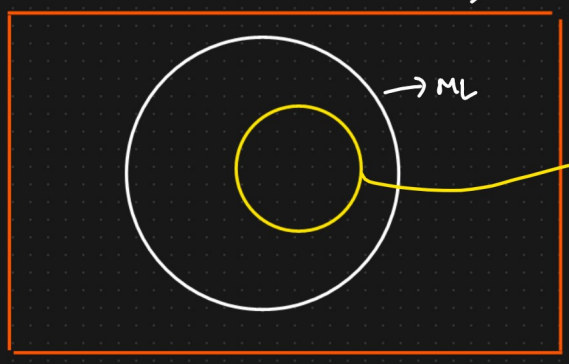
# Deep Learning




# Deep Learning

1. ANN → Artificial Neural N/W → Classification, Regression
2. CNN → Convolutional Neural N/W → I/P : Images, Video frames → RCNN, MASKED RCNN, Detectron, YOLO V5, V6, V7 (Computer Vision, Objection Detection)
3. RNN → Recurrent Neural N/W → NLP → NLP, Time Series

I/P: Text, Time Series

⇓

{ Word Embedding, LSTM RNN, GRU RNN, Bidirectional LSTM RNN, Encoder Decoder, Transformers, BERT }

FRAMEWORK

TENSORFLOW

End to End Project

## ② Why Deep Learning Is Becoming popular?

2005 ⟶ Facebook, Youtube, Whatsapp, Linked, Twitter } ⇒ Social Media platform

⋮

2011-2012

⇓

DATA WAS GENERATED

Exponentially

Image, Text, Documents, Videos

↳ Big DATA ⟶ HADOOP → unstructured DATA

⇓

2011 ⇒ Cloudera, Hortonworks

Netflix → movie streaming platform

⇓

Recommendation System

⇓

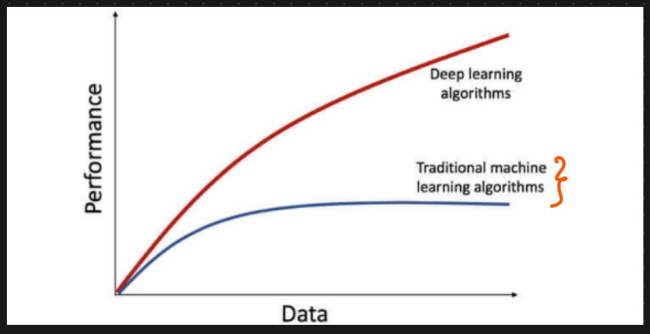
Increase Revenue

2011-2012 :

① Hardware Requirement ⟶ GPU's cost↓↓↓ → NVidia GPU Price↓↓↓

TRAIN MODELS

② Huge amount of data is getting generated → Deep Learning Models Perform well




③ Deep learning is been used in Many Domains

1. Medical
2. ECommerce
3. Retails
4. Marketing . . . . .

④ Frameworks Open source

Community size ↑↑

Tensorflow

⇓

Google

Pytorch

⇓

Facebook

⇓

More Research

# ③ Perceptron [Artificial Neuron or Neural Network Unit]

① Input layer ✓

② Hidden layer ✓

③ Weights ✓

④ Activation function ✓

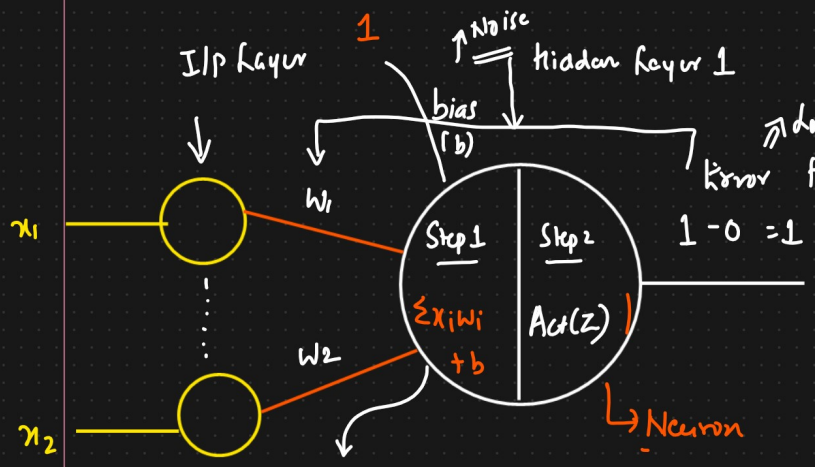
[Single layered NN]

⇓

Binary classifier

DATASET

| | $x_1$ Id | $x_2$ No. of study hours | o/p Pass/Fail |
|---|---|---|---|
| → | 95 | 3 | 0 |
| → | 110 | 4 | 1 |
| → | 100 | 5 | 1 |



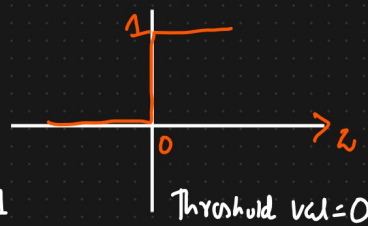

$$z = w_1 x_1 + w_2 x_2 + b$$

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

**Activation function**

Transform the output between 0 to 1, -1 to 1

**Step Function**

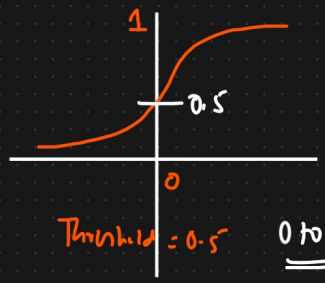



0 to 1

Threshold val = 0

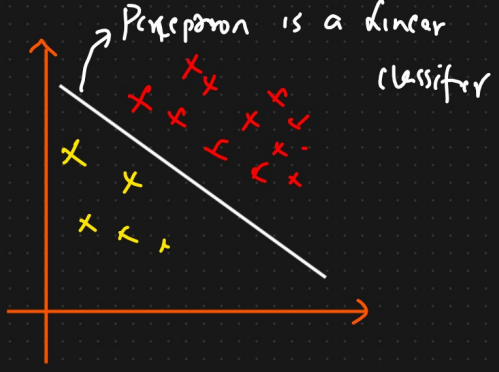
$$\begin{cases} 0 & z \le 0 \\ 1 & z > 0 \end{cases}$$

**Sigmoid Function**




Threshold = 0.5 0 to 1

$$\begin{cases} 1 & z > 0.5 \\ 0 & z \le 0.5 \end{cases}$$




**Step 1**

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

$$z = b + w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n$$

$$y = mx + c$$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \; - - - + \beta_n x_n$$

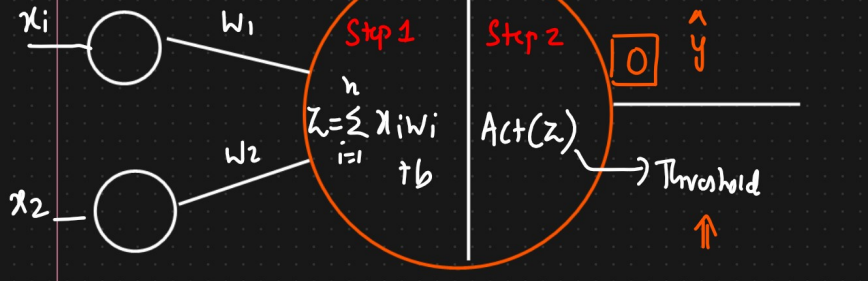
linear problem Statement

# Perceptron Models

- Single Layer Perceptron Model
- Multi Layered Perceptron Model [ANN]

## Single Layer Perceptron Model

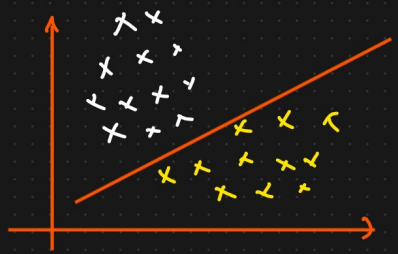
Feed Forward Neural NW




## Multi Layered Perceptron Model

1. Forward Propogation
2. Backward "
3. Loss functions
4. Activation functions
5. Optimizers

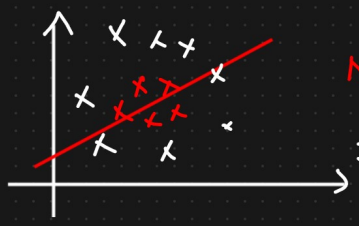
## linear Separable problem



Binary classification

⇓

linearly separable



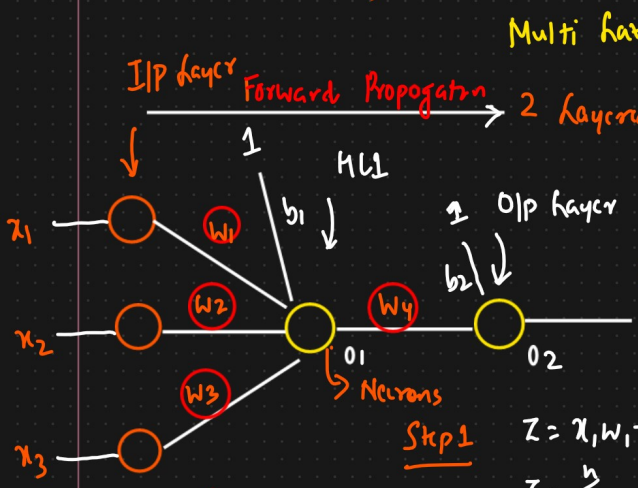
Non linear separable

⇒ Multi Layered NN

# 5 Multi Layered Perceptron Model [Artificial Neural N/W]

① Forward Propogation
② Backward Propogation → Geoffrey Hinton →

③ Loss function
④ Optimizers ✓
⑤ Activation function:




| $x_1$ | $x_2$ | $x_3$ | Pass/Fail |
|---|---|---|---|
| IQ | Study hours | Play hours | O/p |
| 95 | 4 | 4 | 1 |
| 100 | 5 | 2 | 1 ← |
| 95 | 2 | 7 | 0 |

Step 1 $z = x_1w_1 + x_2w_2 + x_3w + b_1$

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

① Forward Propogation 2: Activation(z)

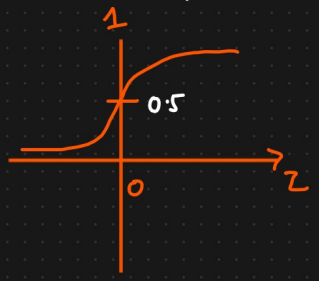
$W_1$ $W_2$ $W_3$ [0.01, 0.02, 0.03] [0.001]

