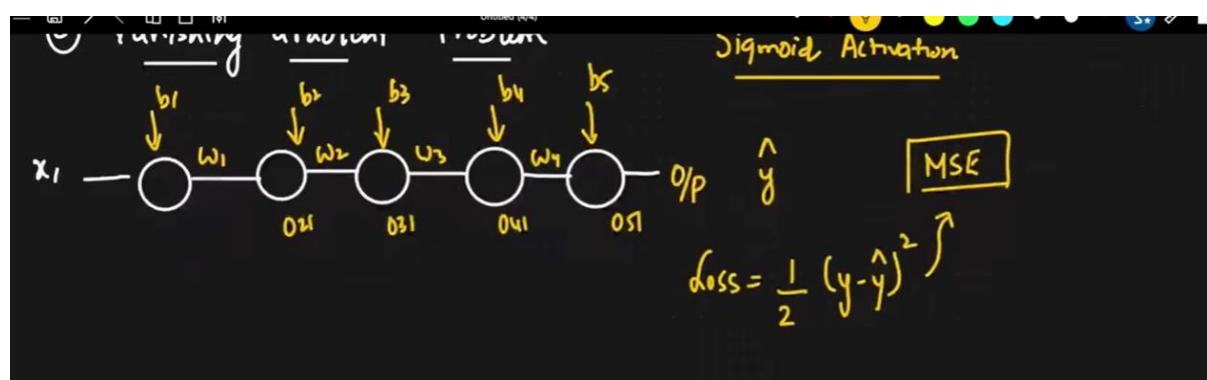


$$w_{1,\text{new}} = w_{1,\text{old}} - \eta \frac{\partial L}{\partial w_{1,\text{old}}}$$

$$\begin{aligned} \frac{\partial L}{\partial w_{1,\text{old}}} &= \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1,\text{old}}} \right] \\ &\quad + \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{22}} * \frac{\partial o_{22}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1,\text{old}}} \right] \end{aligned}$$

Chain rule of derivatives

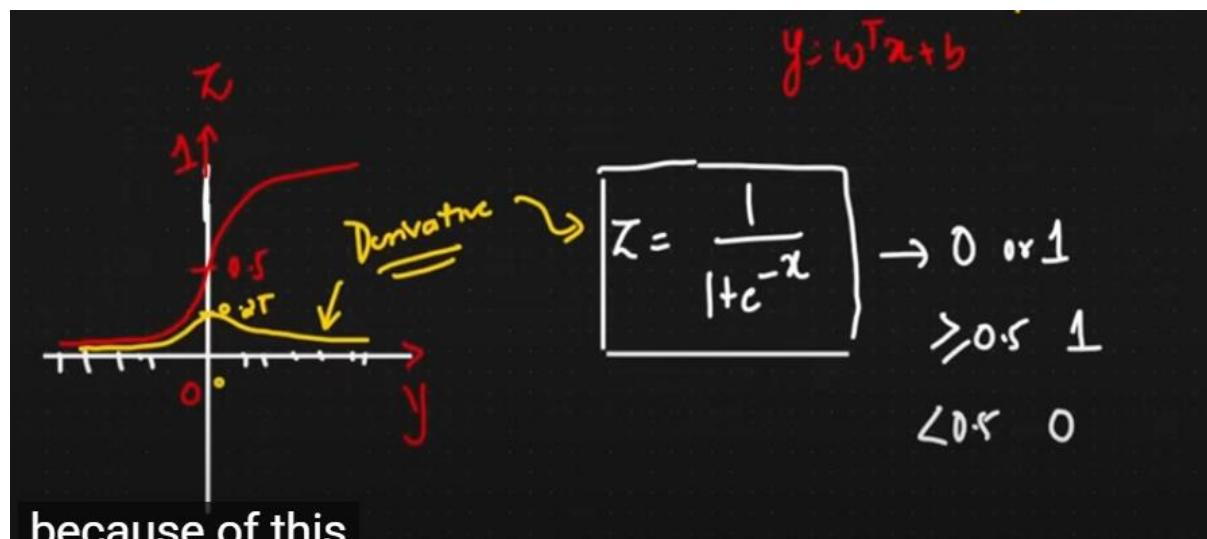
Vanishing gradient descent programme



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

$$\frac{\partial L}{\partial w_{i,\text{new}}} = \frac{\partial L}{\partial o_{51}} * \frac{\partial o_{51}}{\partial o_{41}} * \frac{\partial o_{41}}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial w_i}$$

Sigmoid activation function



Derivative of sigmoid function ranging from 0 to 0.25

Derivative:

$$0 \leq \sigma(y) \leq 0.25$$

$$o_{51} = \sigma \left[(o_{41} * w_4) + b \right]$$

↓ Sigmoid Activation

The value will range from 0 to 0.25 because we multiply with sigmoid activation function

$$= 0.25 * 0.15 * 0.10 + 0.05 * 0.02 \quad \boxed{y = w^T x + b}$$

If we keep compute like this and multiply with sigmoid activation function then values will get smaller what we do next then we will update our weights

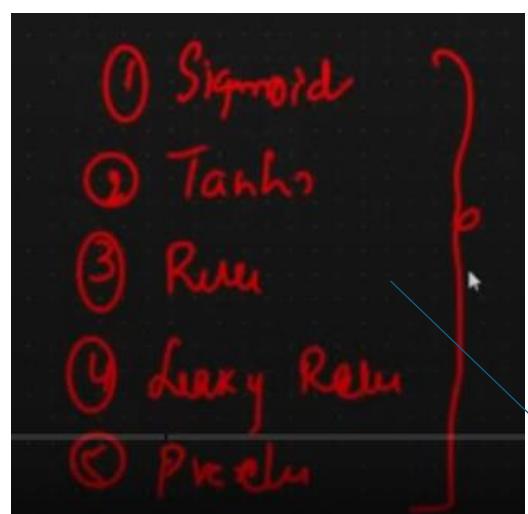
$$w_{\text{new}} = w_{\text{old}} - \eta \left(\text{small number} \right)$$

$w_{\text{new}} \approx w_{\text{old}}$

 $\Rightarrow \text{Vanishing gradient Problem}$

If there is no change in weights it is called vanishing gradient problem

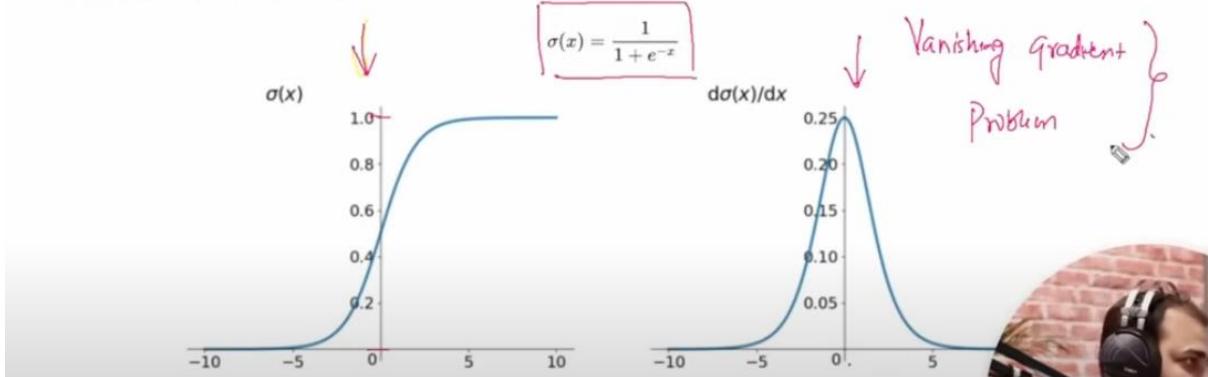
To solve this problem use another activation function



different types of activation function

1. Sigmoid function

The function formula and chart are as follows



1) When the input is slightly away from the coordinate origin, the gradient of the function becomes very small, almost zero. In the process of neural network backpropagation, we all use the chain rule of differential to calculate the differential of each weight w . When the backpropagation passes through the sigmoid function, the differential on this chain is very small. Moreover, it may pass through many sigmoid functions, which will eventually cause the weight w to have little effect on the loss function, which is not conducive to the optimization of the weight. This problem is called gradient saturation or gradient dispersion.

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

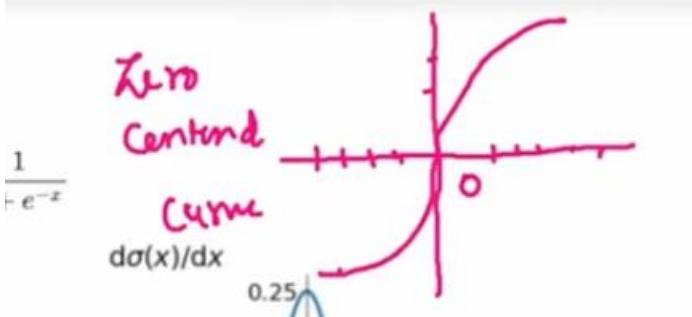
3) The sigmoid function performs exponential operations, which is slower for computers.

Advantages of Sigmoid Function :-

1. Smooth gradient, preventing "jumps" in output values.
2. Output values bound between 0 and 1, normalizing the output of each neuron.
3. Clear predictions, i.e. very close to 1 or 0.

Sigmoid has three major disadvantages:

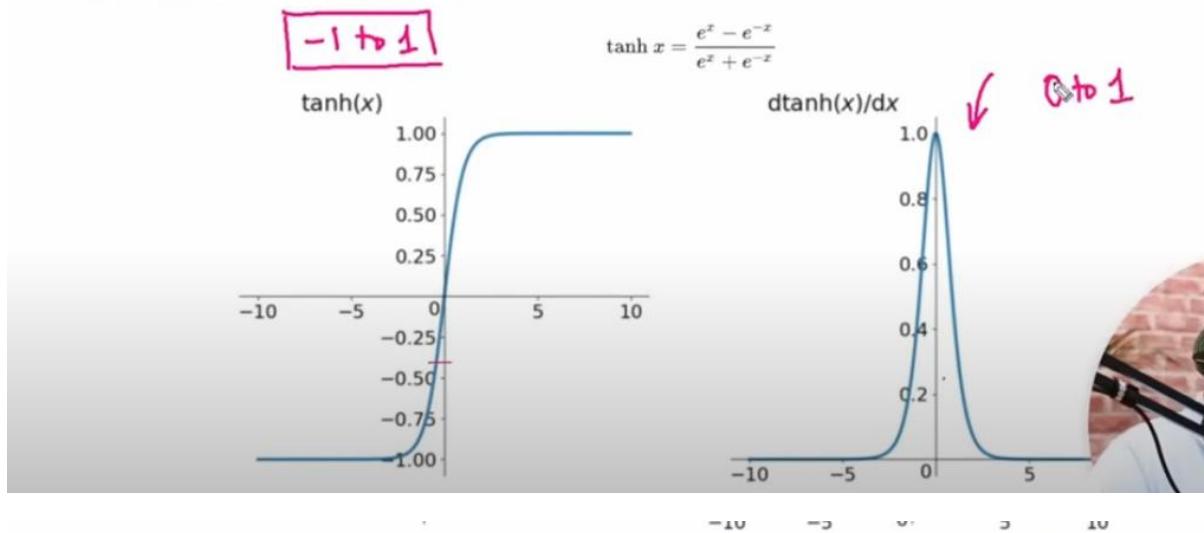
- Prone to gradient vanishing
- Function output is not zero-centered
- Power operations are relatively time consuming



If we have zero centred curve then weight updation becomes very faster

2. tanh function

The tanh function formula and curve are as follows



Tanh is a hyperbolic tangent function. The curves of tanh function and sigmoid function are relatively similar. Let's compare them. First of all, when the input is large or small, the output is almost smooth and the gradient is small, which is not conducive to weight update. The difference is the output interval.

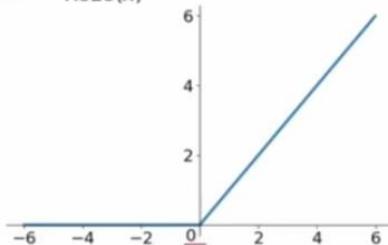
The output interval of tanh is 1, and the whole function is 0-centric, which is better than sigmoid.

In general binary classification problems, the tanh function is used for the hidden layer and the sigmoid function is used for the output layer. This is not static, and the specific activation function to be used must be analyzed according to the specific problem, or it depends on debugging.

3. ReLU function

ReLU function formula and curve are as follows

$\downarrow h(x)$
~~X~~ \rightarrow Dead
O
O

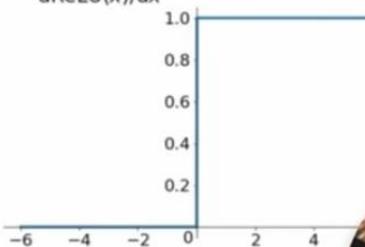


$$\frac{\partial h}{\partial w_{old}} = 0$$

ReLU = $\max(0, x)$

$w_{new} \approx w_{old}$

dReLU(x)/dx



The ReLU function is actually a function that takes the maximum value. Note that this is not fully interval-derivable, but we can take the figure above. Although ReLU is simple, it is an important achievement in recent years.

The ReLU (Rectified Linear Unit) function is an activation function that is currently more popular. Compared with the sigmoid function, it has the following advantages:

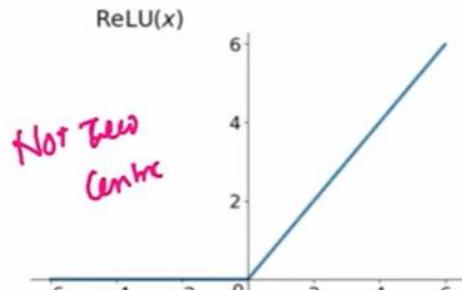
- 1) When the input is positive, there is no gradient saturation problem.

