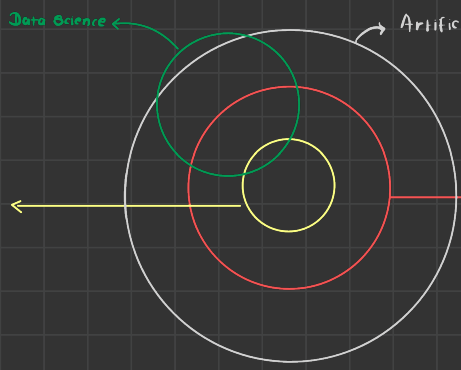
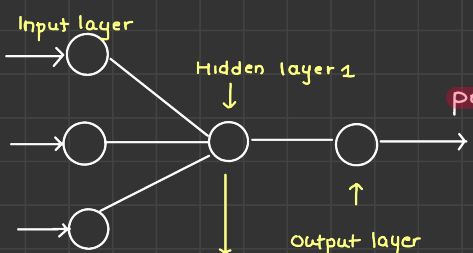
# Deep Learning

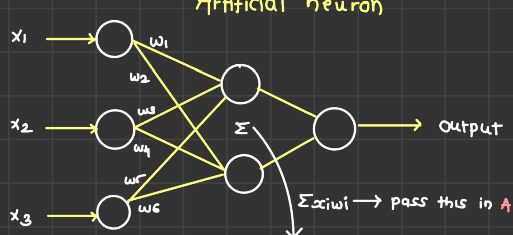
## AI vs ML vs DL vs DS




Perceptron → A single layer neural network




perceptron is a same as supervised algorithm




weight → tells you that how the neuron activated.

if weight = 0 than neuron will get deactivated so to solve this problem we have bias parameter.

$\Sigma x_i w_i$ → pass this in Activation function

$\Sigma x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_5 w_4 \Rightarrow w^T x + \text{bias}$

# Activation function

Basically activation decide the whether the neuron should be activated or not

lets take sigmoid activation function (Binary classification)

$$\sigma = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-(x_1 w_1 + b)}} \Rightarrow \text{output b/w 0 to 1}$$

threshold value = 0.5

$\sigma \geq 0.5 \longrightarrow$ Activated neuron

$\sigma < 0 \longrightarrow$ deactivated neuron

# Forward propogation

Passing inputs with weight through activation function to generates output.

Inputs $\longrightarrow$ Activation $\longrightarrow$ output

# Backward Propogation

Suppose we got the output 0 but actual it is 1

Now calculate error or difference between $y = 1$ and $\hat{y} = 0$

$\therefore$ loss function $= y - \hat{y}$

$= 1 - 0$

$= 1$ → It should be a minimal

→ To reduce this we need to update the weights & biases

Reverse the process of forward propogation

We need optimization algo to update weights

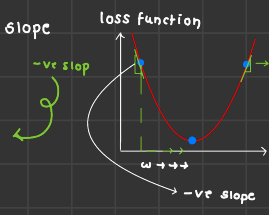
↓

GRADIEM DESCENT

## Update formula for weight

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w_{old}}$$ → derivation of loss with respect to $w_{old}$

$\frac{\partial L}{\partial w_{old}}$ → Calculating slope



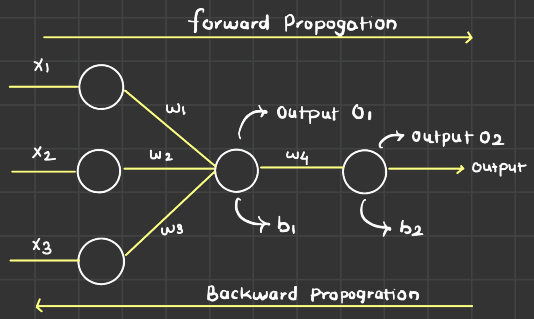

$w_{new} = w_{old} - \alpha(-ve)$

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

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

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

Chain rule of differentiation




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

lets say that we want to update $w_4$

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

By chain rule

Weight → $$\frac{\partial L}{\partial w_{4old}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial w_{4old}} = \frac{\partial L}{\partial w_{4old}}$$