Hidden Layer 1

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

$= 1.151$

Sigmoid $= \frac{1}{1+e^{-z}}$




Step 2: Activation (z)

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

$O_1 = 0.759$

Hidden Layer 2

$W_4 = 0.02$

$b_2 = 0.03$

Step 1: $z = O_1 * W_4 + b_2$

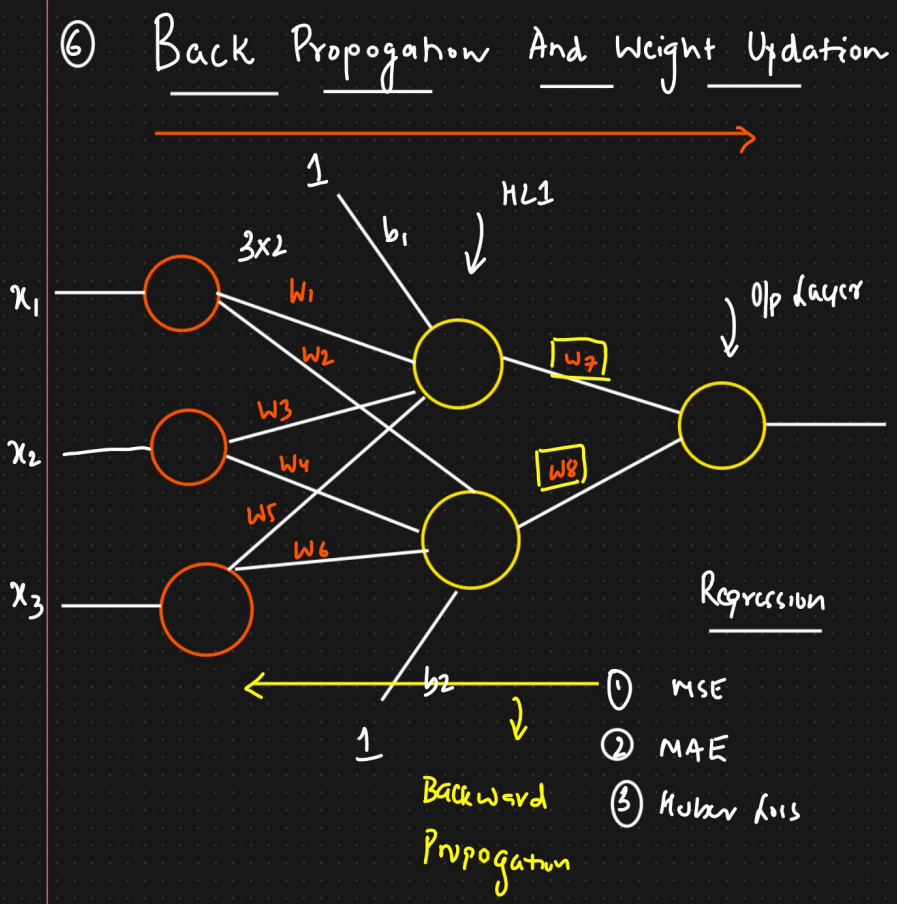
$= 0.759 * 0.02 + 0.03$

$= 0.045180$

Step 2 : Activation(z) $\frac{1}{1+e^{-(0.04518)}} = 0.511129$

$O_2 = 0.511129 \Rightarrow \hat{y}$

## ⑥ Back Propogation And Weight Updation Formula




| I/p features → | | | ↓ |
|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | Pass/Fail |
| IQ | Study hours | Play hours | O/p |
| > 95 | 4 | 4 | 1 |
| → 100 | 5 | 2 | 1 ← |
| 95 | 2 | 7 | 0 |

Loss function $(y-\hat{y})^2$ ↓↓↓↓↓↓

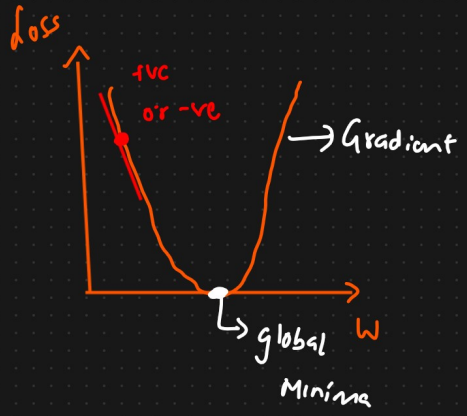
**Regression**
1. MSE
2. MAE
3. Huber Loss

**Classification**
1. Binary Cross Entropy
2. Categorical CROSS Entropy

### Weight updation Formula

$$W_{7_{new}} = W_{7_{old}} - \eta \left[ \frac{\partial L}{\partial W_{7old}} \right]$$ ← slope

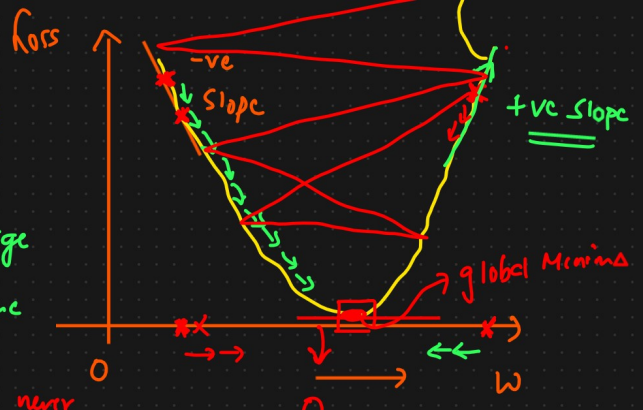
$$W_{8new} = W_{8old} - \eta \left[ \frac{\partial L}{\partial W_{8old}} \right]$$ ← Learning Rate




$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$ ⇒ Weight Updation Formula.

Gradient Descent ↗ Optimizers

**Optimizers** : To reduce the loss value




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

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

$W_{new} >> W_{old}$

$\eta$ = large value ⇓ It may never converge

↳ learning Rate

0.001 → Small value.

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

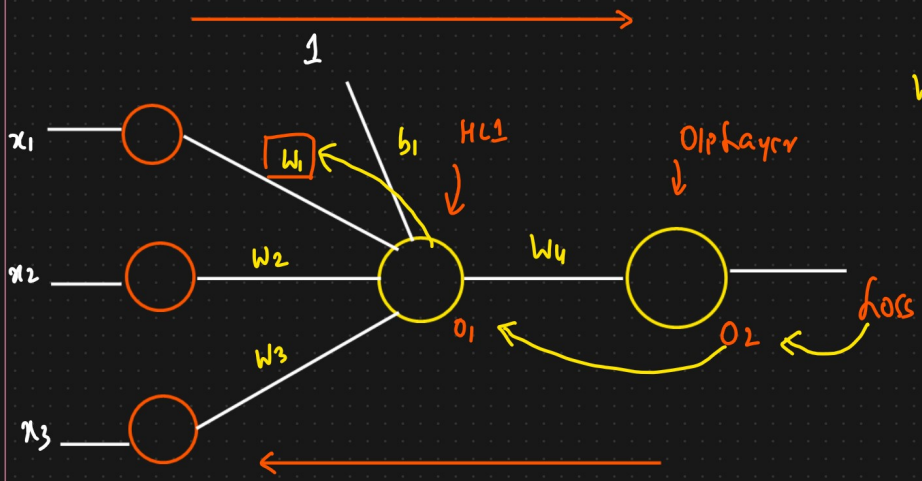
$= W_{old} - \eta\,(+ve)$

$W_{new} << W_{old}$

When W reaches global Minima

$W_{new} = W_{old}$

## ⑦ Chain Rule of Derivative




$$W_{new} = W_{old} - \eta \frac{\partial h}{\partial W_{old}}$$

$$W_{4_{new}} = W_{4_{old}} - \eta \frac{\partial h}{\partial W_{4_{old}}}$$

$$\frac{\partial h}{\partial w_{4old}} = \frac{\partial h}{\partial o_2} * \frac{\partial o_2}{\partial w_{4old}} \Rightarrow \text{Chain Rule of Derivation}$$

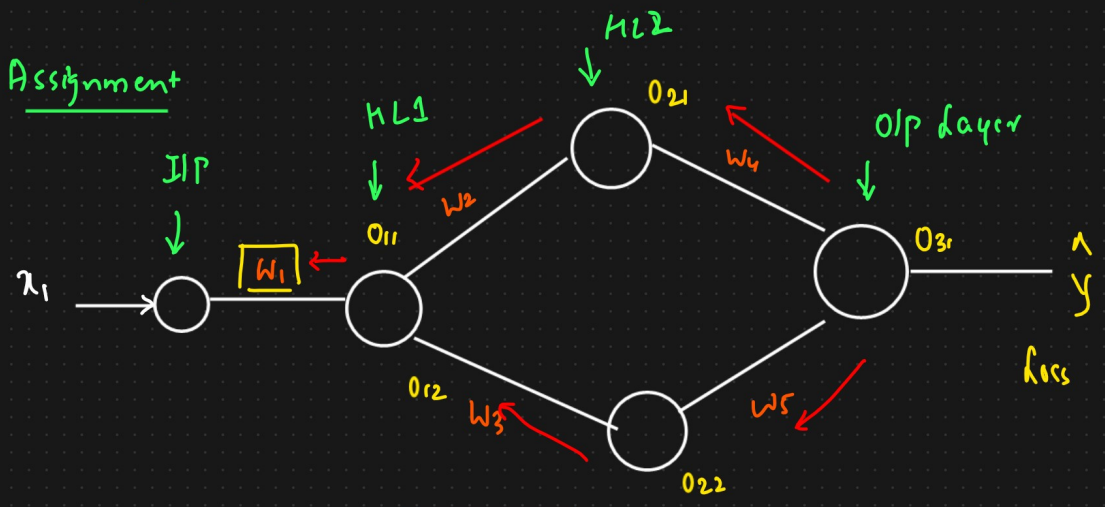
$$w_{1new} = w_{1old} - \eta \boxed{\frac{\partial h}{\partial w_{1old}}}$$

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

$w_{2new}$

$w_{3new}$.