3. ReLU function

Sigmoid, tanh

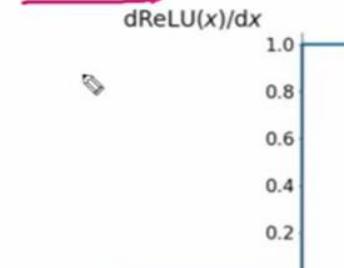
ReLU function formula and curve are as follows

$\downarrow h(x)$
~~X~~ \rightarrow Dead
O
O



$$\text{ReLU} = \boxed{\max(0, x)}$$

\rightarrow quicks



The ReLU (Rectified Linear Unit) function is an activation function that is currently more popular. Compared with the sigmoid function and the tanh function, it has the following advantages:

- 1) When the input is positive, there is no gradient saturation problem.
- 2) The calculation speed is much faster. The ReLU function has only a linear relationship. Whether it is forward or backward, it is much faster than sigmoid and tanh. (Sigmoid and tanh need to calculate the exponent, which will be slower.)

Of course, there are disadvantages:

- 1) When the input is negative, ReLU is completely inactive, which means that once a negative number is entered, ReLU will die. In this way, in the forward propagation process, it is not a problem. Some areas are sensitive and some are insensitive. But in the backpropagation process, if you enter a negative number, the gradient will be completely zero, which has the same problem as the sigmoid function and tanh function.

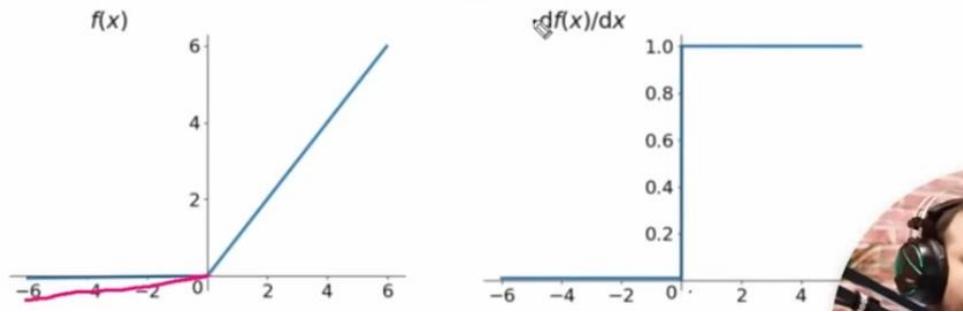
- 2) We find that the output of the ReLU function is either 0 or a positive number, which means that the ReLU function is not a 0-centric function.



4. Leaky ReLU function

Solve dead Neuron (ReLU)

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

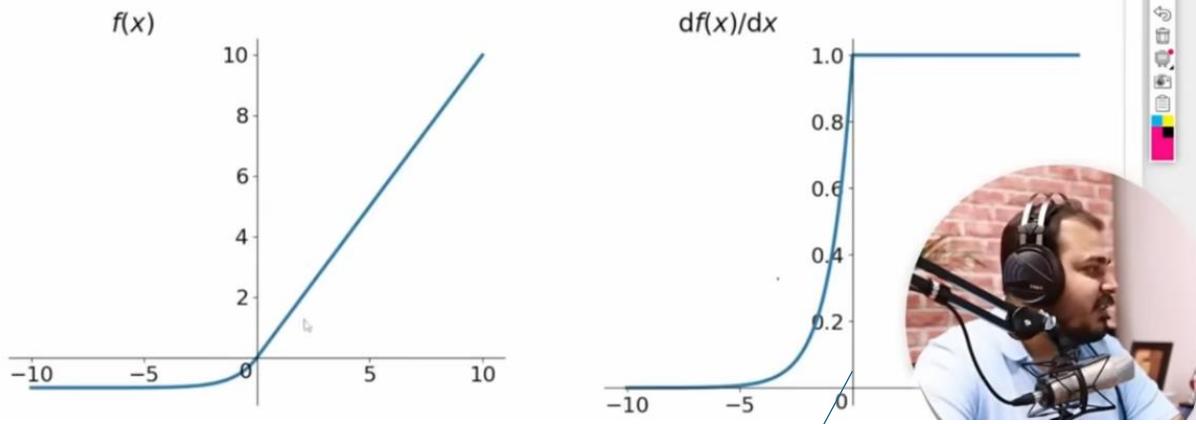


In order to solve the Dead ReLU Problem, people proposed to set the first half of ReLU 0.01x instead of 0. Another intuitive idea is a parameter-based method. Parametric ReLU : $f(x) = \max(\alpha x, x)$, which alpha can be learned from back propagation. In theory, Leaky ReLU has all the advantages of ReLU, plus there will be no problems with Dead ReLU, but in actual operation, it has not been fully proved that Leaky ReLU is always better than ReLU.

5. ELU (Exponential Linear Units) function

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

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



Values ranging from 0 to 1

ELU is also proposed to solve the problems of ReLU. Obviously, ELU has all the advantages of ReLU, and:

- No Dead ReLU issues
- The mean of the output is close to 0, zero-centered

One small problem is that it is slightly more computationally intensive. Similar to Leaky ReLU, although theoretically better than ReLU, good evidence in practice that ELU is always better than ReLU.

prelu

PReLU is also an improved version of ReLU. In the negative region, PReLU has a small slope, which can also avoid the problem of ReLU death. Compared to ELU, PReLU is a linear operation in the negative region. Although the slope is small, it does not tend to 0, which is a certain advantage.

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

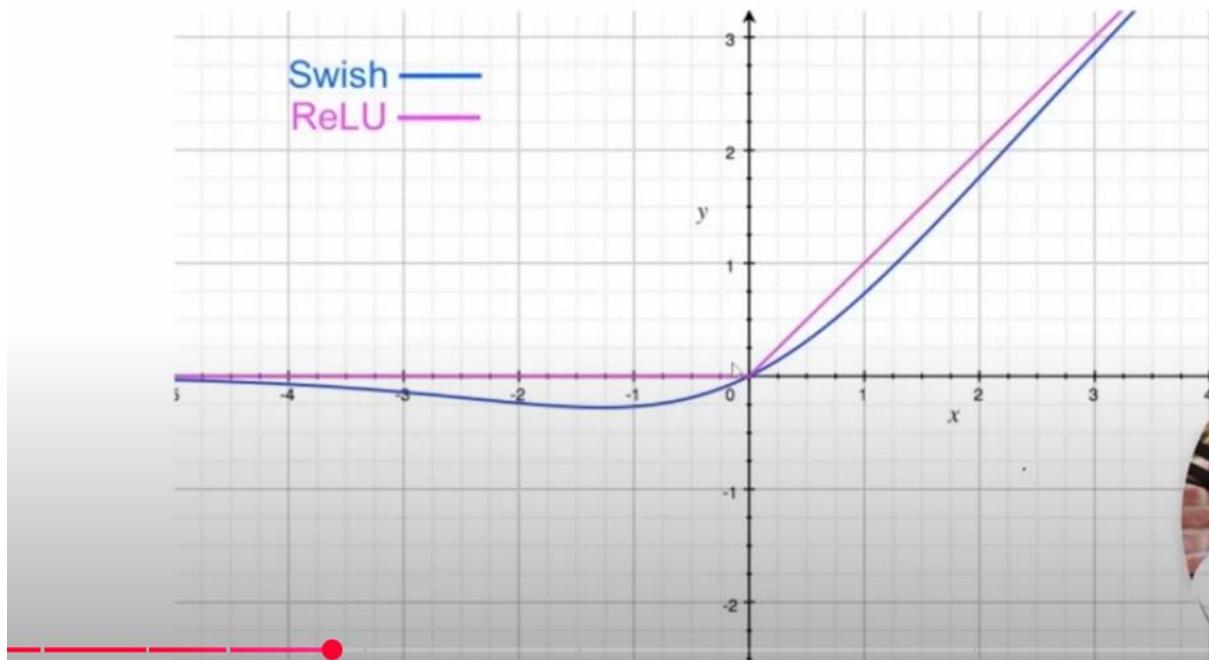
We look at the formula of PReLU. The parameter a is generally a number between 0 and 1, and it is generally relatively small, such as a few zeros. When $a = 0.01$, we call PReLU as Leaky ReLU, it is regarded as a special case PReLU it.

Above, y_i is any input on the i th channel and a_i is the negative slope which is a learnable parameter.

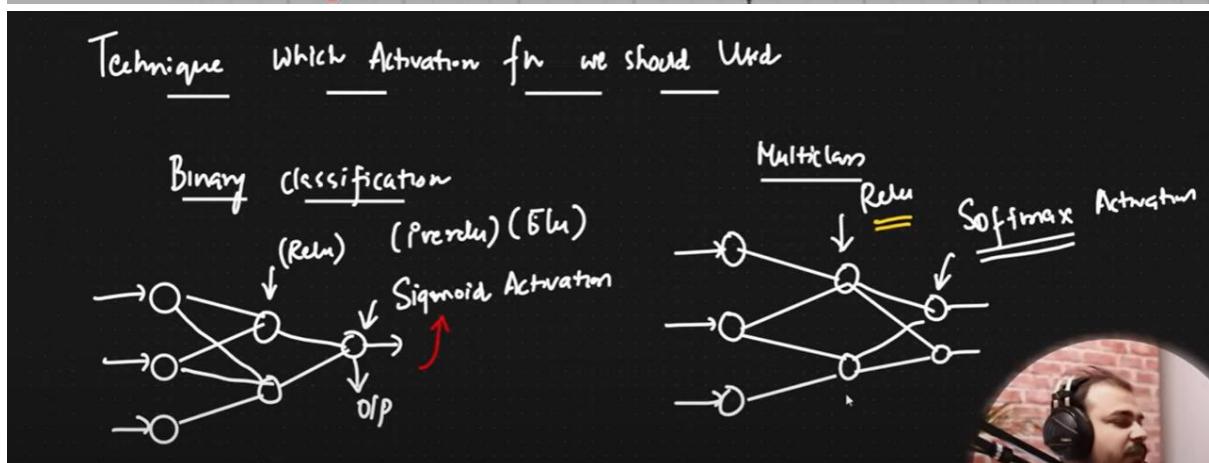
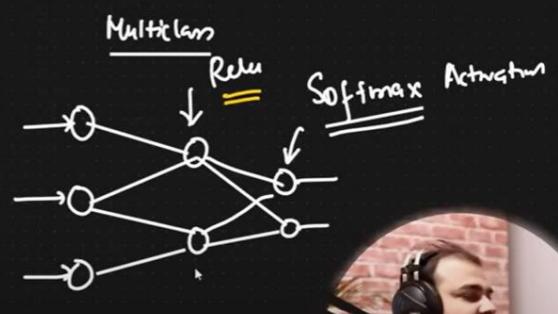
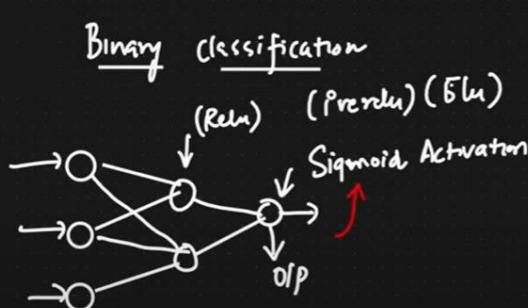
- if $a_i=0$, f becomes ReLU
- if $a_i>0$, f becomes leaky ReLU
- if a_i is a learnable parameter, f becomes PReLU