want to update bias

$$b_{2new} = b_{2old} - \eta \frac{\partial L}{\partial b_{2old}}$$

By chain rule

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

By chain rule

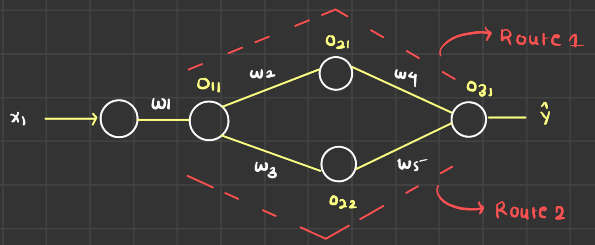
$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial o_1} * \frac{\partial o_1}{\partial w_{1old}}$$

$$w_{2new} = w_{2old} - \eta \frac{\partial L}{\partial w_{2old}}$$

By chain rule

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

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

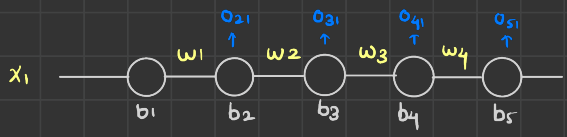



$$w_{1new} = w_{1old} - \eta \frac{\partial L}{\partial w_{1old}}$$

↳ By chain rule

$$\frac{\partial L}{\partial w_{1old}} = \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1old}}\right] + \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{22}} * \frac{\partial o_{22}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1old}}\right]$$

# VANISHING GRADIENT PROBLEM

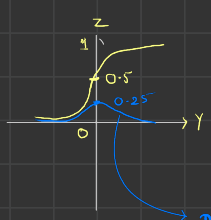



$$\text{mse loss} = \frac{1}{2}(y - \hat{y})^2$$

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

$$\frac{\partial L}{\partial w_{1new}} = \frac{\partial L}{\partial o_{51}} * \frac{\partial o_{51}}{\partial o_{41}} * \frac{\partial o_{41}}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial w_{1old}}$$

Sigmoid activation




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

→ 0 or 1

≥ 0.5 → 1

< 0.5 → 0

Derivative of $\sigma$ always ranging between 0 to 0.25

lets say we got derivative of $\sigma = 0.25$

$0.25$ → this will get reduce in backward propogation

$0.25 * 0.15 * 0.10 * 0.05 * 0.02$ = very small value

$w_{1new} = w_{1old} - \eta$ (very small value)

↳ also small

Than the result → $w_{new} = w_{old} - 0$

↳ Not changing any more

And this is problem called **vanishing gradient descent** that lead not updating weights any more

To solve this problem we have different types of activation functions.

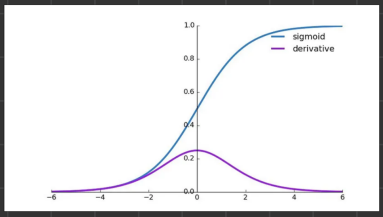
1.) Tanh

2.) Relu

3) Leaky Relu

and many more are there.

## A SIGMOID ACTIVATION FUNCTION




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

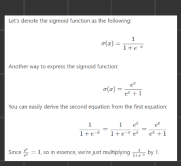
Advantages:

- Convergence become more faster due to smoothness of curve.
- Output between 0 to 1.
- Also give clear prediction.

Disadvantage:

- prone to vanishing gradient.
- function output is not zero-centerred.
- Computationally cost → perform exponential operation

Derivation of Sigmoid:



Let's denote the sigmoid function as the following:

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

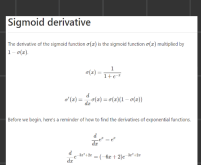
Another way to express the sigmoid function:

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

You can easily derive the second equation from the first equation:

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

Since $\frac{e^x}{e^x} = 1$, so in essence, we're just multiplying $\frac{e^x}{e^x+1}$ by 1.



Sigmoid derivative

The derivative of the sigmoid function $\sigma(x)$ is the sigmoid function $\sigma(x)$ multiplied by $1 - \sigma(x)$.

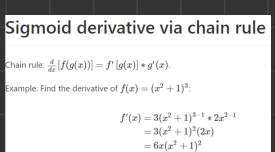
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$$

Before we begin, here's a reminder of how to find the derivatives of exponential functions:

$$\frac{d}{dx}e^x = e^x$$

$$\frac{d}{dx}e^{-kx^2+3x} = (-4kx+3)e^{-kx^2+3x}$$



Sigmoid derivative via chain rule

Chain rule: $\frac{d}{dx}[f(g(x))] = f'[g(x)] * g'(x)$.