Assignment




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

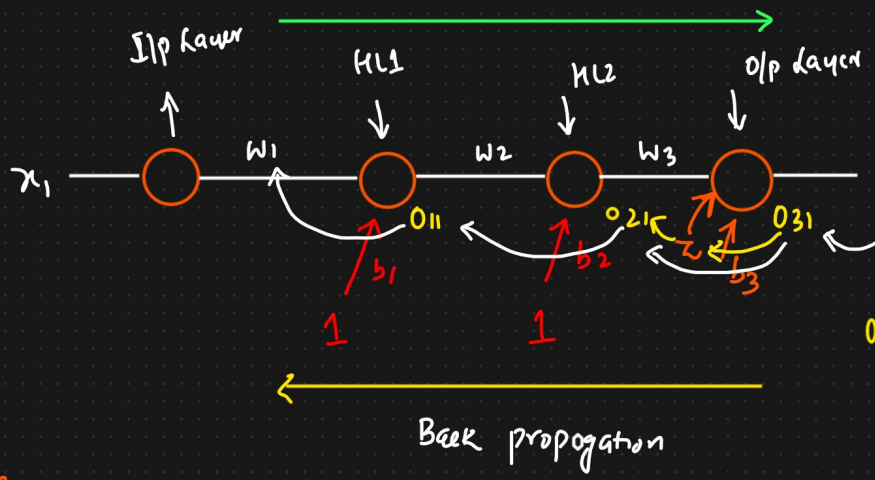
$$\Downarrow$$

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

$$+$$

$$\left[\frac{\partial h}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{22}} * \frac{\partial o_{22}}{\partial o_{12}} * \frac{\partial o_{12}}{\partial w_{1old}}\right]$$

# ⑧ Vanishing Gradient Problem And Activation functions $\phi(z)$

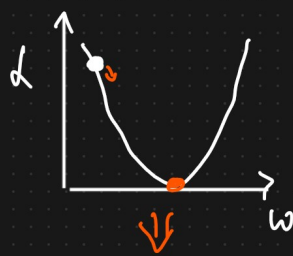



$$z = \sum_{i=1}^{n} w_i x_i + b$$

$\sigma(z) \Rightarrow 0$ to $1$

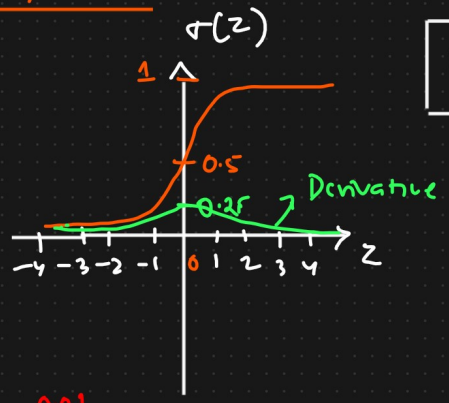
↳ Sigmoid

Optimizers




## ⑨ Sigmoid Activation function

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




$$0 \le \sigma(z) \le 1$$

Derivative $(\sigma(z))$

$$0 \le \sigma(z) \le 0.25$$

Vanishing Gradient Problem.

$$W_{1\,new} = W_{1\,old} - \eta \left[ \frac{\partial h}{\partial W_{old}} \right]$$

$\eta$ = 0.01

$\frac{\partial h}{\partial W_{old}} \Rightarrow$ small value

$\Rightarrow$ small value

$$W_{1\,new} \approx W_{1\,old}$$ (Equal)

$$\frac{\partial h}{\partial W_{1old}} = \frac{\partial h}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial O_{11}} * \frac{\partial O_{11}}{\partial w_{11}}$$

$0-0.25 * 0-0.25 * 0-0.25 \Rightarrow 0.000009$

$\Downarrow$

$\Rightarrow$ Smaller value

$$O_{31} = \sigma(\underbrace{w_3 * O_{21} + b_3}_{z})$$

$z = w_3 * O_{21} + b_3$

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

$$\frac{\partial O_{31}}{\partial O_{21}} = \frac{\partial(\sigma(z))}{\partial(z)} * \frac{\partial z}{\partial O_{21}}$$ (Sigmoid)

{Chain Rule}.

$0 \leq \sigma(z) \leq 0.25$ * $\frac{\partial((w_3 * o_{21}) + b_3)}{\partial(o_{21})}$

⇓

Derivative of Sigmoid

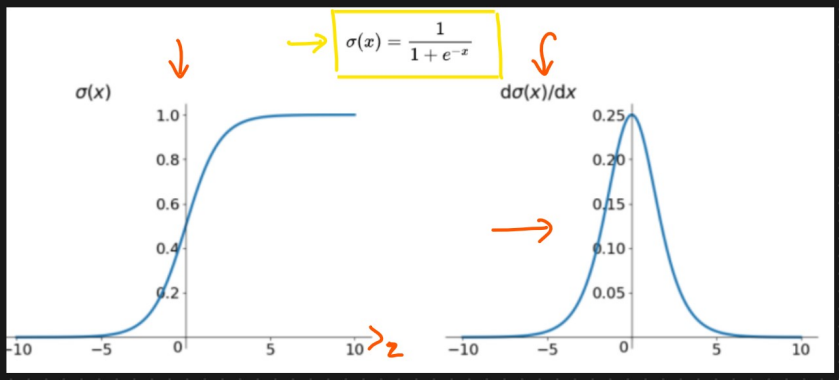
$$\frac{\partial o_{31}}{\partial o_{21}} = 0 \leq \sigma(z) \leq 0.25 * w_{3_{old}}$$

* To fix this problem Researchers started exploring other Activation function

① Tanh ② Relu ③ Prelu ④ Swiss

# Activation Functions

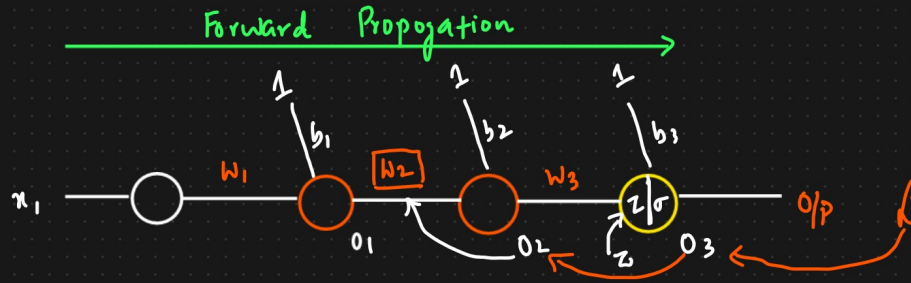
① Sigmoid Activation function [0 to 1] $z = \sum_{i=1}^{n} w^T x + b$




$\sigma(z) \Rightarrow$ 0 to 1

$\phi(z) \Rightarrow$

$\frac{\partial \sigma(z)}{\partial z}$ = 0 to 0.25





$$W_{2new} = W_{2old} - \eta \left[ \frac{\partial h}{\partial W_{2old}} \right]$$

$\eta$ = 0.001

$\frac{\partial h}{\partial W_{2old}}$ = 0.00001

$\Rightarrow W_{2new} \approx W_{2old} \Rightarrow$

$$\frac{\partial h}{\partial W_{2old}} = \frac{\partial h}{\partial O_3} * \frac{\partial O_3}{\partial O_2} * \frac{\partial O_2}{\partial W_2}$$

0.20 * 0.01 *

Let $z = (O_2 * W_3) + b_3$

$$\frac{\partial O_3}{\partial O_2} = \frac{\partial(\sigma(z))}{\partial z} * \frac{\partial z}{\partial O_2}$$

[0 to 1]

$$= [0 - 0.25] * \frac{\partial[(O_2 * W_3) + b_3]}{\partial O_2}$$

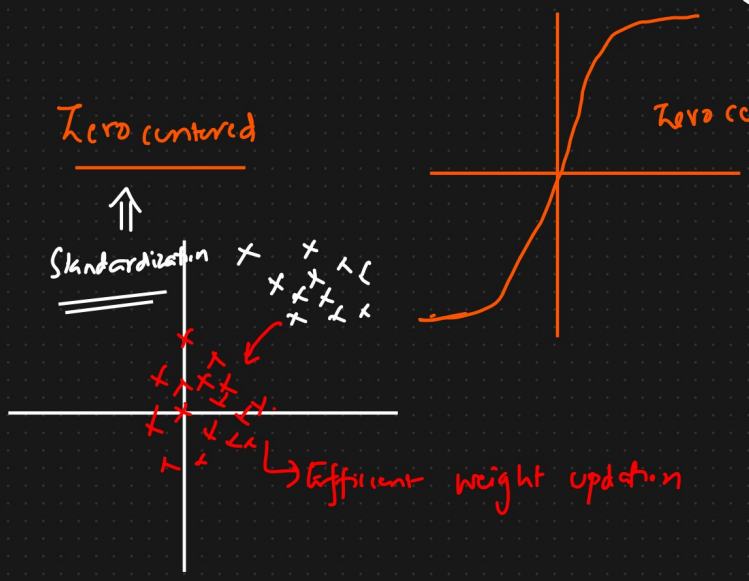
$$= [0 - 0.25] * W_3 \Rightarrow \text{Small value} \Rightarrow$$

## Advantages

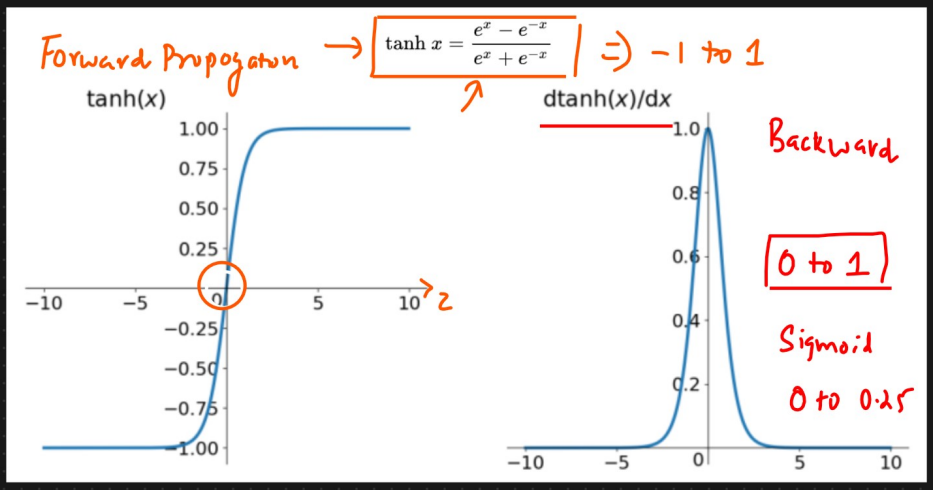
1. Binary classification Suitable.
2. Clear prediction i.e very close 1 or 0