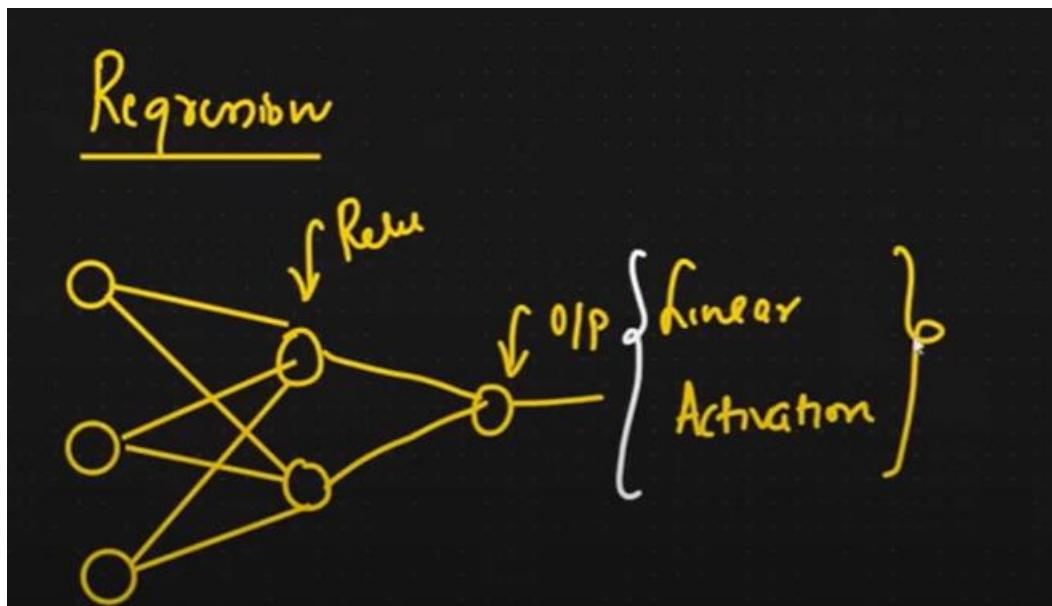


8. Swish (A Self-Gated) Function

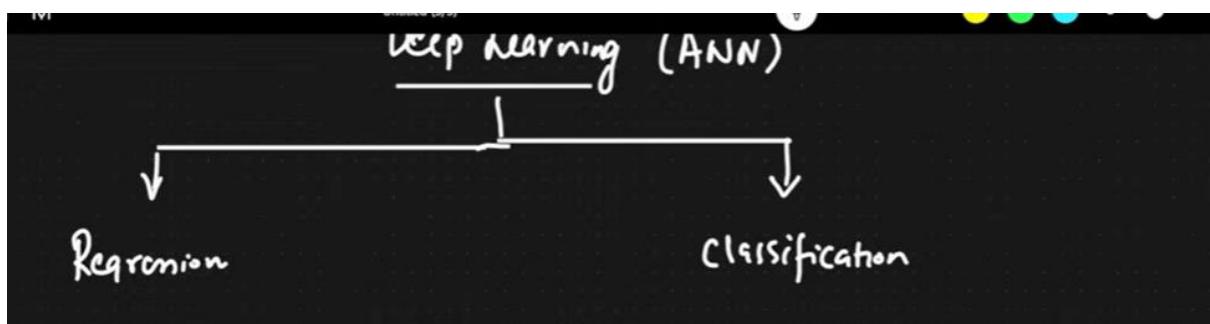


Technique Which Activation fn we should Use





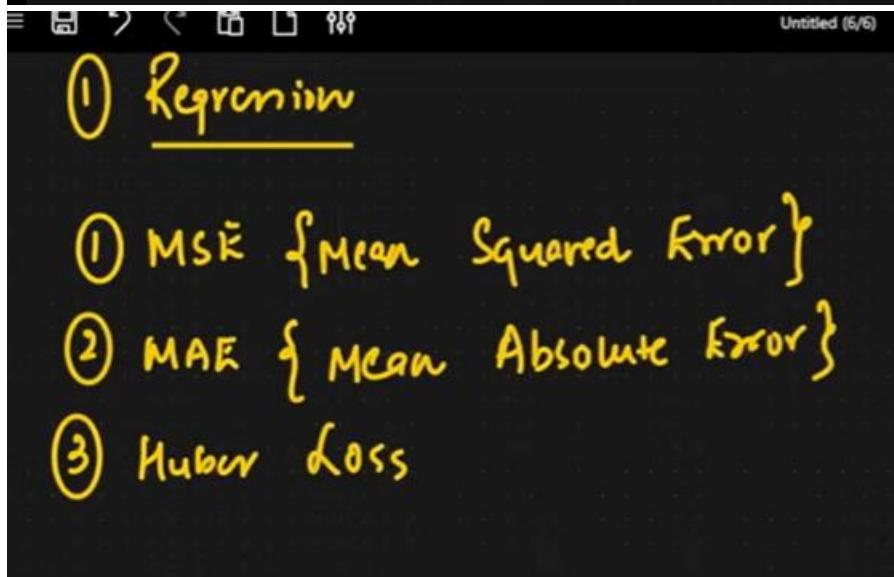
Loss function



Regression			Classification		
Exp	Degree	Salary	Play	Study	Pass/Fail
10	phd	-	10	2	Fail
-	-	-	4	3	Fail
-	-	-	5	5	Maybe
			2	7	Pass

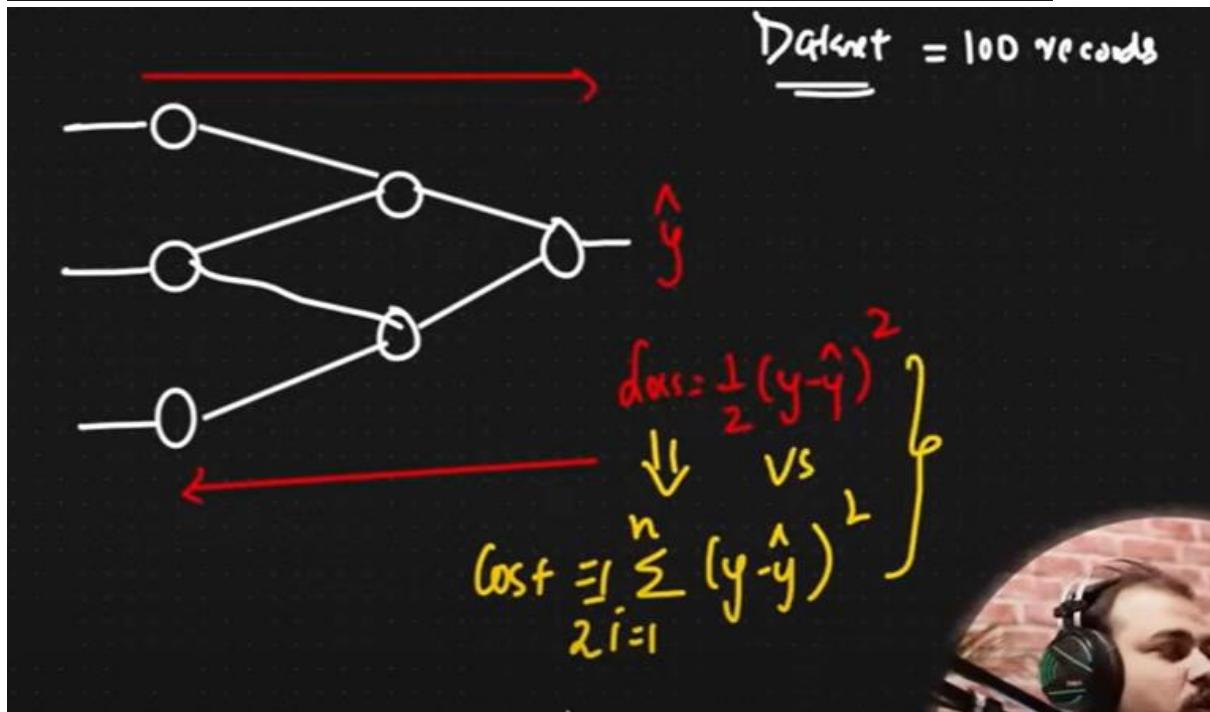
loss functions

Data set

Loss function AND

COST Function



Epochs – we provide batch of datasets for each propagation

① Mean Squared Error (MSE)

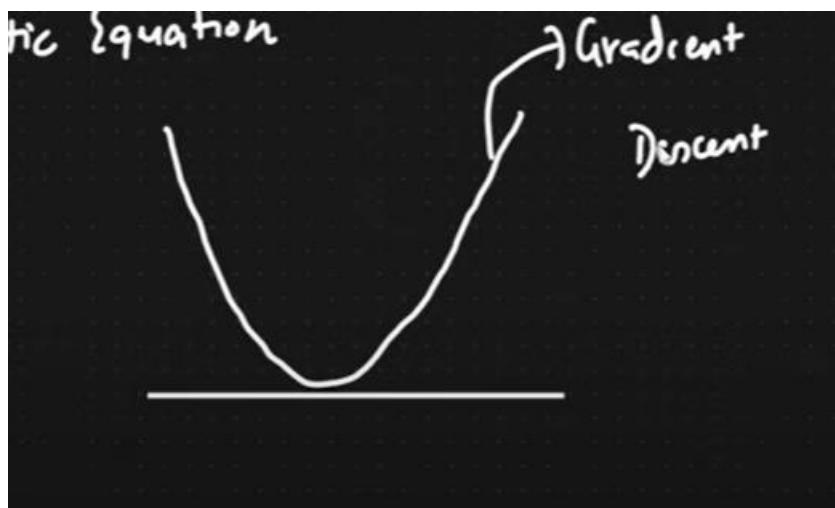
$$\text{Loss function} = \frac{1}{2} (y - \hat{y})^2 \quad \text{Cost function} = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

↓
Quadratic Equation

$$(a-b)^2 = a^2 - 2ab + b^2$$

$$\boxed{ax^2 + bx + c}$$

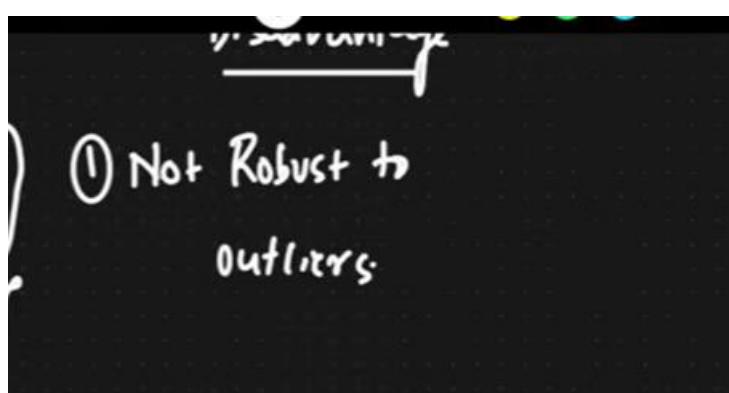
General equation of quadratic function



Quadratic curve it is giving some kind of gradient descent

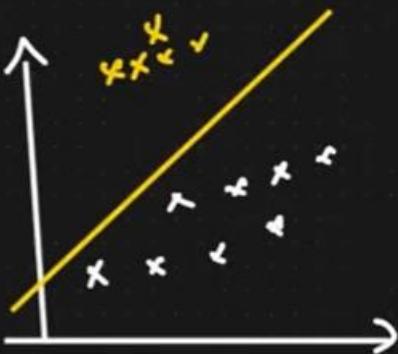
Advantages

- ① Differentiable
- ② It has only 1 local or Global Minima.
- ③ It converges faster



- ① Not Robust to outliers

Penalizing Error



$$\text{Loss function} = \frac{1}{2} (y - \hat{y})^2$$

↓ Error

Penalizing the error with respect to error by squaring them

② Mean Absolute Error

$$\text{Loss fn} = \frac{1}{2} |y - \hat{y}| \quad \text{Cost fn} = \frac{1}{2} \sum_{i=1}^n |y_i - \hat{y}_i|$$

① Robust to outliers



Sub gradient – time consuming divide the line part by part and compute the slope

③ Huber loss

① MSE

② MAE

outliers are not
present

Hyperparameter

$$\text{loss} = \begin{cases} \frac{1}{2} (\hat{y} - y)^2 & \text{if } |\hat{y} - y| \leq \delta \\ \delta |\hat{y} - y| - \frac{1}{2} \delta^2, \text{ otherwise} & \end{cases}$$

When outliers are present

③ Classification

Cross Entropy

→ Binary Cross Entropy → Binary Classification

→ Categorical Cross Entropy → Multiclass Classification.

① Binary Cross Entropy

$$\text{loss} = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y}) \Rightarrow \text{Logistic Regression}$$

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

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

Categorical Cross Entropy { Multiclass Classification Problem }

f_1	f_2	f_3	O/P	Good	Bad	Neutral
2	3	4	Good	1	0	0
5	6	7	Bad	0	1	0
8	9	10	Neutral	0	0	1

In this for dependent variable we convert dep variable into one hot encoder

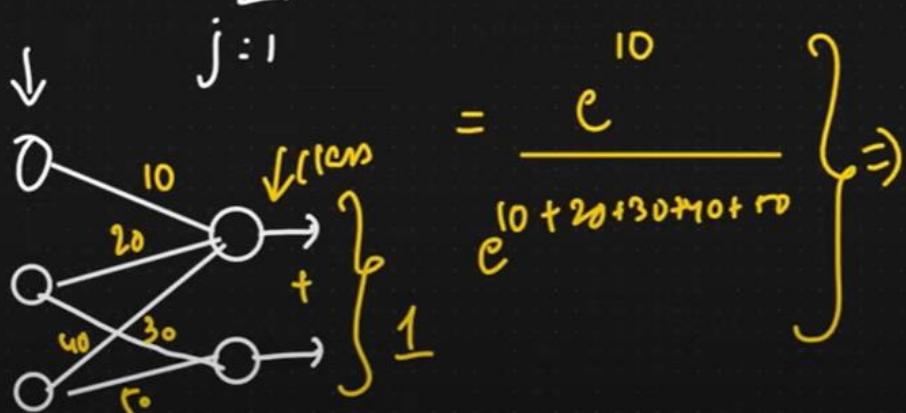
$$L(x_i, y_i) = - \sum_{j=1}^C y_{ij} * \ln(\hat{y}_{ij})$$

$$\mathbf{y}_i = [y_{i1}, y_{i2}, y_{i3}, \dots, y_{ic}]$$

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

\hat{y}_{ij} = Softmax Activation $\xrightarrow{\text{O/P Layer}}$

$$f(z) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \quad \left\{ \begin{array}{l} \Rightarrow \text{Softmax} \\ \text{Activation} \end{array} \right\} \quad 0.4, 0.15, 0.6$$



When we sum the prob value from softmax activation function we get the value probability is equal to 1

Conclusions

ReLU, Softmax \Rightarrow Multiclass }
ReLU, Sigmoid \Rightarrow Binary }

Linear Regression

ReLU, Linear Activa \rightarrow MSE, MAE, Huber Loss

Agenda

① Optimizers

- 1) Gradient Descent
- 2) SGD (Stochastic Gradient Descent)
- 3) Mini Batch SGD
- 4) SGD with Momentum
- 5) Adagrad

⑥ RMSProp

⑦ Adam Optimizer

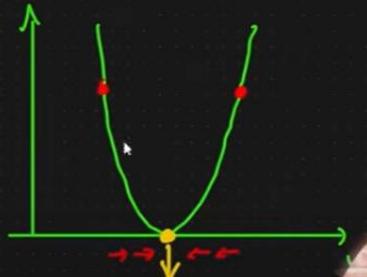
Batch, Epochs, Iterations

① GRADIENT DESCENT

Weight Updation Formula

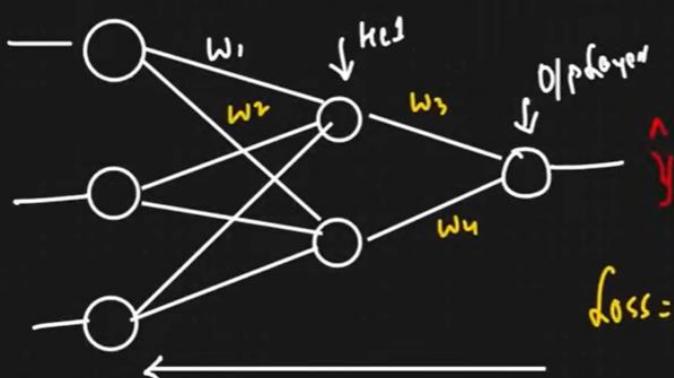
$$W_{\text{new}} = W_{\text{old}} - \eta \left[\frac{\partial L}{\partial W_{\text{old}}} \right]$$

Loss (Cost)



Global Minima.