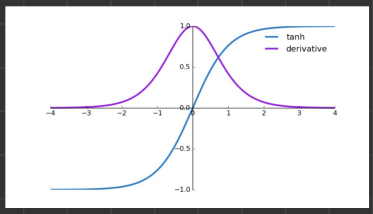
Example: Find the derivative of $f(x) = (x^2+1)^3$:

$$f'(x) = 3(x^2+1)^{3-1} * 2x^{2-1}$$
$$= 3(x^2+1)^2(2x)$$
$$= 6x(x^3+1)^2$$

sigmoid derivation below uses this rule.

$$\frac{d}{dx}\sigma(x) = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \frac{d}{dx}(1+e^{-x})^{-1}$$
$$= -1*(1+e^{-x})^{-2}(-e^{-x})$$
$$= \frac{-e^{-x}}{-(1+e^{-x})^2}$$
$$= \frac{e^{-x}}{(1+e^{-x})^2}$$
$$= \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}}$$
$$= \frac{1}{1+e^{-x}}\frac{e^{-x}+(1-1)}{1+e^{-x}}$$
$$= \frac{1}{1+e^{-x}}\frac{(1+e^{-x})-1}{1+e^{-x}}$$
$$= \frac{1}{1+e^{-x}}\left[\frac{(1+e^{-x})}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right]$$
$$= \frac{1}{1+e^{-x}}\left[1 - \frac{1}{1+e^{-x}}\right]$$
$$= \sigma(x)(1-\sigma(x))$$

## B. TANH ACTIVATION FORMULA




function output lies between -1 to 1

derivatives of tanh ⟶ 0 to 1

Basically when you create deep neural network it also creates problem of vanishing gradient descent.

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

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

it can be written as trignometric function

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

first order derivative is as follows

$$\frac{d}{dx}\tanh(x) = \frac{d}{dx}\frac{\sinh(x)}{\cosh(x)}$$

Applying rule of differention rule of division

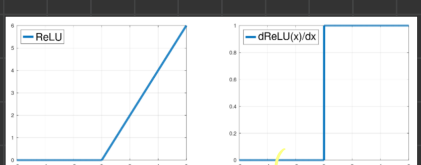
$$\frac{d}{dx}\frac{\sinh(x)}{\cosh(x)} = \frac{\cosh(x)\frac{d}{dx}\sinh(x) - \sinh(x)\frac{d}{dx}\cosh(x)}{\cosh^2(x)}$$

Division Rule

$$\frac{dy}{dx} = \frac{v\frac{du}{dx} - u\frac{dv}{dx}}{v^2}$$

$$\frac{d}{dx}\frac{\sinh(x)}{\cosh(x)} = \frac{\cosh^2 x - \sinh^2 x}{\cosh^2 x} = \frac{\cosh^2 x}{\cosh^2 x} - \frac{\sinh^2 x}{\cosh^2 x} = 1 - \tanh^2(x)$$

## 3. ReLU function: Rectified Linear Unit




derivation of relue

ReLU = max(0, x)

Whenever x is negative it return 0 otherwise it return 1

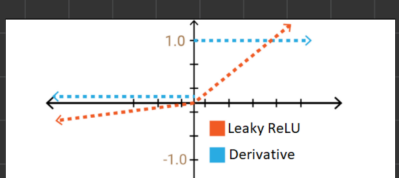
derivative of relu ⟶ 0 to 1

When the input is neg → Relu will die

Calculation is faster than tanh and sigmoid function

Relu is not 0 centric function

## 4. LEAKY RELU FUNCTION:

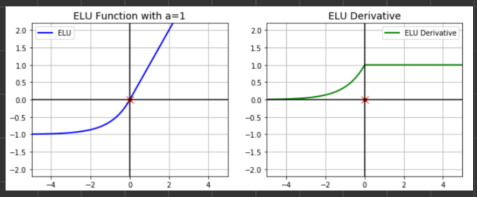



f(x) = max(0.01x, x)

Solve dead neuron problem

5. Elu (Exponential linear units) function:

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



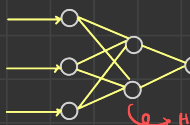

- Computationally expensive
- close to zero centered
- No dead Relu issues

6. SOFTMAX Activation function

Vectorized version of sigmoid activation function whereas sigmoid take scalar value