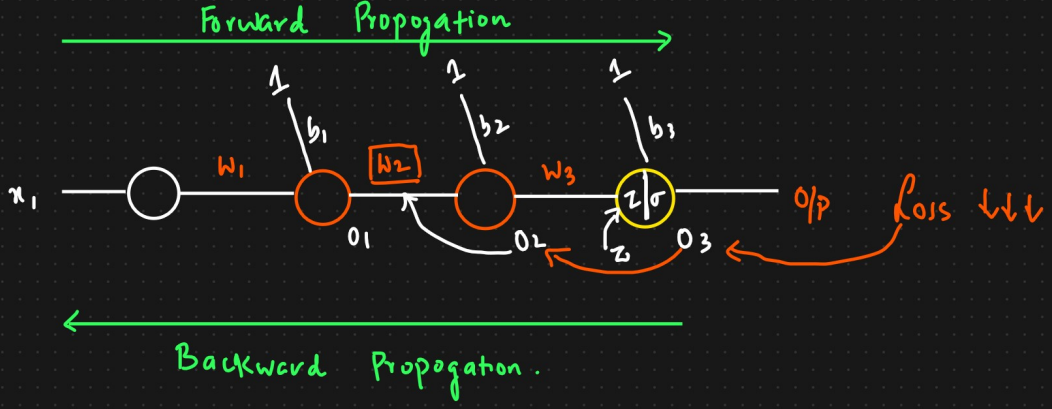
## Disadvantages

1. Prone to vanishing gradient Problem.
2. Function output is not Zero centred ⇒ Efficient weight updation
3. Mathematical operation are relatively time Consuming




## ② Tanh Activation Function



$$\frac{\partial h}{\partial w_{2old}} = \frac{\partial h}{\partial o_3} * \boxed{\frac{\partial o_3}{\partial o_2}} * \frac{\partial o_2}{\partial w_2}$$

$0.20_{11} * 0.01 \times$

Let $z = (o_2 * w_3) + b_3$

$$\frac{\partial o_3}{\partial o_2} = \boxed{\frac{\partial(\tanh(z))}{\partial z}} * \frac{\partial z}{\partial o_2}$$

[0 to 1]

$$= [0-1] * \frac{\partial[(o_2 * w_3) + b_3]}{\partial o_2}$$

$$= [0-1] * w_3 \Rightarrow \text{Small value} \Rightarrow$$

## Advantages

① Zero Centric ⇒ Weight updation is Efficient

## Disadvantages

① Prone to Vanishing Gradient Problem

② Time complexity

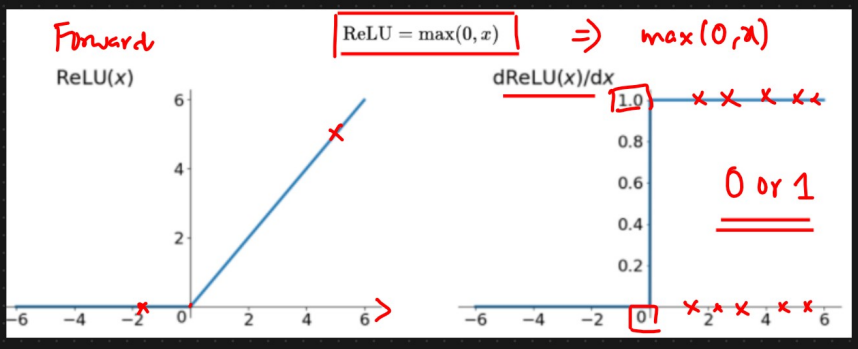
## ③ Relu Activation Function

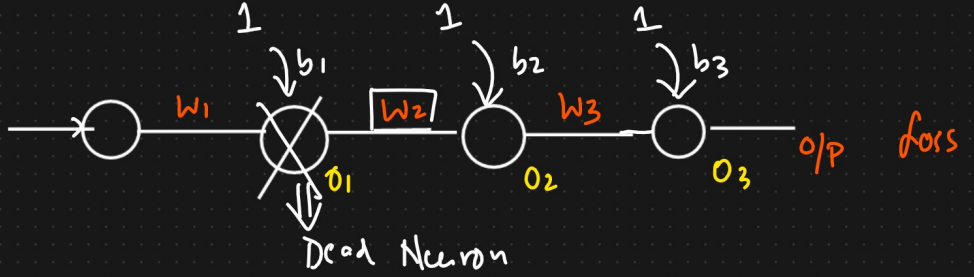
[Rectified Linear Unit].

Tanh ⇒ 0 to 1

Sigmoid ⇒ 0 to 0.25

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



$$\frac{\partial h}{\partial w_{2old}} = \frac{\partial h}{\partial o_3} * \boxed{\frac{\partial o_3}{\partial o_2}} * \frac{\partial o_2}{\partial w_2}$$

$0.20_{11}$ * 0.01 ×

let $z = (o_2 * w_3) + b_3$

$$\frac{\partial o_3}{\partial o_2} = \boxed{\frac{\partial (\text{relu}(z))}{\partial z}} * \frac{\partial z}{\partial o_2}$$

[0 to 1]

$$= [0 \text{ or } 1] * \frac{\partial [(o_2 * w_3) + b_3]}{\partial o_2}$$

$$= [0 \text{ or } \underset{+ve}{1}] * w_3 \quad \Rightarrow \text{Small value} \Rightarrow$$

(-ve)

Derivative of

If Relu output is [1] ⇒ Weight updation will happen

If Relu output is [0] ⇒ Dead Neuron

$$W_{2new} = W_{2old} - \eta \boxed{\frac{\partial h}{\partial w_{2old}}} \Rightarrow 0$$

If Derivative of Relu(z) is 0

$$\boxed{W_{2new} \approx W_{2old}} \Rightarrow \text{Dead Neuron}$$

If $z = +ve$ $\quad \frac{\partial \text{Relu}(z)}{\partial z} = 1$

If $z = -ve$ $\quad \frac{\partial \text{Relu}(z)}{\partial z} = 0$

## Advantages

① Solving Vanishing Gradient Problem

② Max(0, x) → Calculation is superfast. The Relu function has a linear Relationship

③ It is much faster than Sigmoid or Tanh.

## Disadvantages

① Dead Neuron

② Relu function O/P

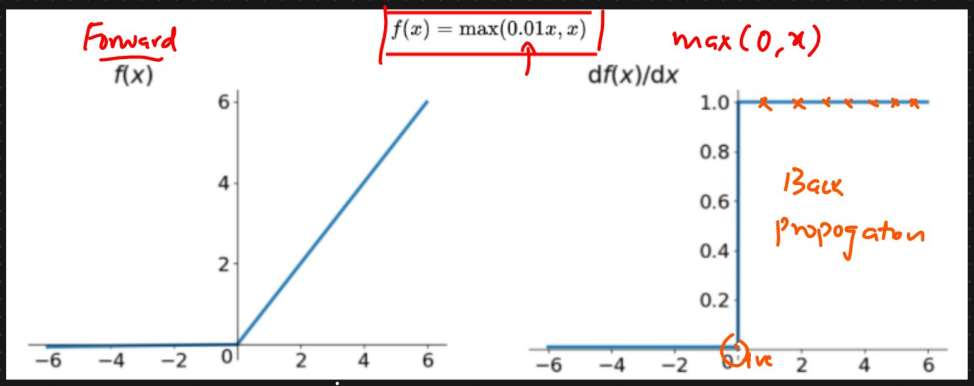
(0, x) ⇒ 0 or +ve number

⇓

It is not zero centric

## ④ Leaky Relu And Parametric Relu

max(αx, x) → hyperparameter  α = alpha = 0.01, 0.02, 0.03




Relu ⟶ Dead Neuron ⟶ Dead Relu Problem

## Advantages

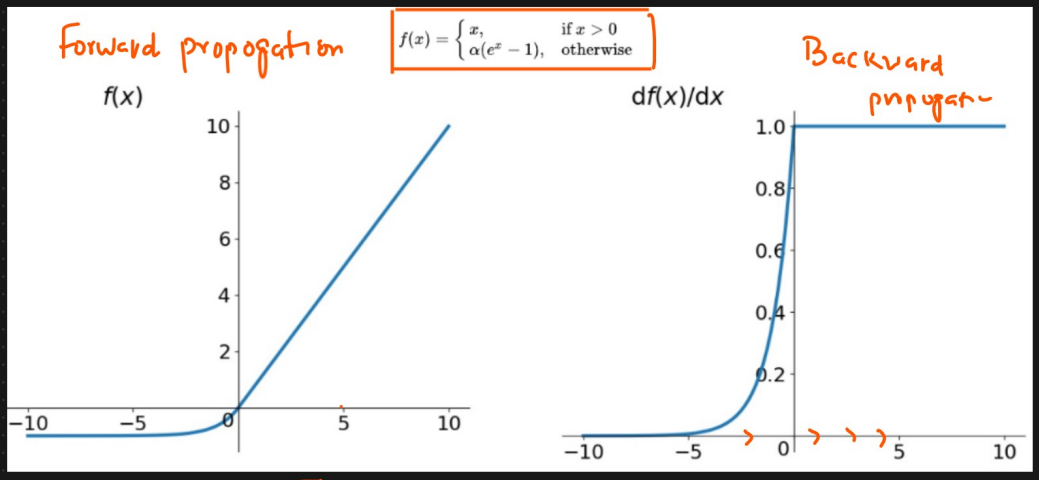
① Leaky Relu has all the advantage of Relu

② It removes the Dead Relu Problem

## Disadvantage

① It is not zero centric

# ⑤ ELU (Exponential Linear Units)




[It is used to solve Relu problems]

**Advantages**

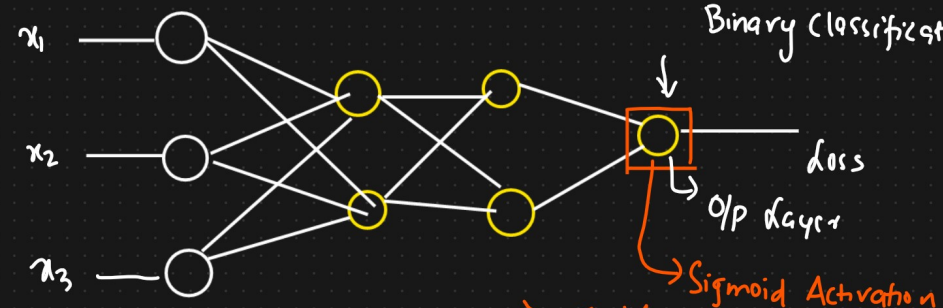
① No Dead Relu Issues

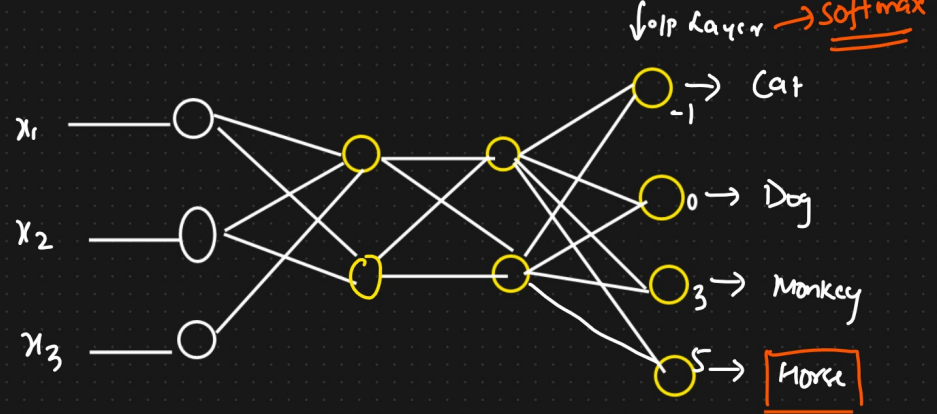
② Zero centered

**Disadvantage**

1) Slightly more computationally Intensive.