↓ Input



MSE

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

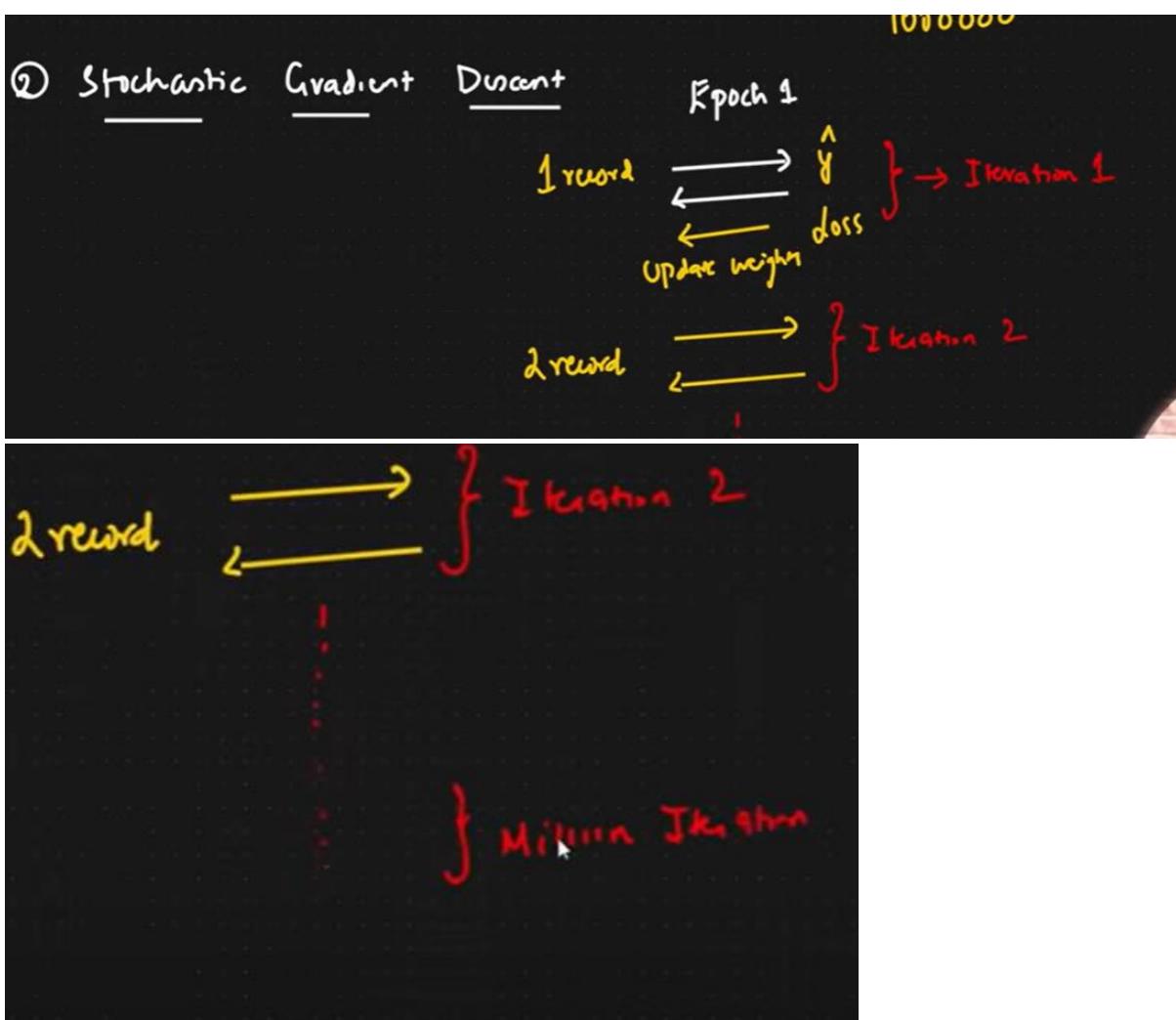
Optimizers

Cost function

$$\Rightarrow \frac{1}{2n} \sum_{i=1}^n (y - \hat{y})^2 \quad \left. \right\} \quad \left. \right\} 1000000 \quad \left. \right\} 1 \text{ Epoch}$$

disadvantage

① Resource intensive {huge RAM}



② Stochastic Gradient Descent

① RAM \downarrow

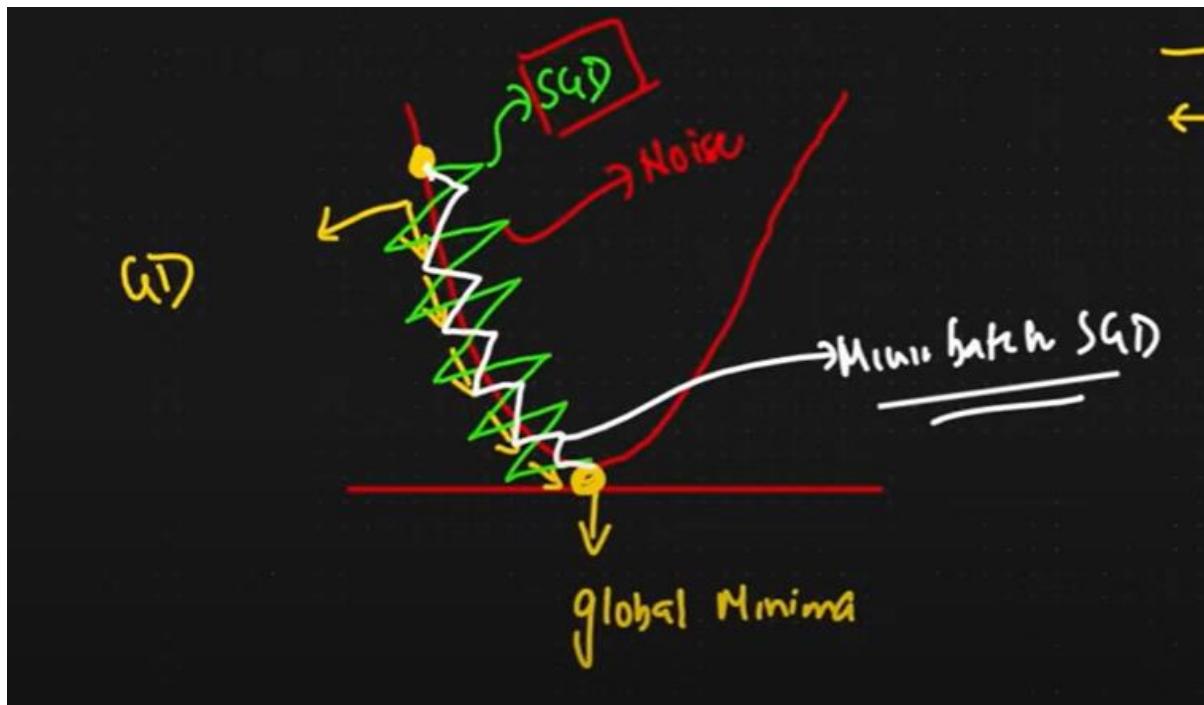
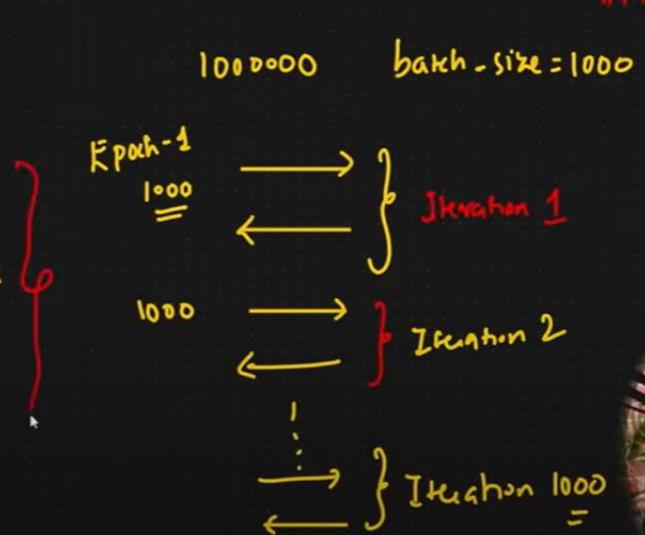
Disadvantage

(F) Convergence will }
be very slow }

⑥ Time complexity will also be high

③ Mini batch SGD

- ① Resource Intensive
- ② Convergence will be better
- ③ Time complexity will improve



Sgd has highest noise

Gd has least noise

But mini batch has less noise

How do we remove the noise ?

We use concept called momentum

④ SGD With Momentum

{ Exponential Weighted Average }



Time Series



ARIMA, ARMA,

$$\begin{aligned} w_{\text{new}} &= w_{\text{old}} - \eta \left[\frac{\partial L}{\partial w_{\text{old}}} \right] \\ b_{\text{new}} &= b_{\text{old}} - \eta \left[\frac{\partial L}{\partial b_{\text{old}}} \right] \end{aligned}$$

$$\omega_t = \omega_{t-1} - \eta \frac{\partial h}{\partial \omega_{t-1}}$$

$$t_1 \quad t_2 \quad t_3 \quad t_4 \quad \cdots t_n$$

$\underbrace{a_1 \quad a_2}_{\alpha} \quad a_3 \quad a_4 \quad \cdots a_n$

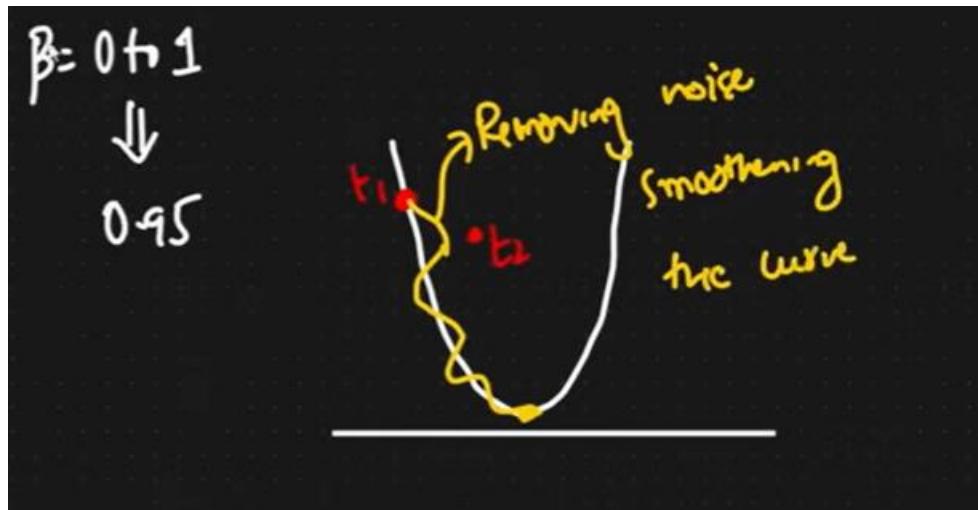
$$V_{t_1} = a_1$$

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

$$\beta = 0 \text{ or } 1$$

$$= (0.95) \nearrow v_{ts} + (0.05) \nearrow a_2$$

Beta is a hyper parameter which value of time stamp should focus on



Beta hyperparameters helps you to smoothen (meaning removing the noise)

$$V_{t_3} = \beta \times V_{t_2} + (1 - \beta) \times a_3$$

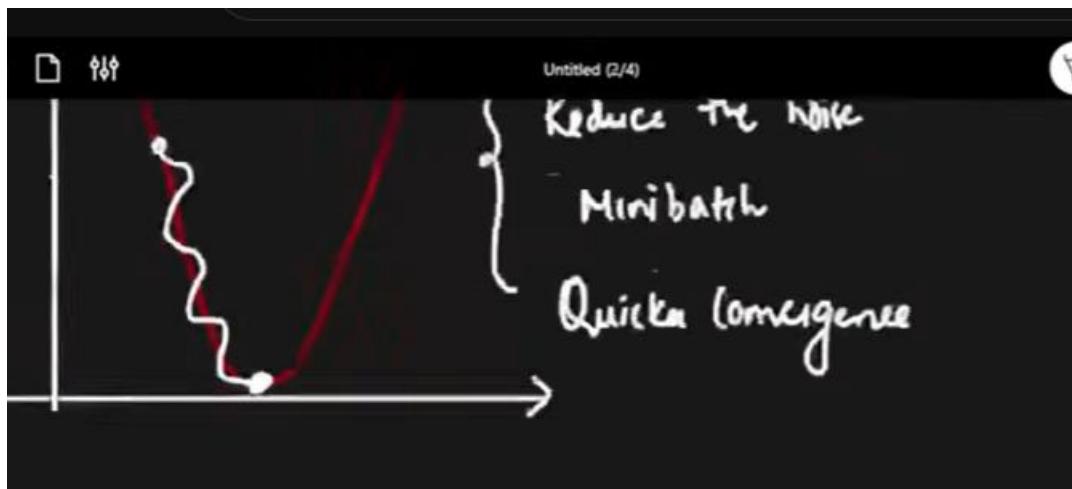
Where do I apply exponentially weighted average – we apply in derivative of loss function

Exponential Weighted Avg

$$w_t = w_{t-1} - \eta \nabla_{dw}$$

$$\boxed{V_{dw} = \beta \times V_{dw_{t-1}} + (1 - \beta) \times \frac{\partial L}{\partial w_{t-1}}}$$