## Technique to choose activation function

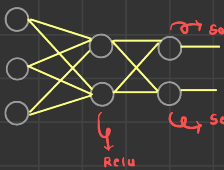
Binary Classification



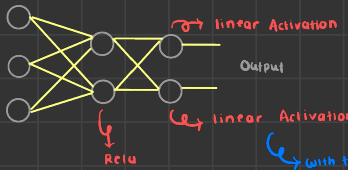

Convergence is not happen

change Relu → (prelu, Elu)

Multiclass Classification

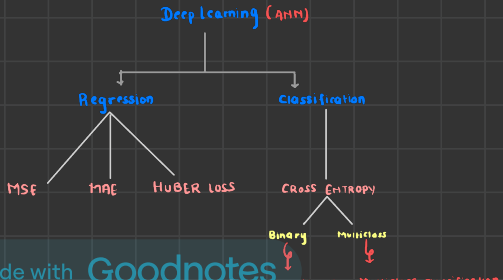



Regression




with this activation we use loss function

## Loss Functions



Cost function vs loss function

loss cause by batch data → not whole data but few data from whole data you have.

Overall loss cause by the data you pass into a forward propogation

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

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

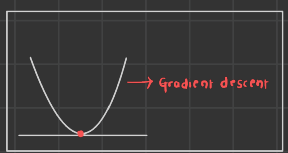
## Mean Squared Error (MSE):

loss function $= \frac{1}{2}(y-\hat{y})^2$ Cost function $= \frac{1}{2}\sum_{i=1}^{n}(y-\hat{y})^2$

↳ quadratic equation

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

$ax^2 + bx + c$




Advantage

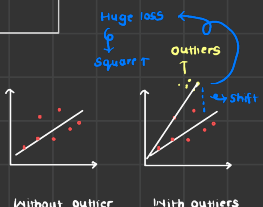
differentiable

It has only only local or global minima

Convergence much faster

Disadvantage

Not robust to outliers




## Mean Absolute Error (MAE)

loss function $= \frac{1}{2}|y-\hat{y}|$ cost function $= \frac{1}{2}\sum_{i=1}^{n}|y-\hat{y}|$

Advantage:

Robust to outliers

Disadvantage

Time consuming


## Huber Loss

Combination of MSE and MAE

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

When outliers are not present

↳ hyperparameter

## Binary Cross Entropy

loss $= -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$ → logistic reg

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

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

# Categorical Cross Entropy

| f1 | f2 | f3 | output |
|---|---|---|---|
| 2 | 3 | 5 | Good |
| 6 | 0 | 1 | Neutral |
| 1 | 4 | 1 | Bad |

→ multiclass classification

↳ Convert using one hot encoding

| Good | neutral | Bad |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

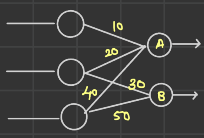
$$L(x_i, y_i) = -\sum_{j=1}^{C} y_{ij} * \ln(\hat{y_{ij}})$$ i = rows, j = column

$y_i = [y_{i1}, y_{i2}, y_{i3} \ldots y_{ic}]$

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

$\hat{y_{ij}}$ = Softmax Activation layers → output layer

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




$$= \frac{e^{10}}{e^{10+20+30+40+50}}$$ → prob → 0.4, 0.5 → like that

Sum of these = 1

if A's prob is high than it is output

## Conclusion.

Relu, Softmax → multiclass classification

Relu, Sigmoid → Binary classification

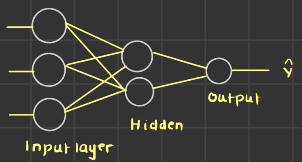
Relu, Linear activation → regression

↳ Loss function

# OPTIMIZER

## GRADIENT DESCENT

weight update formula → $W_{new} = W_{old} - \eta \frac{dL}{dW_{old}}$


$$cost = \frac{1}{2m}\sum_{i=1}^{m}(y-\hat{y})^2$$

↳ To minimize the loss we need optimizers

Epochs

↳ Training neural network with training data within one cycle

Disadvantage

Resource extensive (Huge RAM)

↳ In gradient descent

## STOCHASTIC GRADIENT DESCENT

Suppose I have 2000000's data

Advantage

Require less RAM

Disadvantage:

Convergence will take to much time

epoch 1

1 records → ŷ

← Calcwate loss than update weights } → 1st iteration

Epoch 2

2 records → ŷ

← Calculate loss than update weights } → 2nd iteration

|

|

|

|

|

n numbers of epoch's

## MINI BATCH SGD

Instead of passing 1000000 record once

↳ BATCH size = 10000

In every epoch I will send 10000

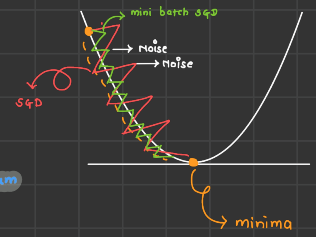
Advantage:

less resources intesive

Convergence will be a faster

Time Complexity → improved

Aim To reach global minima




How remove noise ?