# ⑥ Softmax Activation function [Multiclass classification Problem]




**Softmax Activation**

$$\text{Softmax} = \frac{e^{y_i}}{\sum_{K=0}^{n} e^{y_K}}$$

$y_i = 0 * w + b$

Softmax $\Rightarrow$ Cat $= \frac{e^{-1}}{e^{-1+0+3+5}} = 0.00033$

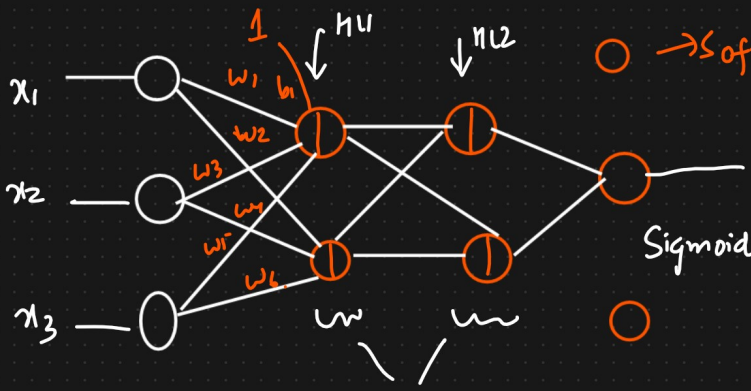
Dog $= \frac{e^{0}}{e^{-1+0+3+5}} = 0.0024$

Monkey $= \frac{e^{3}}{e^{-1+0+3+5}} = 0.0183$

Horse $= \frac{e^{5}}{e^{-1+0+3+5}} = 0.1353$

$Pr(\text{Horse}) = \frac{0.1353}{0.00033 + 0.0024 + 0.0183 + 0.1353} \approx 86\%$

## ⑦ Which Activation Function To Use When ?




↓ O/P layer

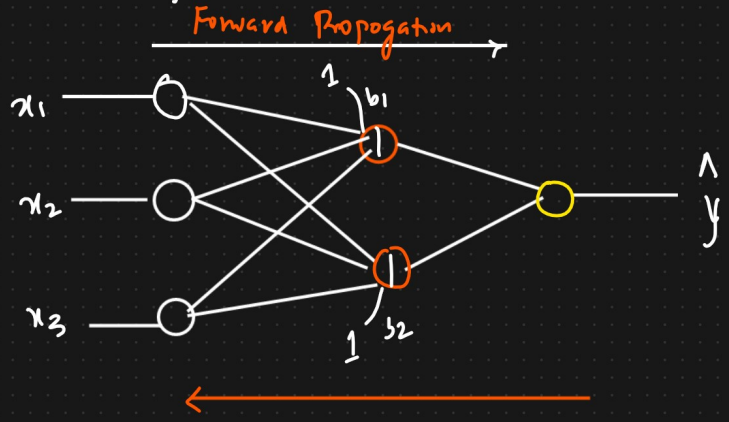
→ Softmax ⇒ ⇓ Multiclass Classification

Sigmoid → Binary classification

Hidden layers → Sigmoid or Tanh ⇓ Vanishing Gradient Problem

Relu or its variant Prelu, Leaky Relu, Elu

# Loss function And Cost Function




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

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

Optimizers

Gradient Descent

| $x_1$ | $x_2$ | $x_3$ | $\hat{y}$ O/P |
|---|---|---|---|
| - | - | - | 0 |
| - | - | - | 1 |
| - | - | - | 0 |
| - | - | - | 1 |

## Loss function

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

## Costfunction

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

## ANN

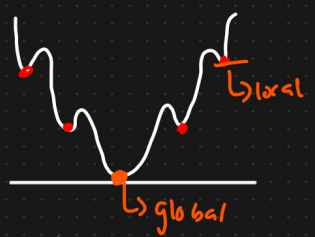
- Classification
- Regression
  1. Mean Squared Error (MSE)
  2. Mean Absolute Error (MAE)
  3. Huber Loss
  4. RMSE

## 1. Mean Squared Error (MSE)

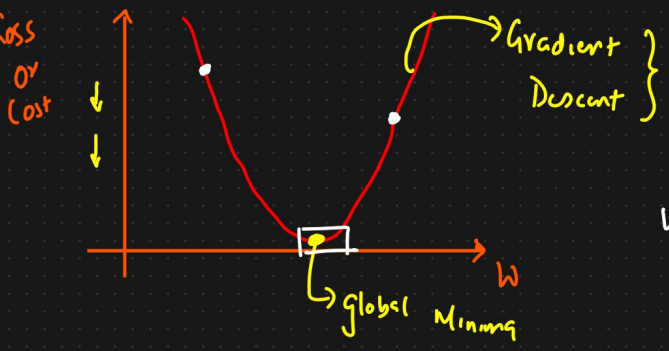
$$\text{loss function} = (y-\hat{y})^2$$

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

↳ Quadratic Equations




Weight updation




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

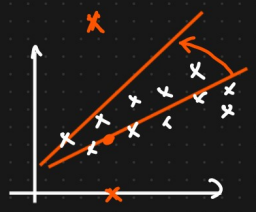
Slope

## Advantages

1. MSE is Differentiable
2. It has 1 local or global Minima
3. It converges faster

## Disadvantages

1. Not Robust to outliers



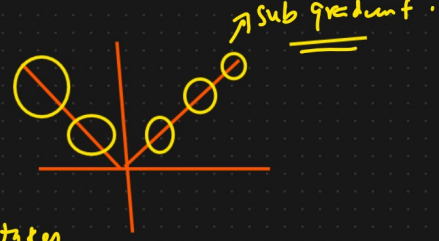
penalizing the Error

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

## 2. Mean Absolute Error (MAE)

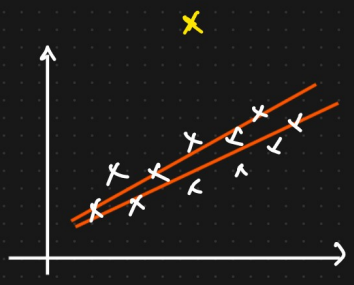
Loss fn : $|y - \hat{y}|$

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




## Advantages

1. Robust to outliers



## Disadvantages

1. Convergence usually takes time in MAE

## 3. Huber loss

1. MSE
2. MAE

Cost fn = $\begin{cases} \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \end{cases}$

MSE — No outlier — Hyperparameter

$$\begin{cases} \delta|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

⇓

MAE

## ④ RMSE (Root Mean Squared Error)

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

Advantages

Disadvantages

## ② Loss or Cost function For Classification Problems

Classification ⟶ Cross Entropy
- ⟶ Binary Cross Entropy {Binary}
- ⟶ Categorical Cross Entropy {Multiclass}
- ⟶ Sparse Categorical Cross Entropy
  ↳ {Multiclass}

### ① Binary Cross Entropy

Log Loss

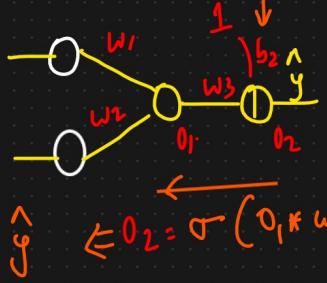
⇓

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

$y$ = Actual value

$\hat{y}$ = Predicted value

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




Binary classification

Sigmoid

$$\hat{y} = \frac{1}{1+e^{-z}}$$ ⇒ Sigmoid Activation function

## ② Categorical CROSS Entropy (Multiclass classification)

ONE → One Hot Encoding

| | $f_1$ | $f_2$ | $f_3$ | O/P | Good (j:1) | Bad (j=2) | Neutral (j=3) |
|---|---|---|---|---|---|---|---|
| → | 2 | 3 | 4 | Good | 1 | 0 | 0 |
| → | 5 | 6 | 7 | Bad | 0 | 1 | 0 |
| → | 8 | 9 | 10 | Neutral | 0 | 0 | 1 |

C = No of categories

i : 1 to n

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

Actual value ⇐ $y_{ij} = [\, y_{11} \;\; y_{12} \;\; y_{13} \cdots y_{1C} \,]$

$[\, y_{21}, \; y_{22}, \; y_{23} \cdots y_{2C} \,]$

$\vdots$

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

Prediction ⇐ $\hat{y}_{ij}$ ⇒ Softmax Activation $= Sof(z) = \dfrac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

↓ softmax Activation

O/p $\hat{y}_{ij}$ = Probabilities

[0.1, 0.2, 0.3, 0.2, 0.2] ← 1

Categorical Cross Entropy ⇒ [0.2, 0.3, 0.5]

[This also gives the probability of other categories]

## ③ Sparse Categorical Cross Entropy

0 1st 2nd → Categories

[0.2, 0.3, 0.5]

2nd Index ⇒ O/p

0 1 2nd 3rd 4th

[0.2, 0.3, 0.1, 0.2, 0.2] ⇒ 1th Index ⇒ Category ⇒ O/p

**Disadvantage**

① Loosing info about the probability of other category.