We have solved



① Gradient Descent

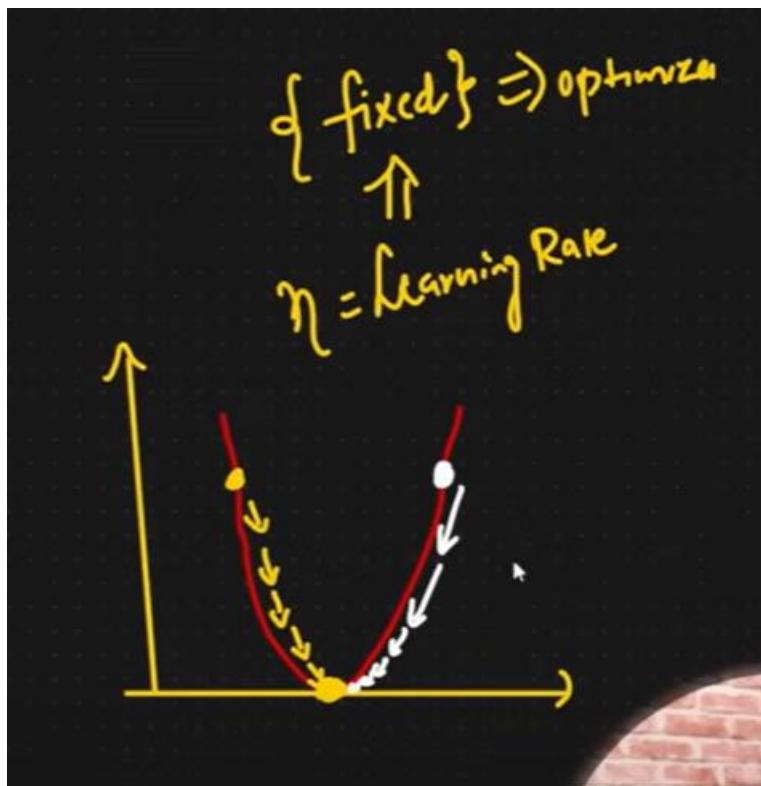
② SGD

③ Mini batch SGD

④ SGD with Momentum

⑤ Adagrad

⑥ Adagrad → Adaptive Gradient Descent



$\eta = \text{fixed} \Rightarrow \text{adaptive}$

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

\downarrow

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

Earlier learning rate is fixed whereas in adagrad learning rate is adaptive meaning it may range

$$\eta' = \frac{\eta}{\sqrt{d_f + \epsilon}}$$

ϵ

$\sqrt{d_f + \epsilon} \rightarrow$ Small number

Just to avoid 0 in denominator we are adding small number epsilon

We need to decrease it as we reach global minima

$$d_f = \sum_{i=1}^t \left(\frac{\partial h}{\partial w_i} \right)^2 \uparrow \uparrow \uparrow$$

$t=1 \quad t=2 \quad t=3$

$\eta = 0.01 \quad \eta = 0.05 \quad \eta = 0.002$

η = fixed \Rightarrow adaptive \Rightarrow learning Rate \Rightarrow Decreasing \rightarrow Global Minima



$$\frac{\partial h}{\partial w_{t-1}}$$

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

~~more~~
~~number~~

$t=1 \quad t=2 \quad t=3$

$\eta = 0.01 \quad \eta = 0.005 \quad \eta = 0.002$

What if we add huge number since we add derivatives learning rate will be changed then in order to overcome we will go for adadelta and rms prop

The learning rate will be smaller and weight will not get updated it will be same as old weight

④ Adadelta and RMSprop

$$\eta' = \frac{\eta}{\sqrt{S_{dw} + \epsilon}}$$

Exponential weighted average

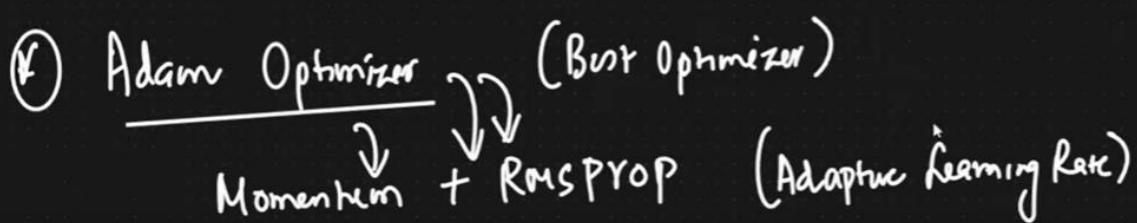
$$S_{dw} = 0$$

$$S_{dw_t} = \beta S_{dw_{t-1}} + (1-\beta) \left(\frac{\partial L}{\partial w_{t-1}} \right)^2$$

$$\beta = 0.95$$

$$S_{dw_t} = (0.95) S_{dw_{t-1}} + (0.05) \left(\frac{\partial L}{\partial w_{t-1}} \right)^2$$

With help of this beta parameter we are controlling the increase there will be slow decreasing of learning rate



$$V_{dw} = 0 \quad S_{dw} = 0 \quad S_{db} = 0$$

$$w_t = w_{t-1} - \eta \begin{bmatrix} V_{dw} \end{bmatrix}$$

$$b_t = b_{t-1} - \eta \begin{bmatrix} V_{db} \end{bmatrix}$$

$$\eta' = \frac{n}{\sqrt{Sd_w + e}}$$

$$Vd\omega_t = \beta \times Vd\omega_{t-1} + (1-\beta) \frac{\partial L}{\partial \omega_{t-1}}$$

It achieves

(a) $\left\{ \begin{array}{l} \text{① Smoothening} \\ \text{② Learning Rate} \\ \text{Adaptive} \end{array} \right\}$

Day 4 - Deep Learning

- ① ANN Practical Implementation ✓
- ② Early Stopping ✓
- ③ Black Box Models Vs White Box Models
- ④ CNN Introduction

$$V_{d\omega_t} = \beta \times V_{d\omega_{t-1}} + (1-\beta) \frac{\partial L}{\partial \omega_t}$$

!pip install tensorflow-gpu

✓ 3s import tensorflow as tf
 print(tf.__version__)
 I
 2.8.0

[3] ## import some basics libraries
 import numpy as np
 import matplotlib.pyplot as plt
 import pandas as pd

dataset=pd.read_csv('Churn_Modelling.csv')
dataset.head()

Divide the dataset into independent and dependent features
X=dataset.iloc[:,3:13]
y=dataset.iloc[:,13]

```
## Feature Engineering  
pd.get_dummies(X['Geography'])
```

France Germany Spain

	France	Germany	Spain
0	1	0	0
1	0	0	1
2	1	0	0
3	1	0	0
4	0	0	1

```
## Feature Engineering  
pd.get_dummies(X['Geography'], drop_first=True)
```

```
[10] ## Feature Engineering  
geography=pd.get_dummies(X['Geography'], drop_first=True)  
gender=pd.get_dummies(X['Gender'], drop_first=True)
```

```
[12] ## concatenate these variables with dataframe  
X=X.drop(['Geography', 'Gender'], axis=1)
```

```
[15] X=pd.concat([X,geography,gender],axis=1)
```

Interview

① For which all algorithms
feature scaling is required?

ANN

{ Feature Scaling }

{ Feature Scaling }

ANN? ✓

LR? ✓

fog R? ✓

XGBoost ✓

Decision? ✗

KNN ✓

RF ? ✗

KMeans ✓

ANN

{ Feature Scaling }

① Distance based

Scaling XGBoost

② Gradient Descent ✓

KNN ✓

KMeans ✓

```

[16] #Splitting the dataset into Training set and Test Set
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)

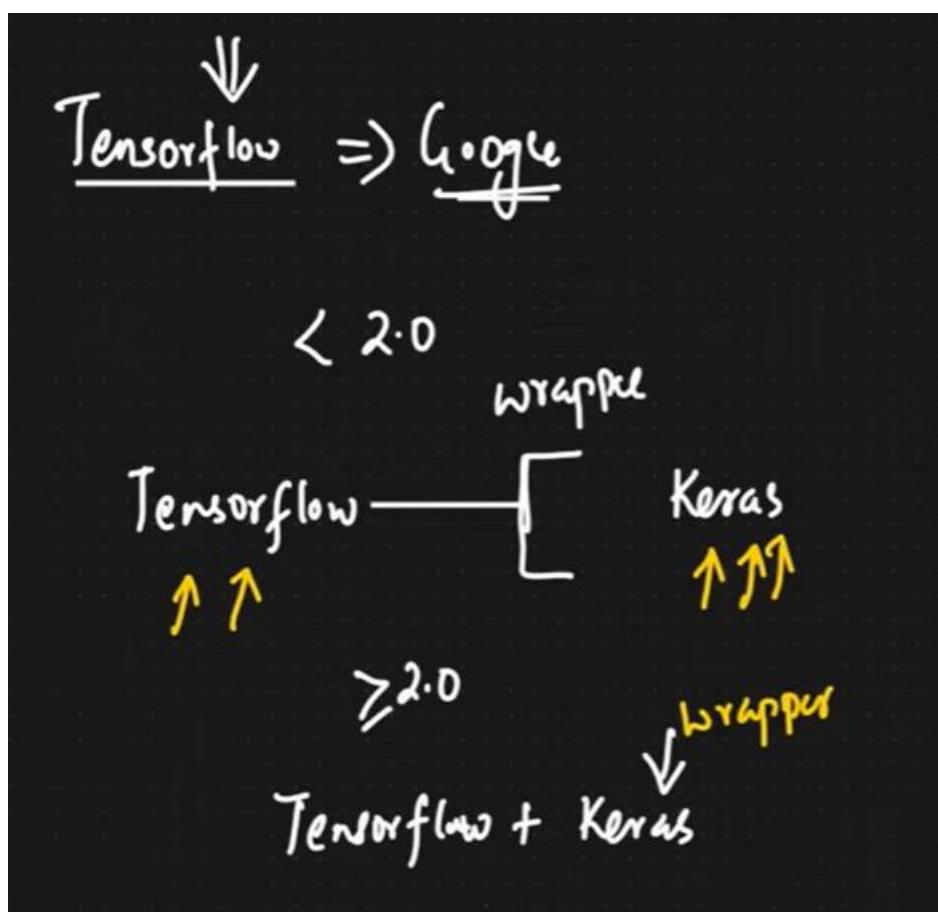
▶ #feature Scaling

from sklearn.preprocessing import StandardScaler
sc =StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

↳ /usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: Use
   f"X has feature names, but {self.__class__.__name__} was fitt

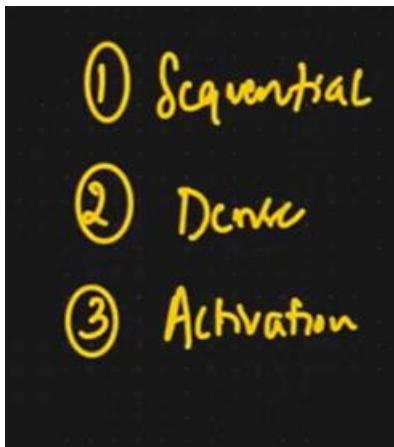
```

Why we apply fit_transform on training dataset and not on test dataset - just to avoid data leakage

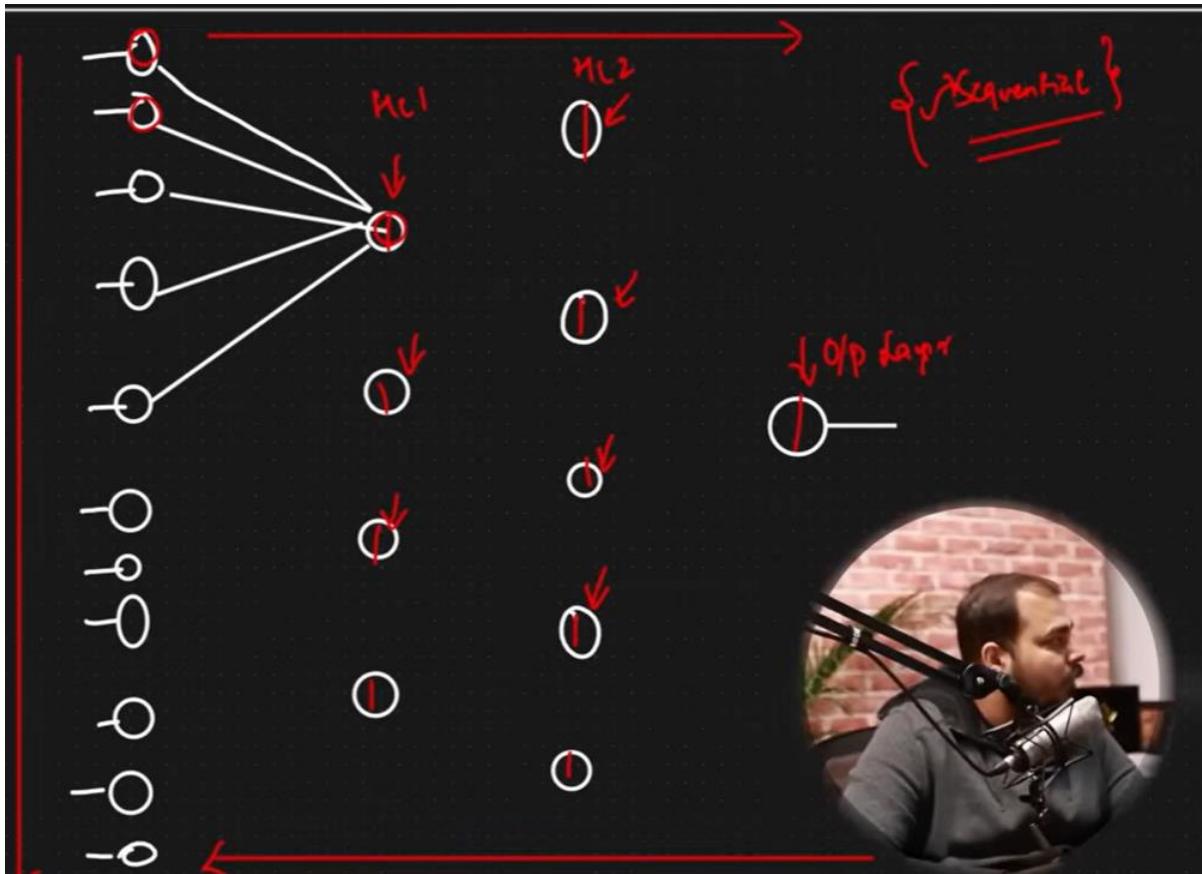


Keras has been integrated with tensorflow 2.0

```
## Part 2 Now lets create the ANN
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LeakyReLU,PReLU,ELU,ReLU
from tensorflow.keras.layers import Dropout
```



With help of dense layer we can create output layer, hidden layer



To avoid overfitting we can introduce drop out layer in ann drop out ratio =0.30 means 30 % of neurons will be deactivated

Remaining will be randomly selected

```
[23] ## Part 2 Now lets create the ANN
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.layers import LeakyReLU,PReLU,ELU,ReLU
    from tensorflow.keras.layers import Dropout

[24] ### Lets initialize the ANN
    classifier=Sequential()

    ## Adding the input Layer
    classifier.add(Dense(units=11,activation='relu'))

[26] # adding the first hidden layer
    classifier.add(Dense(units=7,activation='relu'))

    ##adding the second hidden layer
    classifier.add(Dense(units=6,activation='relu'))

[28] ## Adding the output layer
    classifier.add(Dense(1,activation='sigmoid'))

    classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Adam uses default learning rate is 0.01

```
import tensorflow
opt=tensorflow.keras.optimizers.Adam(learning_rate=0.01)

<keras.optimizer_v2.adam.Adam at 0x7ff2d0187e50>

32] classifier.compile(optimizer=opt,loss='binary_crossentropy',metrics=['accuracy'])

model_history=classifier.fit(X_train,y_train,validation_split=0.33,batch_size=10,epochs=1000)

... Epoch 1/1000
535/536 [=====>.] - ETA: 0s - loss: 0.4262 - accuracy: 0.8114
```

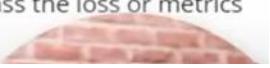
What will be my number of epochs for that we will use early stopping

Stop training when a monitored metric has stopped improving.



Assuming the goal of a training is to minimize the loss. With this, the metric to be monitored would be '`loss`', and mode would be '`min`'. A `model.fit()` training loop will check at end of every epoch whether the loss is no longer decreasing, considering the `min_delta` and `patience` if applicable. Once it's found no longer decreasing, `model.stop_training` is marked True and the training terminates.

The quantity to be monitored needs to be available in `logs` dict. To make it so, pass the loss or metrics at `model.compile()`.