↳ we use concept of momentum

## SGD WITH MOMENTUM

momentum makes the smoother the zig zag curve

Exponential weighted average

↳ Time Series → ARIMA, ARMA

Update Rule

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

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

$t \Rightarrow$ Current time

$t-1 \Rightarrow$ previous time

Exponential weighted average

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

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad \quad a_n$

$V_{t_1} = a_1$

value of $t_1 = a_1$

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

$\beta$ - hyperparameter → 0 to 1

↳ 0.95

$V_{t_2} = 0.95 * V_{t_1} + 0.05 * a_2$

↳ We are giving more important to $V_{t_1}$

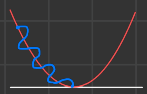
Removing noise and smoothening the curve

Exponential weighted avg

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

↳ value of derivation

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

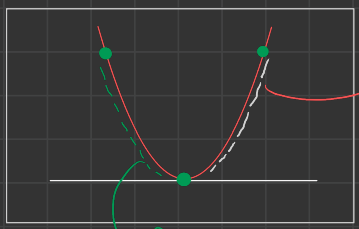


→ Reducing noise

→ Also work for mini batch

→ Convergence will be faster

## AJAGRAD ( Adaptive Gradient Descent )



We are changing the speed initially it is to fast taking big step as it move to closer the global minima taking smaller steps.

This step control by learning rate which constant in entire process

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

$\eta$ → fixed value

Change → we will make it adaptive

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

$\eta' = \frac{\eta}{\sqrt{\eta_t + \varepsilon}}$ → Already small

$\varepsilon$ → Small value to avoid infinity

if we divide → Suppose this is zero you will get infinity

by

$$\eta_t = \sum_{i=1}^{t} \left( \frac{\partial L}{\partial w_t} \right)^2$$ → This value keep on increasing (Σ)

Small / huge → huge

t=1 t=2 t=3

$\eta = 0.01$ $\eta = 0.05$ $\eta = 0.02$ → Reducing

$\eta'$ → 0

And your weight will not going to update

# Adadelta and Rmsprop

$$\eta' = \frac{\eta}{\sqrt{Sdw + \varepsilon}}$$

→ Applying weighted Average

↳ $Sdw = \beta$

Initialize $Sdw = 0$

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

$\beta = 0.95$

$$Sdw_t = 0.95 * Sdw_{t-1} + (0.05)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

↳ now not gonna very high

controlling

# Adam Optimizer (Best optimizer)

↳ Momentum + Rmsprop (Adaptive learning)

bias bias

$Vdw = 0$, $Vdb = 0$, $Sdw = 0$, $Sdb = 0$

$$w_t = w_{t-1} - \eta' Vdw \qquad \eta' = \frac{\eta}{\sqrt{Sdw + \varepsilon}}$$

$$b_t = w_{b_{t-1}} - \eta' Vdb$$

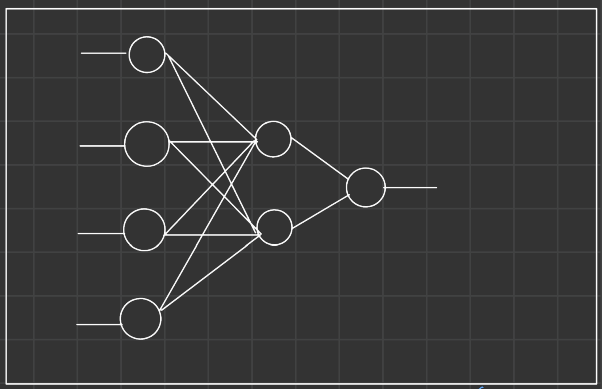
$$Vdw_t = \beta * Vw_{t-1} + (1-\beta)\frac{\partial L}{\partial w_t}$$

$$Vdb_t = \beta * Vdb_{t-1} + (1-\beta)\frac{\partial L}{\partial b_t}$$

> Smoothening the curve to reduce noise make convergence faster

> Also making learning rate adaptive

# Artificial Neural Network



Sequential

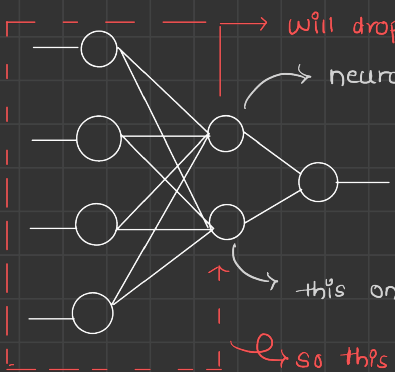
Dense
- neurons getting input from previous neuron.