## ④ Right Combination

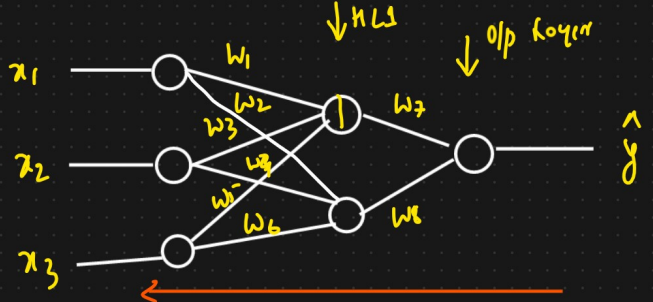
Activation applied ⇓

| | Hidden Layers | O/p Layer | Problem Statement | Loss function |
|---|---|---|---|---|
| ① | Relu or its Variants | Sigmoid | Binary classification | Binary Cross Entropy |
| ② | Relu or its Variants | Softmax | Multi class | Categorical or Sparse CE |
| ③ | Relu or its Variants | linear | Regression | MSE, MAE, Huber loss, RMSE |

# Optimizers

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Mini batch SGD
4. SGD With Momentum
5. Adagrad and RMSPROP
6. Adam Optimizers

## 1. Gradient Descent Optimizer



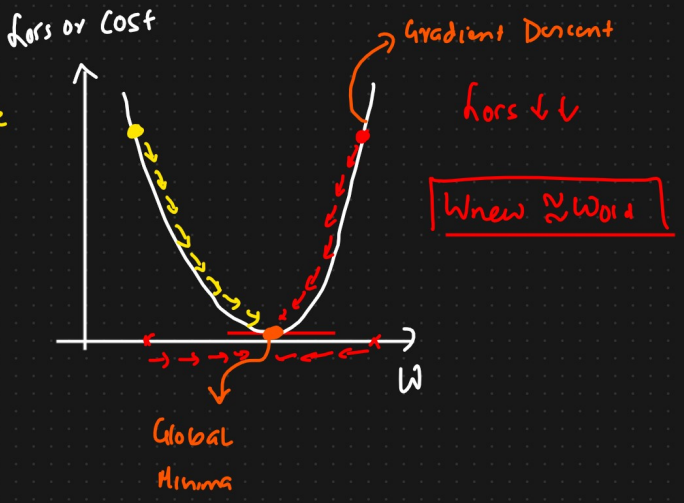

Backward Propogation

$$Loss = [\quad] \downarrow\downarrow\downarrow$$

Optimizers

Learning Rate

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




MSE

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

Dataset = 1000 Data points

## Epochs, Iteration

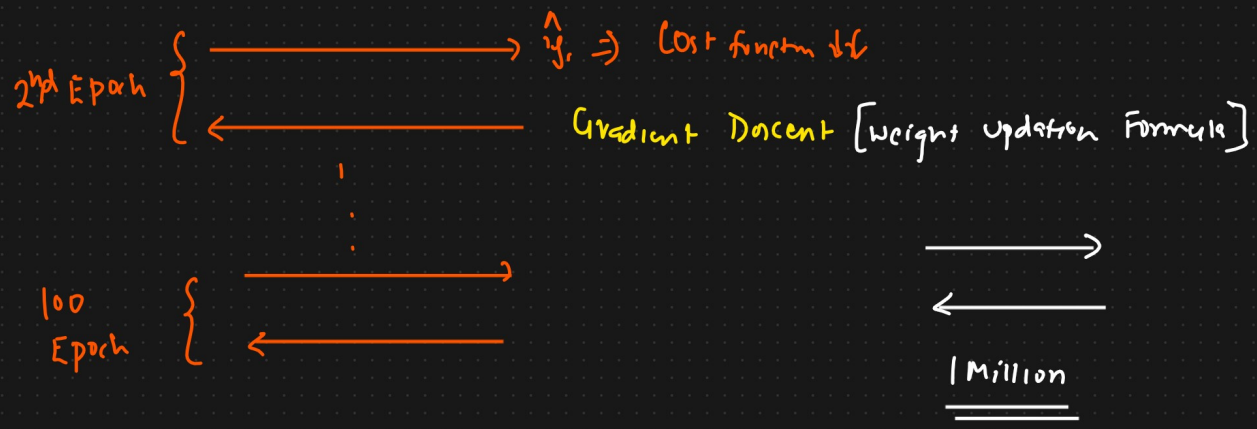
1 Epoch = 1 Iteration

10 Iteration

$100/10 = 100$

1 Epoch { 1000 datapoint → $\hat{y}_i$ ⇒ Cost function ↓↓ ; Weights will get updated ← } Gradient Descent optimizer [Weight updation]

100




## Advantages

① Convergence will happen.

## Disadvantage

① Huge Resource RAM, GPU

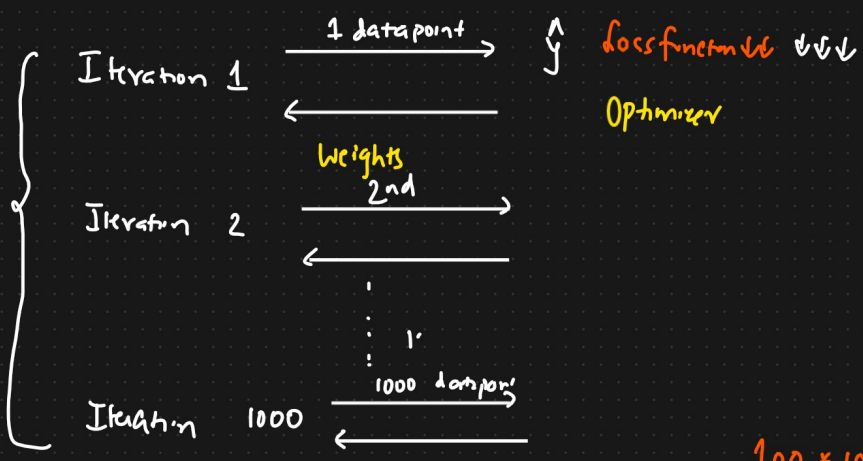
⇓

Resource Intensive.

## ② Stochastic Gradient Descent (SGD)

1000 datapoint

Epochs & Iteration



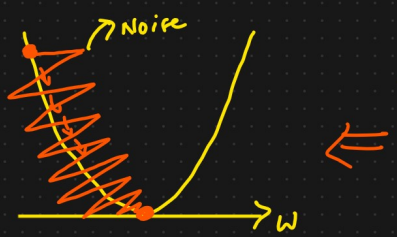

100 × 1000 = 100000

Epochs

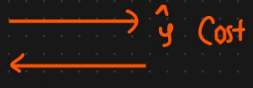
100000 datapoint

## Advantage

Ⓐ Solve Resource Issue




## Disadvantage




Ⓐ Time Complexity ↑↑

Ⓐ Convergence will also take more time.

Ⓐ Noise gets Introduced

# ③ Mini Batch SGD

No. of iterations = $\frac{100000}{1000}$ = 100 iterations

Epoch, Iteration, Batch_size

Datapoints = 100000

batch_size = 1000

MSE

Cost fn = $\sum_{i=1}^{1000} (y_i - \hat{y}_i)^2$ ↓↓

Epoch 1

Iteration 1 — 1000 →

← Optimizer ⇒ Mini Batch SGD

change the weights

Iteration 2 — 1000 →

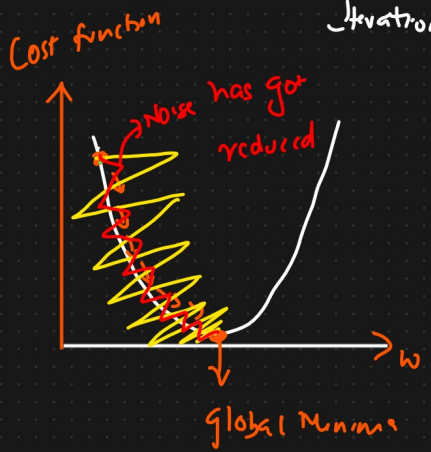
↓↓

| | |
|---|---|
| 8gb | 1 |
| 16gb | 5000 |

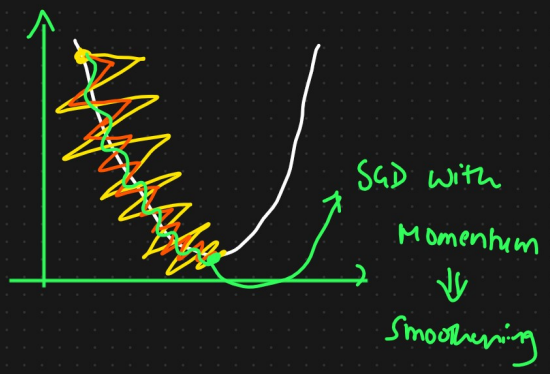
Iteration 3

!

Iteration 100




Batch Size




## Advantages

① Convergence speed will increase =

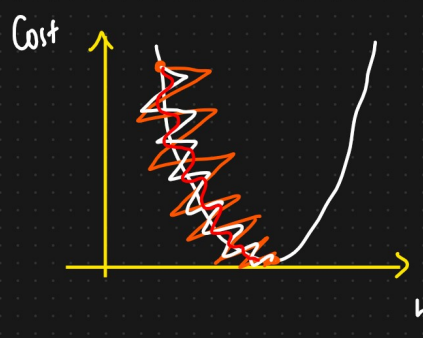
② Noise will be less when compared to SGD

③ Efficient Resource Usage (RAM)

## Disadvantage

① Noise still exists

## ④ SGD With Momentum




Weight Updation formula

learning rate

$$W_{new} = W_{old} - \eta \frac{\partial h}{\partial W_{old}}$$

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

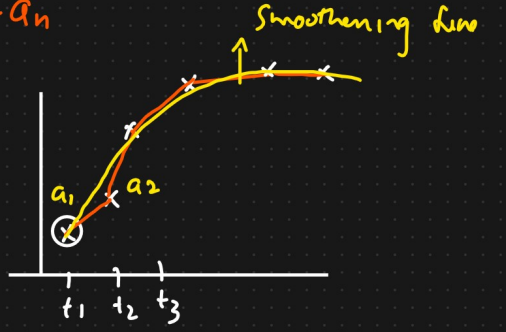
$$W_t = W_{t-1} - \eta \frac{\partial h}{\partial W_{t-1}}$$