```
## Early Stopping
import tensorflow as tf
early_stopping=tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0.0001,
    patience=20,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
model_history=classifier.fit(X_train,y_train,validation_split=0.33,batch_size=10,epochs=1000,callbacks=early
```

Epoch 1/1000
347/536 [=====>.....] - ETA: 0s - loss: 0.4509 - accuracy: 0.7939

```
36] model_history.history.keys()  
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
37] # summarize history for accuracy  
plt.plot(model_history.history['accuracy'])  
plt.plot(model_history.history['val_accuracy'])  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train', 'test'], loc='upper left')  
plt.show()
```

▶ # summarize history for accuracy
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

```
[39] # Part 3 - Making the predictions and evaluating the model
```

```
# Predicting the Test set results  
y_pred = classifier.predict(X_test)  
y_pred = (y_pred >= 0.5)
```

✓ ⏎ ## make the confusion matrix
↳ from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm

[↳ array([[1527, 68],
 [215, 190]])

```
[42] ## Calculate the accuracy
    from sklearn.metrics import accuracy_score
    score=accuracy_score(y_pred,y_test)

▶ score
└ 0.8585

▶ ##get the weights
    classifier.get_weights()
└ -1.072681 , -0.9926772 , -1.2926713 , -0.16671637, 0.598037 , 

▶ # adding the first hidden layer
    classifier.add(Dense(units=7,activation='relu'))
    classifier.add(Dropout(0.2))