forward
- inputs → output layer

Backward

Activation
- Whether the neurons are activated or deactivated

Dropout



- will drop this neuron
- neuron is highly activated
  - it will learn more than this
- this one is not activated
  - so this will get activate & start learn

- kind of overfitting

# Convolution Neural Network

## CNN vs Human brain


Image

Human Brain

Cerebral Cortex

Also know as visual cortex

It has many layers

passing data in layers

Output

Will recognize or identify the image

## Convulation neural network

Image
- RGB → it has 3 channel
- B/W → it has 1 channel

### Black and White Image

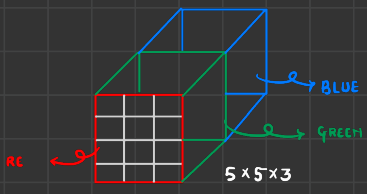
pixels value lies between 0 to 255

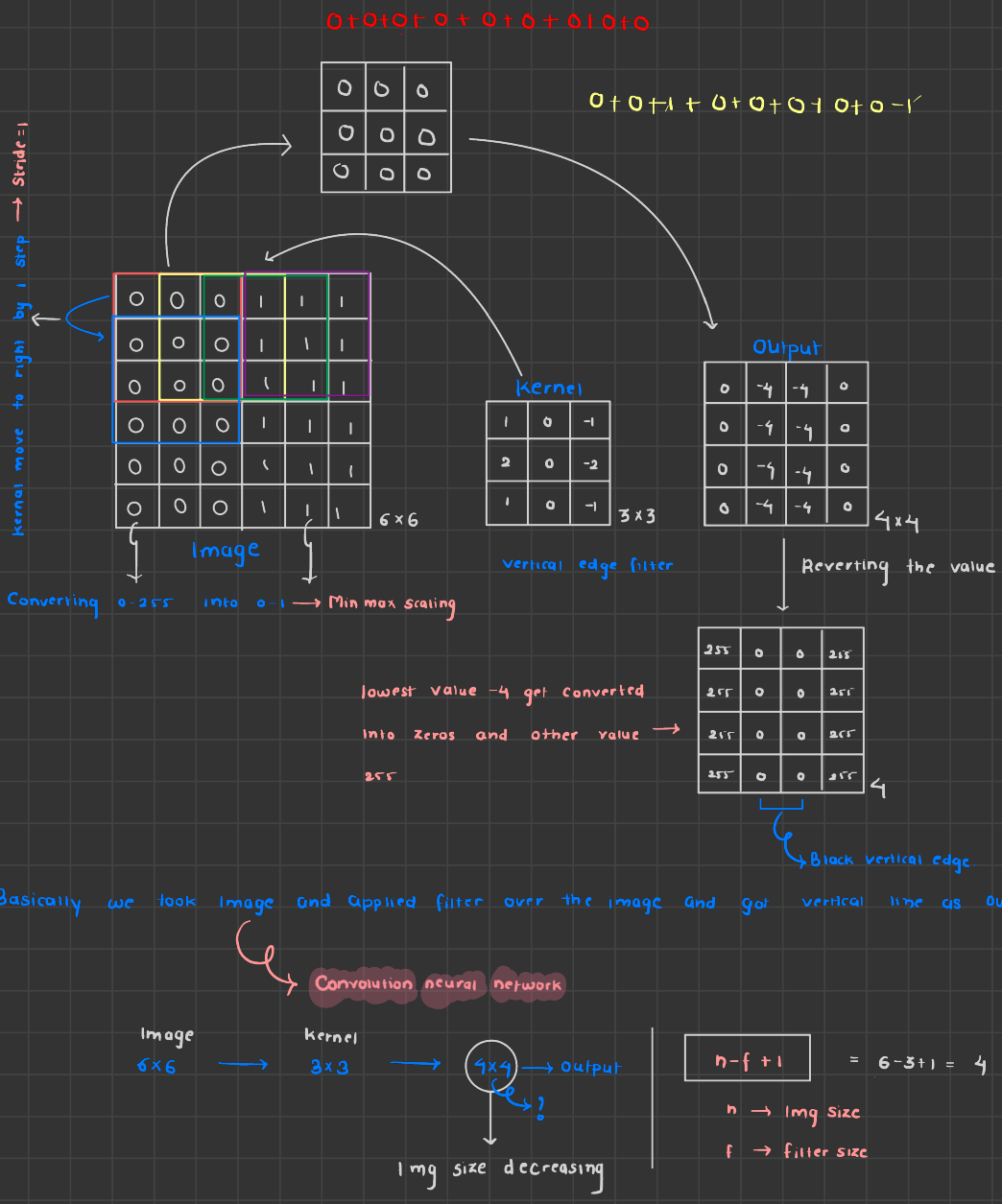
0 - BLACK

255 - WHITE

4x4

### RGB Image



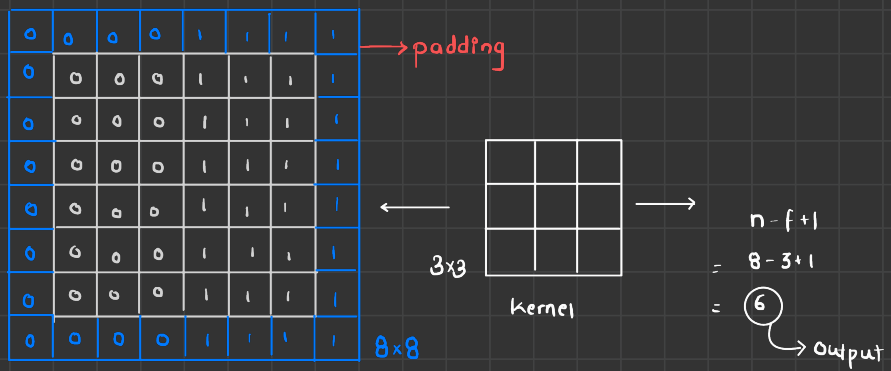


6 x 6 → 3x3 → 4x4

we are loosing information

To avoid this

# PADDING




Padding help to avoid the problem of loosing information in convolution

Now after padding formula = $n + 2p - f + 1$ → $\frac{n+2p-f+1}{s}$

s → Stride

n = input size

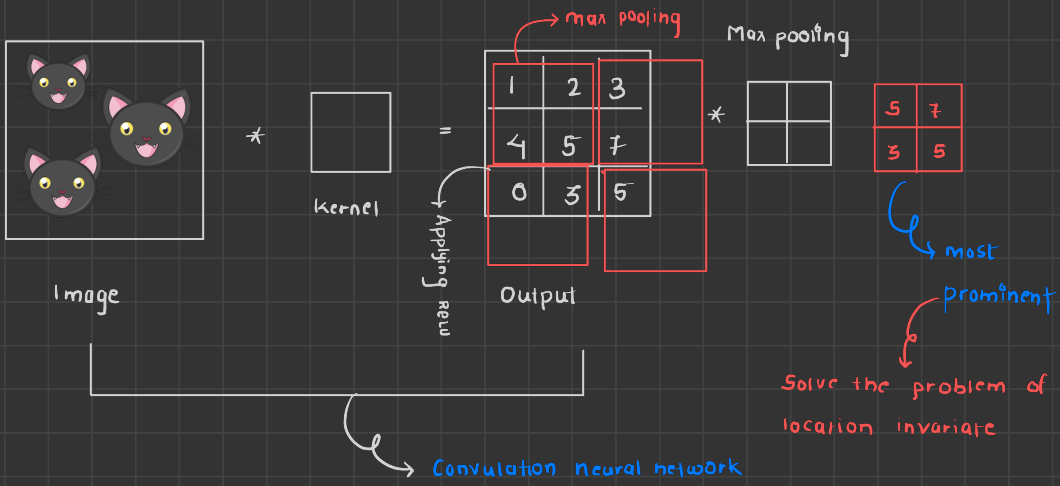
p = padding

f = kernel size

n = 6, k = 3, p = 1 $\quad 6 + 2(1) - 3 + 1$

$= \underline{6}$

# MAXPOOLING




# FLATENING

→ like ANN Dense layer