Exponential Weight Average. {Smoothening} ⇒ ARIMA, SARIMAX

Time = $t_1$ $t_2$ $t_3$ $t_4$ - - - $t_n$

Values = $a_1$ $a_2$ $a_3$ $a_4$ - - - - $a_n$

Time Series




$$V_{t_1} = a_1$$

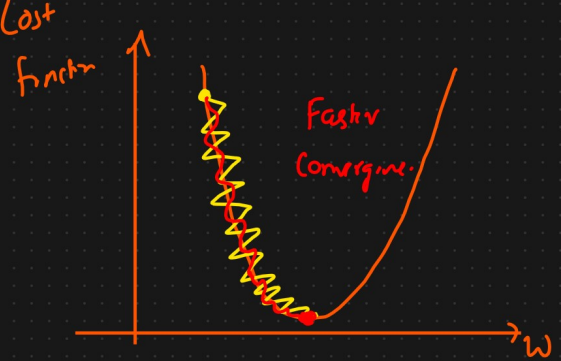
$$V_{t_2} = \boxed{\beta} * V_{t_1} + (1-\beta) * a_2$$

$\beta = 0$ to $1$

$$= 0.95 * a_1 + (0.05)\, a_2$$

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

$$= 0.95 \left[ 0.95 * a_1 + (0.05)\, a_2 \right] + (0.05) * a_3$$




Advantage

① Reduces the noise

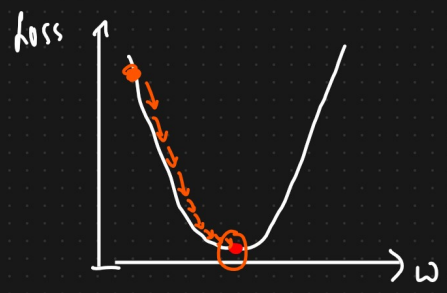
② Quick Convergence

## ⑤ Adagrad : Adaptive Gradient Descent

$\eta$ = fixed ⇒ Dynamic learning Rate

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

→ Learning Rate = 0.001




As the convergence happen the learning rate should change

$0.00001 \approx 0$

$$W_t = W_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}$$

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

$\eta' \downarrow\downarrow$ $\alpha_t \uparrow\uparrow$

$\epsilon$ → Epsilon ≈ small value

$\alpha_t \uparrow\uparrow$

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

$W_t \approx W_{t-1}$

$t=1 \rightarrow t=2 \rightarrow t=3$

$\eta = 0.01$ $\eta = 0.005$ $\eta = 0.003$

### Disadvantage

① $\eta'$ → Possibility to become a very small value ≈ 0

## ⑥ Adadelta And RMSPROP

Exponential Weight Average

$\beta = 0.95$ $\quad Sdw_t = 0$

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

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

[Dynamic LR + Smoothening EWA]

$$\boxed{W_t = W_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}}$$

## ⑦ Adam Optimizer

SGD With Momentum + RMSPROP [Dynamic LR + Smoothening]

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

Weight Updation

$$b_t = b_{t-1} - \eta' V db$$

→ Bias Updation

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

$Sdw_t = 0$

EWA

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

[EWA]

$$Vdw_t = \beta * Vdw_{t-1} + (1-\beta)\frac{\partial h}{\partial w_{t-1}}$$

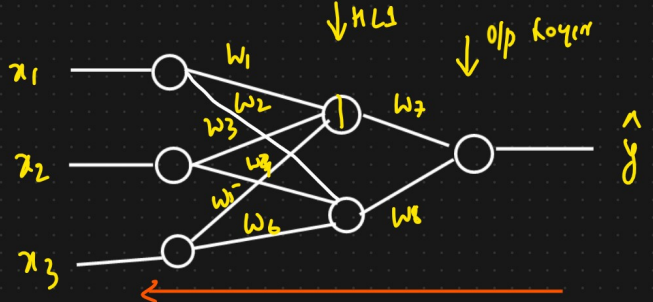
$$Vdb_t = \beta * Vdb_{t-1} + (1-\beta)\frac{\partial h}{\partial b_{t-1}}$$

⇒ Momentum
↳ Smoothening

# Optimizers

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Mini batch SGD
4. SGD With Momentum
5. Adagrad and RMSPROP
6. Adam Optimizers

## 1. Gradient Descent Optimizer



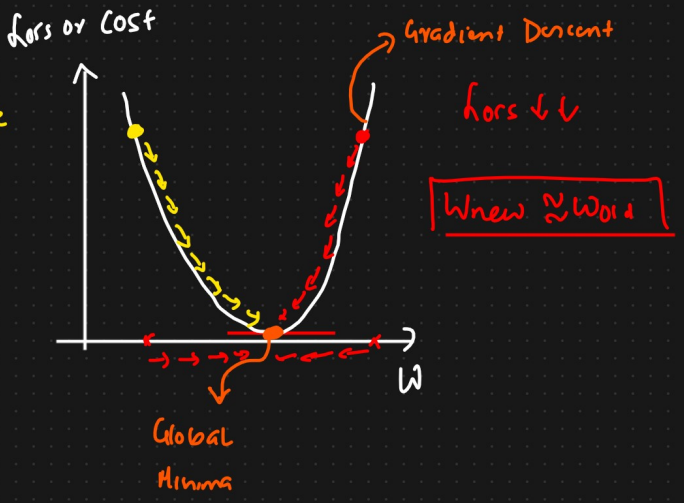

Backward Propogation

$$Loss = [\quad] \downarrow\downarrow\downarrow$$

Optimizers

Learning Rate

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




MSE

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

Dataset = 1000 Data points

## Epochs, Iteration

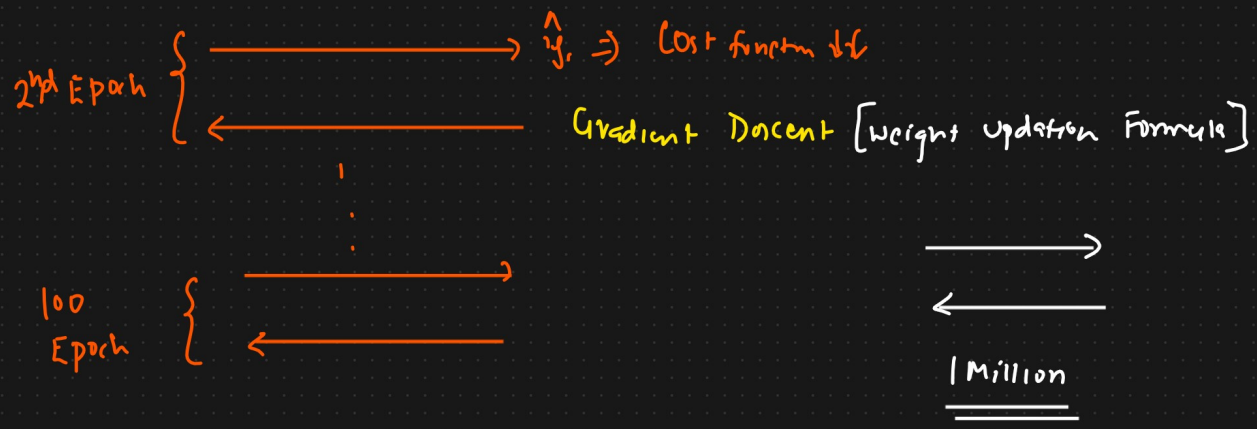
1 Epoch = 1 Iteration

10 Iteration

$100/10 = 100$

1 Epoch { 1000 datapoint → $\hat{y}_i$ ⇒ Cost function ↓↓ ; Weights will get updated ← } Gradient Descent optimizer [Weight updation]

100




## Advantages

① Convergence will happen.

## Disadvantage

① Huge Resource RAM, GPU

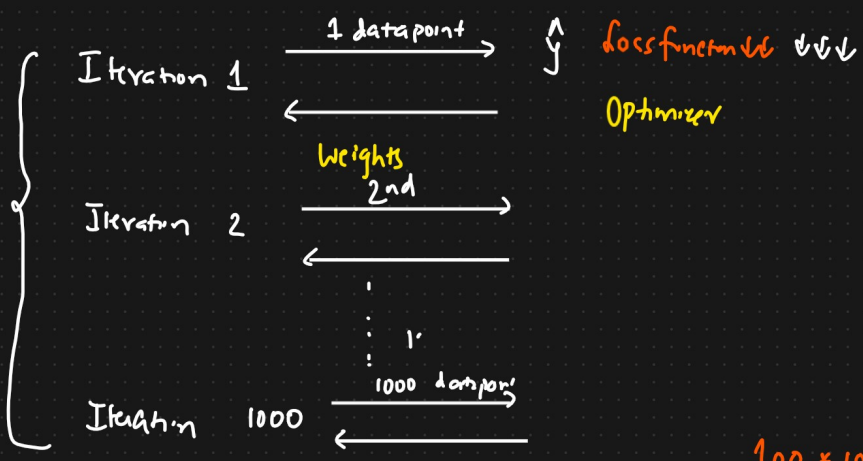
⇓

Resource Intensive.

## ② Stochastic Gradient Descent (SGD)

1000 datapoint

Epochs & Iteration



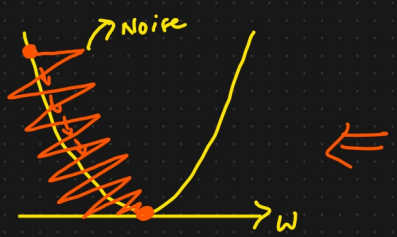

100 × 1000 = 100000

Epochs

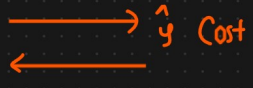
100000 datapoint

## Advantage

Ⓐ Solve Resource Issue




## Disadvantage




Ⓐ Time Complexity ↑↑

Ⓐ Convergence will also take more time.