▶ ##adding the second hidden layer
    classifier.add(Dense(units=6,activation='relu'))
    classifier.add(Dropout(0.3))
```

Black Box Model VS White Box Model ANN → Black Box Model

Random Forest → Black Box Model

Decision Tree → White Box Model

Xgboost → Black Box Model

Linear Regres → White Box Model

CNN

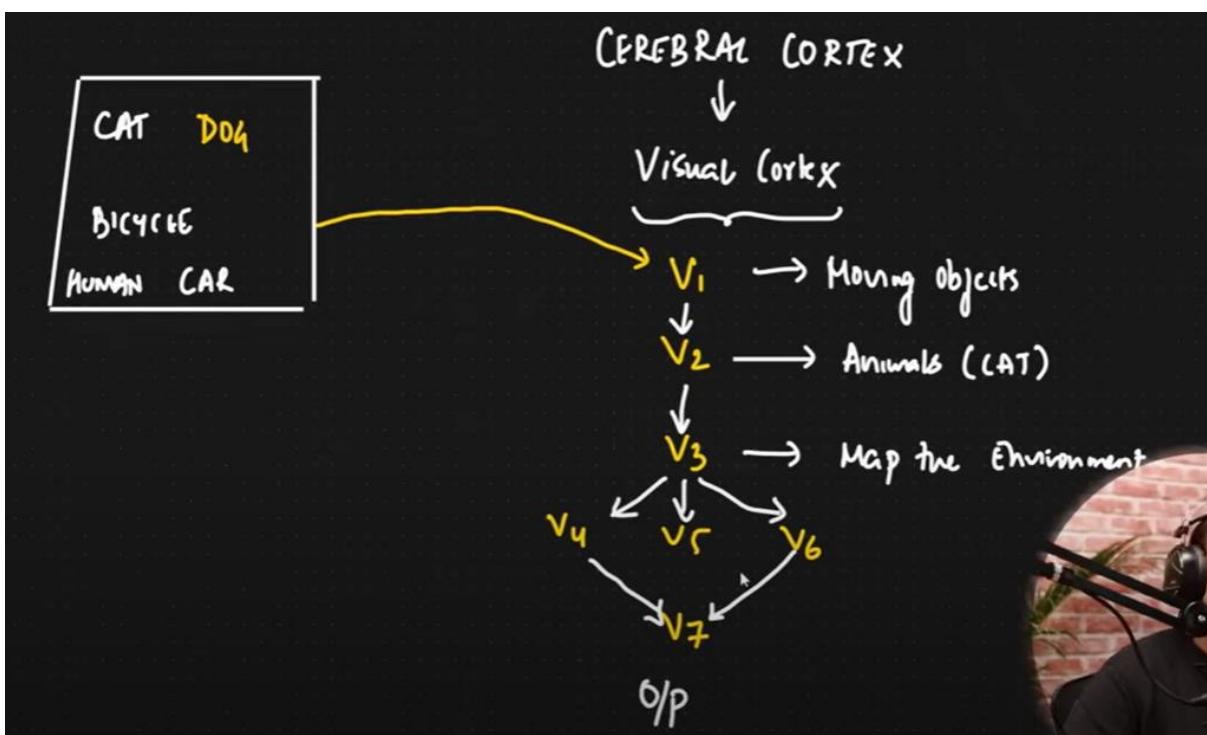
ANN → Theoretical, Practical

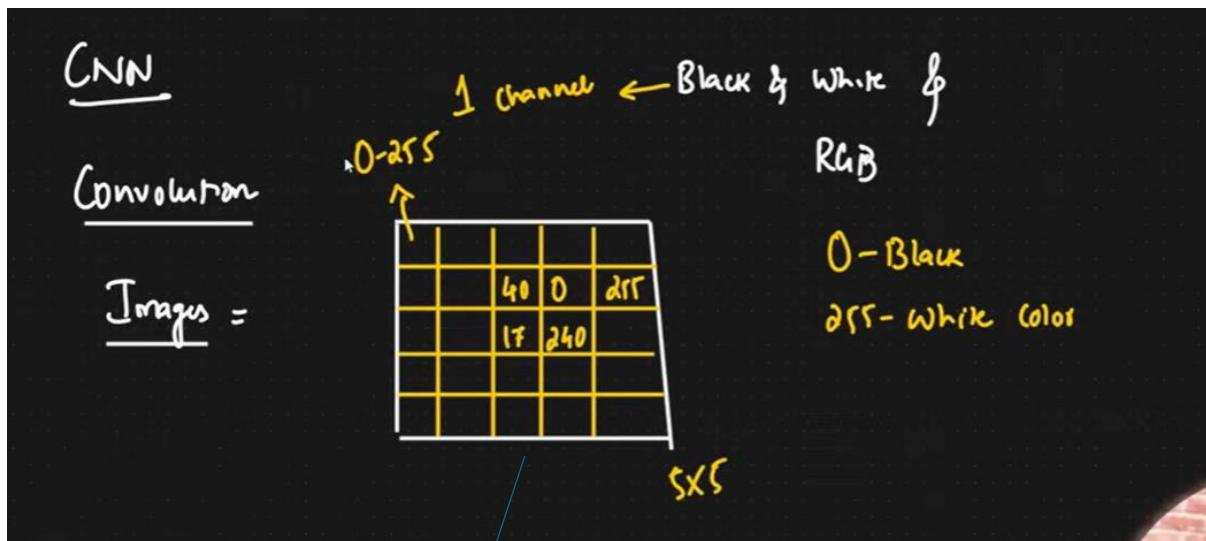
CNN → Convolution Neural Net ⇒ Image, Video frame

Agenda

- ① CNN VS Human Brain
- ② Convolution Operation
 - Convolution
 - Padding
 - Stride
 - Filters (Kernels)

- ③ Max Pooling
- ④ Flattening
- ⑤ Practical Implementation

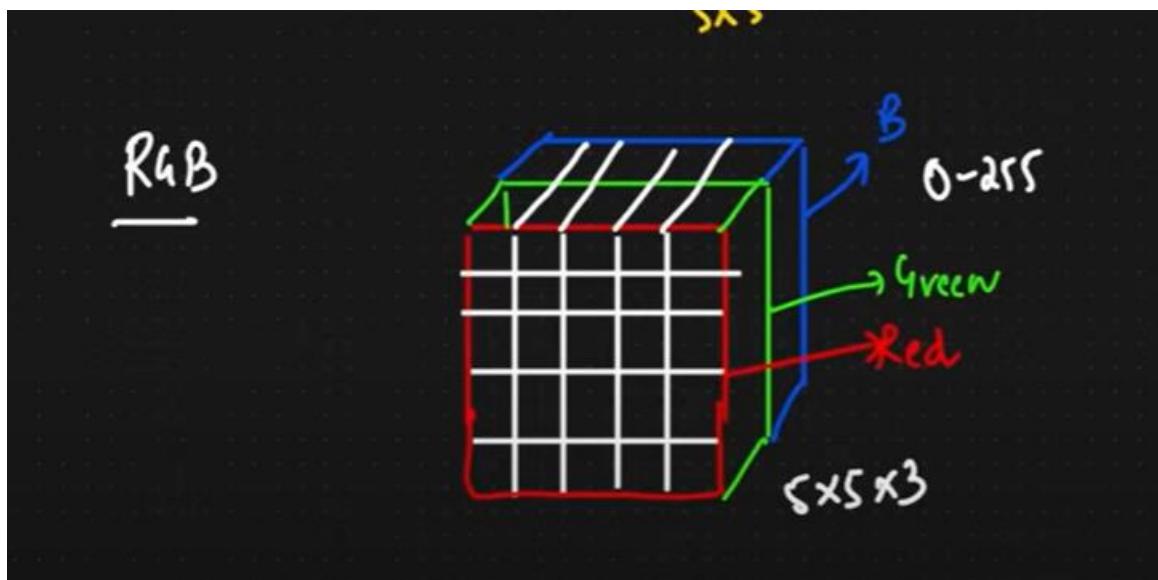


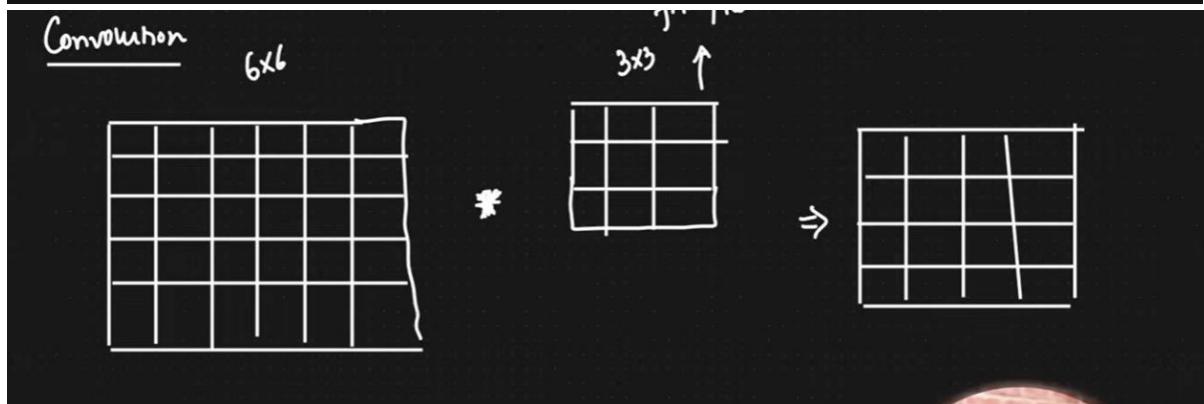
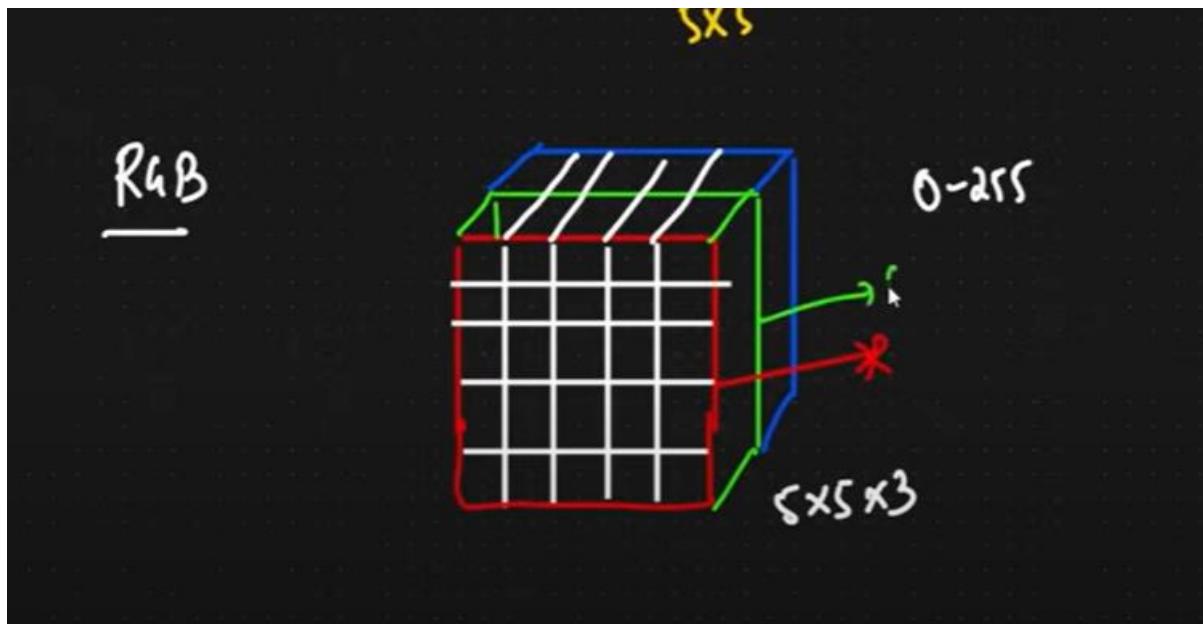


Each image it has a pixel range from 0 to 255

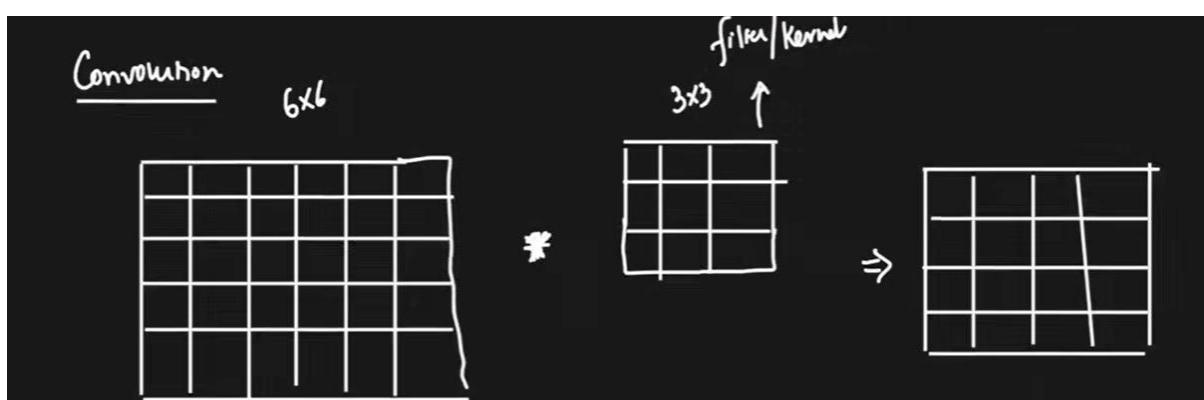
5 * 5 pixel

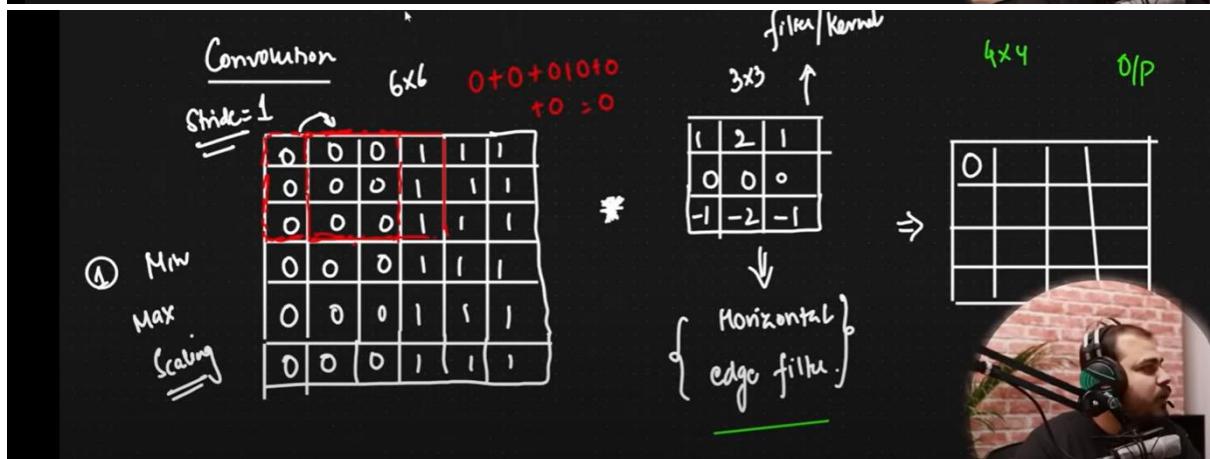
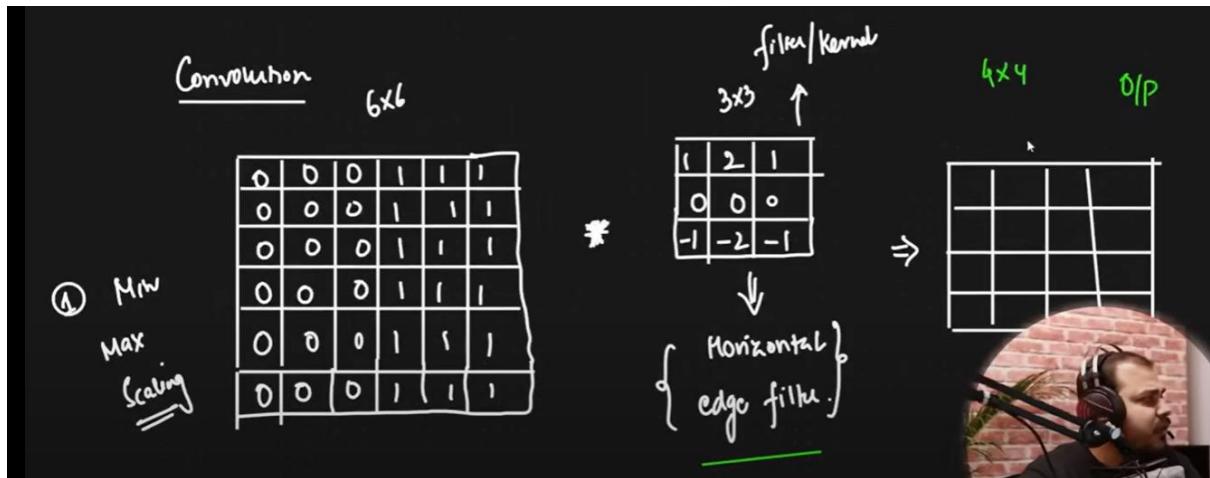
RGB every color image has been setup in RGB format (red blue and green)



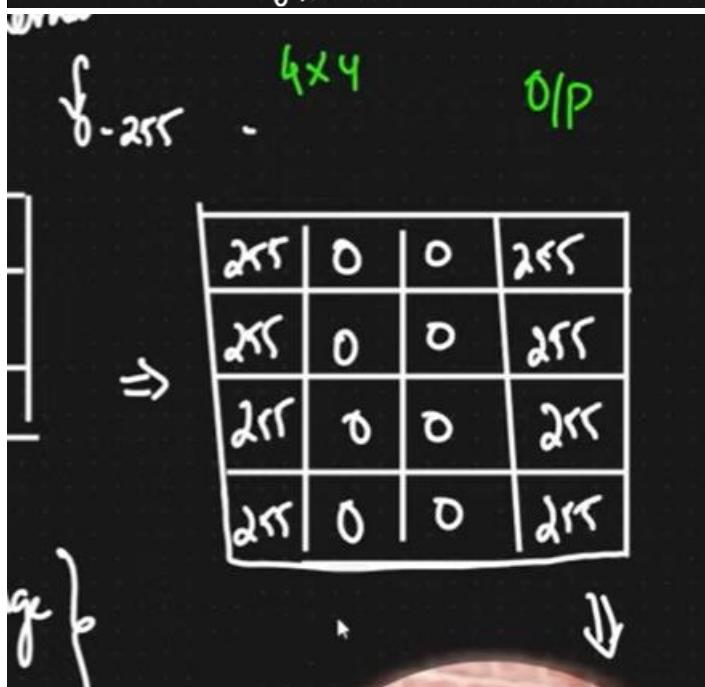
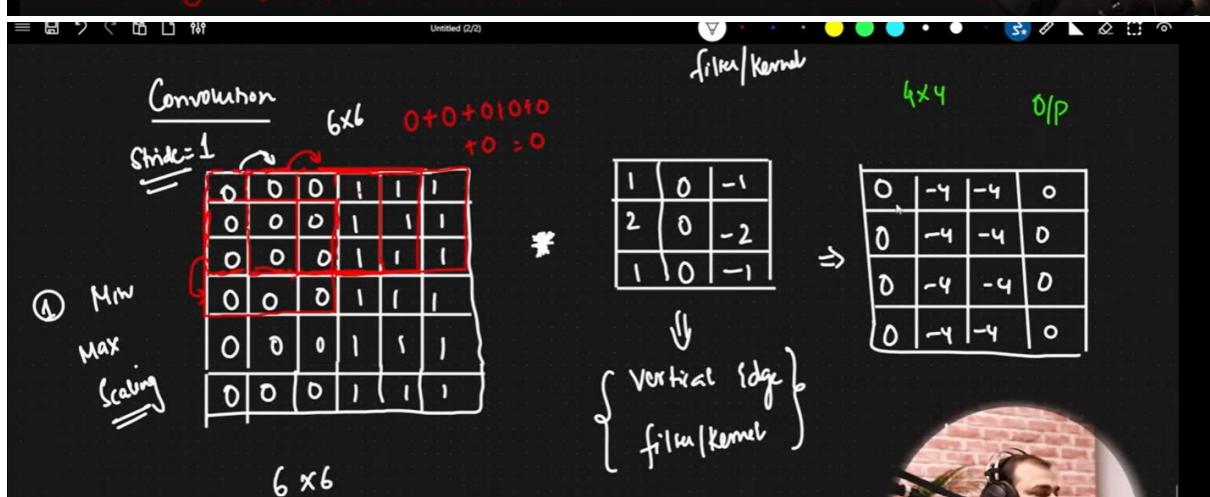
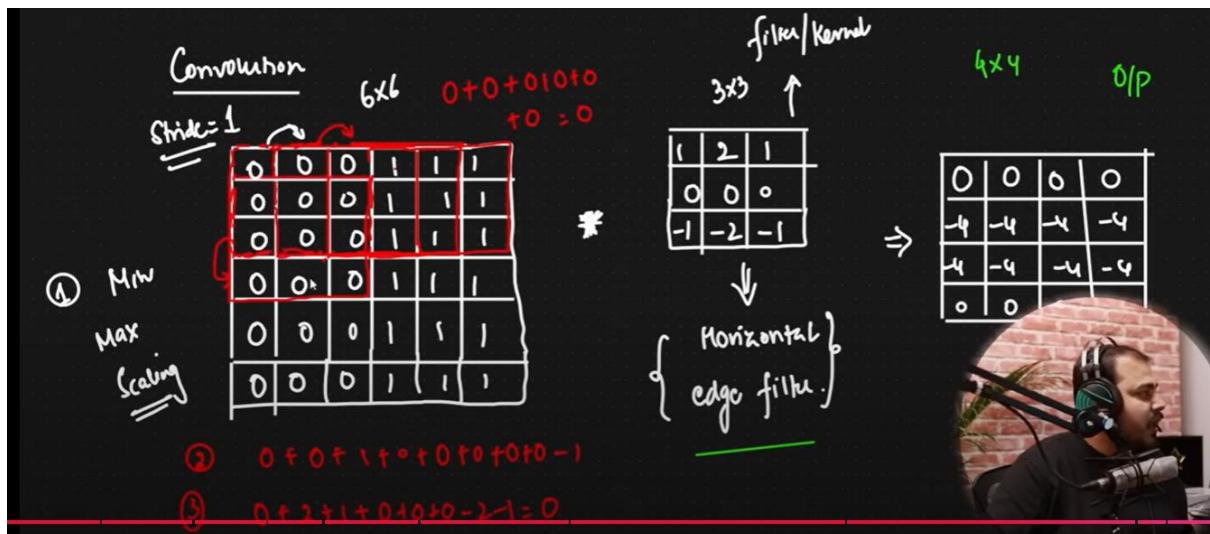


In convolution we bring the values 0 to 1 by min - max scalar





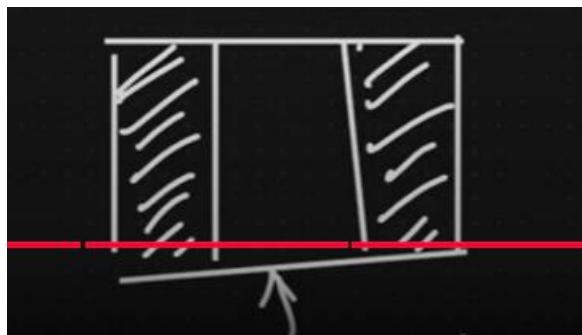
When we move the filter on top of matrix then we called it as stride say when move towards left side with one step then stride will equal to 1 , if move towards 2 steps then stride will be 2



If we inverse the values means -4 becomes 0 and 0 will become 255 since 0 being the highest and -4 being the lowest

255 is white color 0 is black color

If we apply vertical filter then we should vertical edges this process is called as convolution operation we are trying to extract information from image



Calculation for getting output matrix

$$\begin{array}{cccc} n=6 & f=3 & O/p & \xrightarrow{\text{vertical 1dg}} \\ \hline & & & \\ & 6 \times 6 & = & 3 \times 3 \\ & & & = \\ & & & n-f+1 = 6-3+1 = 3+1=4 \\ & & & = \\ & & & O/p = 4 \times 4 \\ & & & = \end{array}$$

If image size decreases we lose some kind of information

Padding

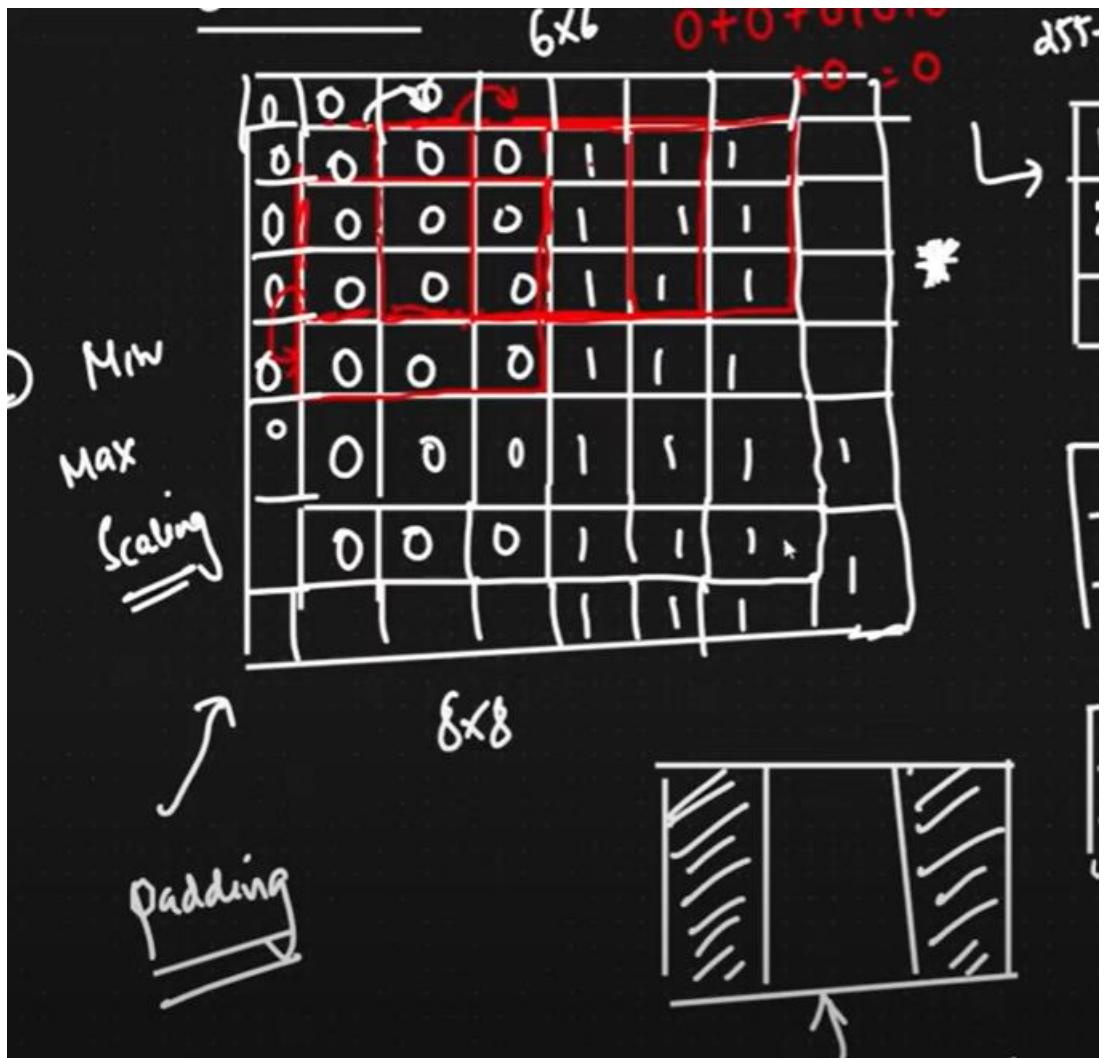
Concept of adding a layer is called as padding

Protecting image (size of a image) by adding a layer

What values we can fill there is called zero-padding we add simply 0

Whatever nearest value we can take that one also

To prevent information loss of a image we are adding a layer called padding



After applying padding

Then output will become like

\Rightarrow

255	0	0	1	255
255	0	0	255	
255	0	0	255	
255	0	0	255	

6×6

After padding

$$= 8 - 3 + 1 = 6$$

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

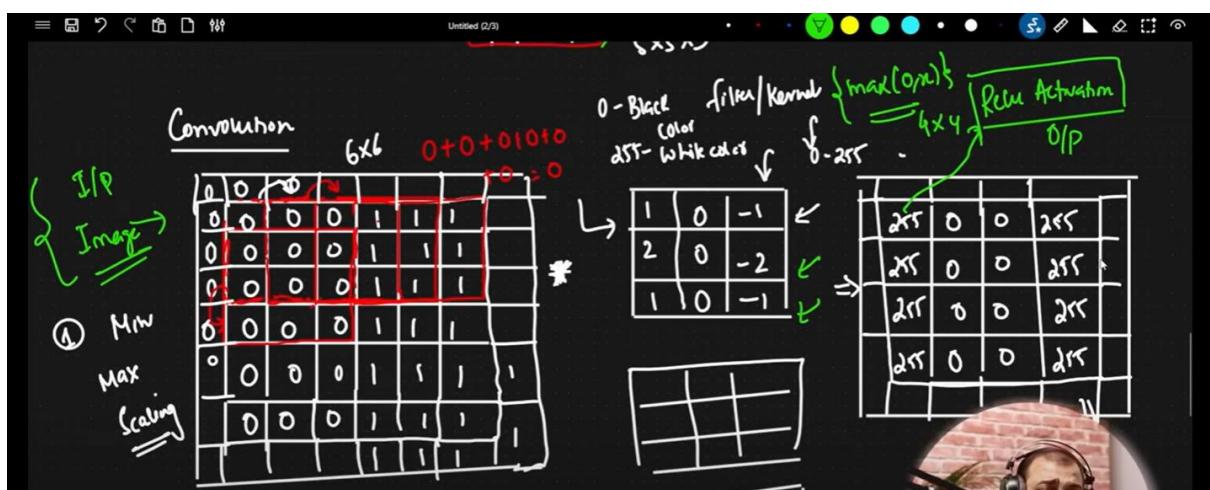
$$n + 2p - f + 1 = 6 + 2 - 3 + 1$$

$$= 6$$

Add padding

$$\begin{array}{l}
 h=6 \quad S=1 \\
 p=1 \quad S=2 \\
 f=3 \quad = \\
 \end{array}
 \quad
 \frac{h + 2p - f + 1}{S} = \frac{6 + 2 - 3 + 1}{2} = 6$$

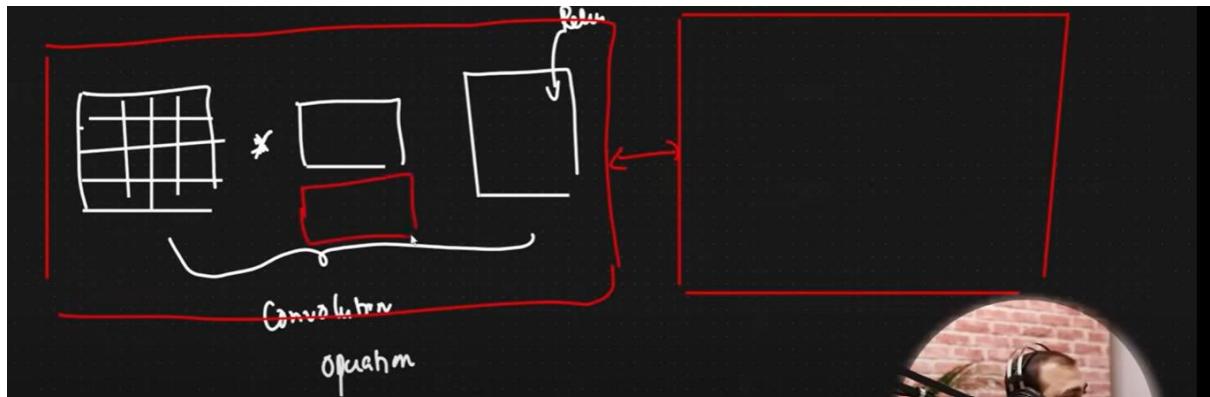
S is stride



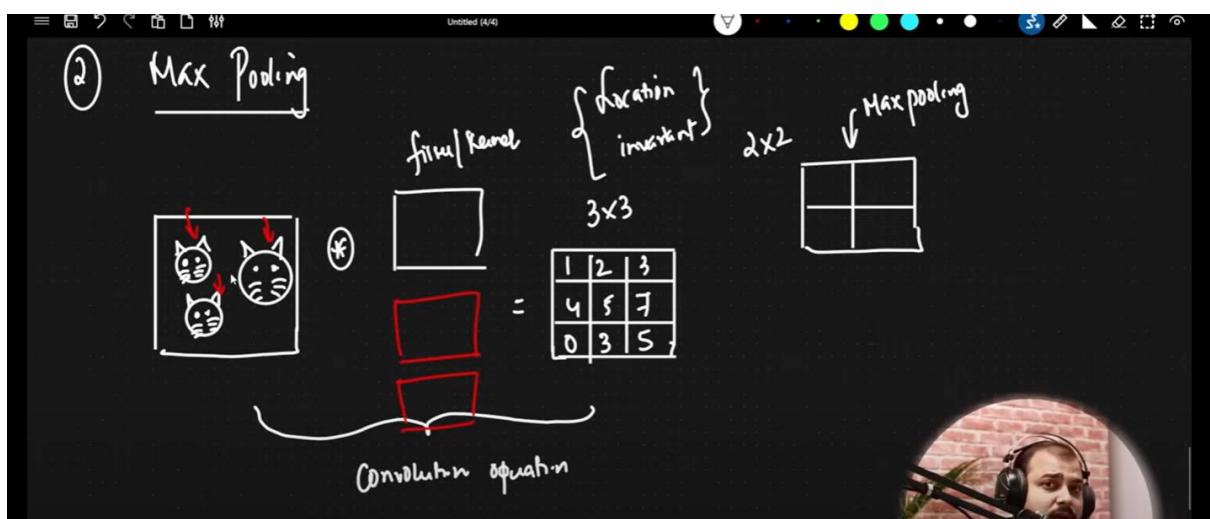
We apply activation function each and every cell to find out derivative to update the filters

By means of back propagation

Processing the image filtering it and getting output and applying output on top of relu activation function is called as convolution operation



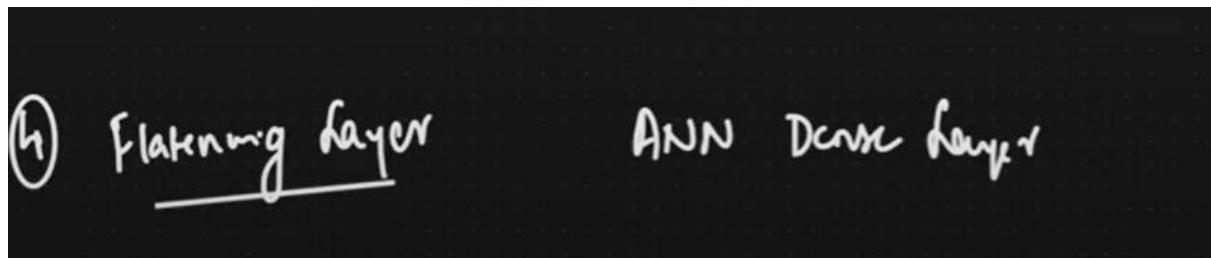
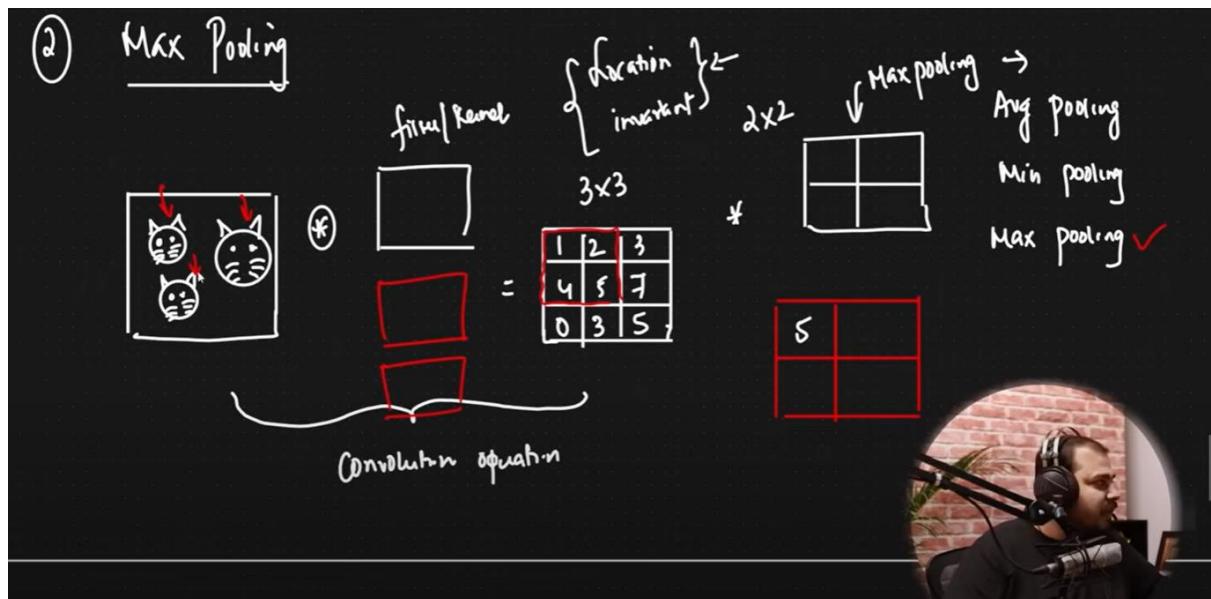
Max pooling



Location invariant – says as we pass on or move on with more convolution layers we extract more images capture max information for that we go for max pooling

Avg Pooling
Min pooling
Max pooling ✓

We place 2×2 max pooling matrix on top of 3×3 filters then we can get another matrix in red



Output of max pooling will flatten basically like a input in ann (will become a dense layer)