Ⓐ Noise gets Introduced

# ③ Mini Batch SGD

No. of iterations = $\frac{100000}{1000}$ = 100 iterations

Epoch, Iteration, Batch_size

Datapoints = 100000

batch_size = 1000

MSE

Cost fn = $\sum_{i=1}^{1000} (y_i - \hat{y}_i)^2$ ↓↓

Epoch 1

Iteration 1 — 1000 →

← Optimizer ⇒ Mini Batch SGD

change the weights

Iteration 2 — 1000 →

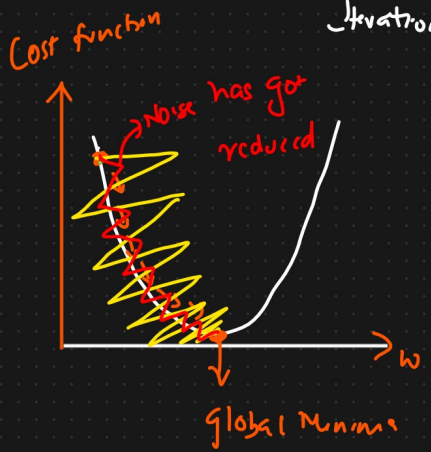
↓↓

| | |
|---|---|
| 8gb | 1 |
| 16gb | 5000 |

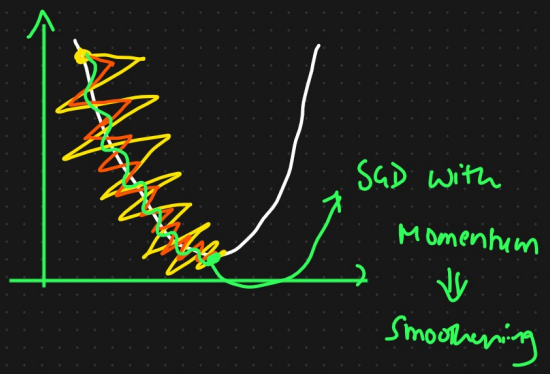
Iteration 3

!

Iteration 100




Batch Size




## Advantages

① Convergence speed will increase =

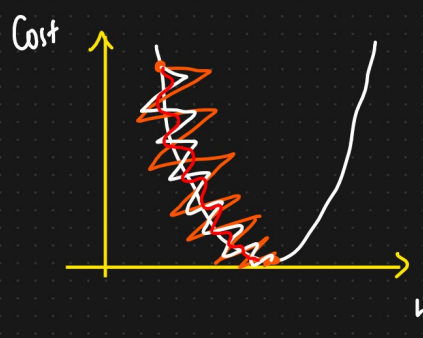
② Noise will be less when compared to SGD

③ Efficient Resource Usage (RAM)

## Disadvantage

① Noise still exists

## ④ SGD With Momentum




Weight Updation formula

learning rate

$$W_{new} = W_{old} - \eta \frac{\partial h}{\partial W_{old}}$$

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

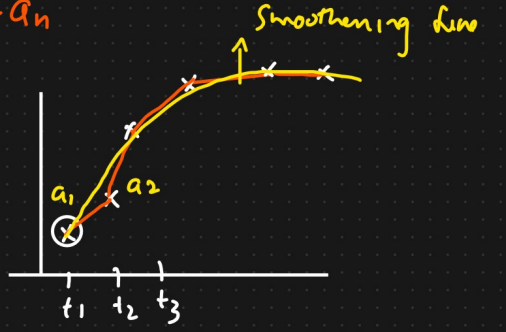
$$W_t = W_{t-1} - \eta \frac{\partial h}{\partial W_{t-1}}$$

Exponential Weight Average. {Smoothening} ⇒ ARIMA, SARIMAX

Time = $t_1$ $t_2$ $t_3$ $t_4$ - - - $t_n$

Values = $a_1$ $a_2$ $a_3$ $a_4$ - - - - $a_n$

Time Series




$$V_{t_1} = a_1$$

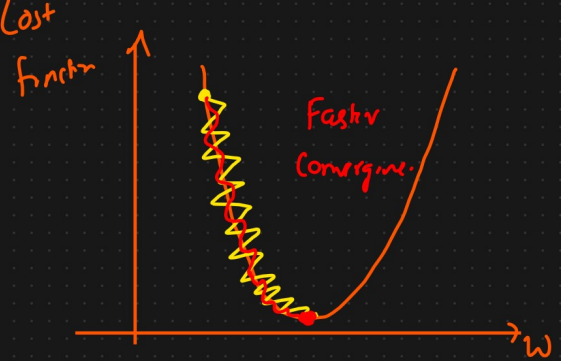
$$V_{t_2} = \boxed{\beta} * V_{t_1} + (1-\beta) * a_2$$

$\beta = 0$ to $1$

$$= 0.95 * a_1 + (0.05)\, a_2$$

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

$$= 0.95 \left[ 0.95 * a_1 + (0.05)\, a_2 \right] + (0.05) * a_3$$




Advantage

① Reduces the noise

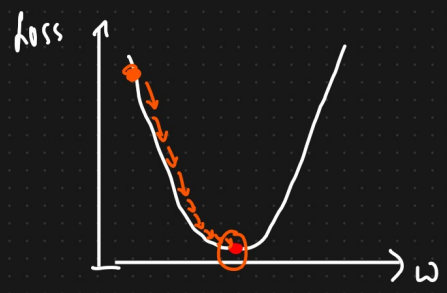
② Quick Convergence

## ⑤ Adagrad : Adaptive Gradient Descent

$\eta$ = fixed ⇒ Dynamic learning Rate

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

→ Learning Rate = 0.001




As the convergence happen the learning rate should change

$0.00001 \approx 0$

$$W_t = W_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}$$

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

$\eta' \downarrow\downarrow$ $\alpha_t \uparrow\uparrow$

$\epsilon$ → Epsilon ≈ small value

$\alpha_t \uparrow\uparrow$

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

$W_t \approx W_{t-1}$

$t=1 \rightarrow t=2 \rightarrow t=3$

$\eta = 0.01$ $\eta = 0.005$ $\eta = 0.003$

### Disadvantage

① $\eta'$ → Possibility to become a very small value ≈ 0

## ⑥ Adadelta And RMSPROP

Exponential Weight Average

$\beta = 0.95$ $\quad Sdw_t = 0$

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

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

[Dynamic LR + Smoothening EWA]

$$\boxed{W_t = W_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}}$$

## ⑦ Adam Optimizer

SGD With Momentum + RMSPROP [Dynamic LR + Smoothening]

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

Weight Updation

$$b_t = b_{t-1} - \eta' V db$$

→ Bias Updation

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

$Sdw_t = 0$

EWA

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

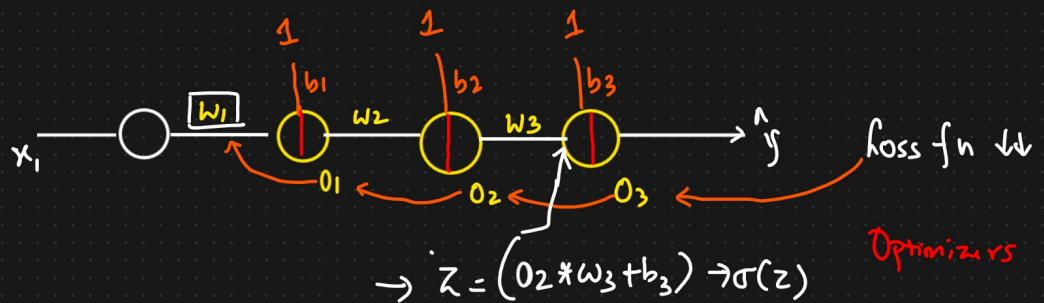
[EWA]

$$Vdw_t = \beta * Vdw_{t-1} + (1-\beta)\frac{\partial h}{\partial w_{t-1}}$$

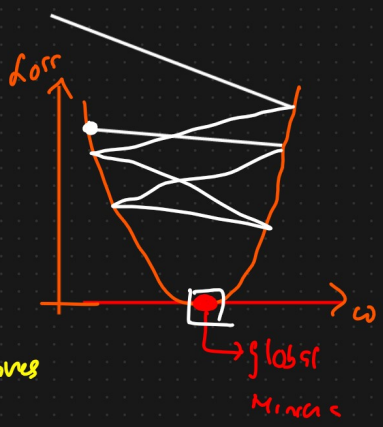
$$Vdb_t = \beta * Vdb_{t-1} + (1-\beta)\frac{\partial h}{\partial b_{t-1}}$$

⇒ Momentum
↳ Smoothening

# Exploding Gradient Problem ⇒ Weight Initialization Technique




Optimizers




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

$\{ W_{new} >>>> W_{old} \quad W_{new} <<<< W_{old} \}$

Chain Rule of Derivatives

$$\frac{\partial h}{\partial W_{old}} = \frac{\partial h}{\partial O_3} * \boxed{\frac{\partial O_3}{\partial O_2}} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial w_{old}}$$

big * big * big * big ⇒ big value

Weight initialization ⇓ Very high value

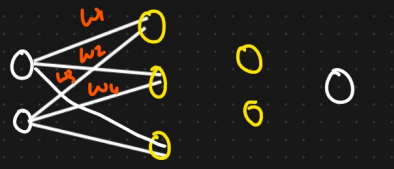
$$\frac{\partial O_3}{\partial O_2} = \boxed{\frac{\partial \sigma(z)}{\partial z}} * \frac{\partial z}{\partial O_2}$$

$$= [0 - 0.25] * \frac{\partial (O_2 * w_3 + b_3)}{\partial O_2}$$

$$= [0 - 0.25] * w_3 \Rightarrow 500 - 1000$$

## Weight Initializing Techniques

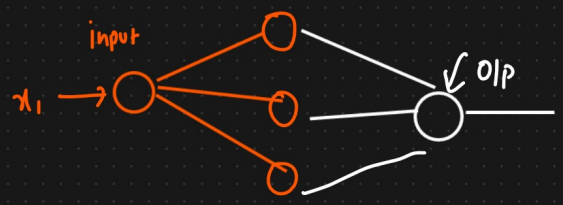
1. Uniform Distribution ✓
2. Xavier / Glorot Initialization ✓
3. Kaiming He Initialization ✓




## Key Points

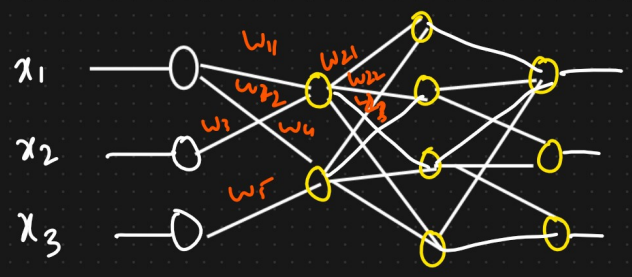
1. Weights should be small ✓
2. Weights should not be same ✓
3. Weights should have good variance

} ✓




input = 1

Output = 1




input = 3

O/p = 3

## ① Uniform Distribution

$W_{ij} \sim$ Uniform Distribution $\left[\frac{-1}{\sqrt{input}}, \frac{1}{\sqrt{input}}\right]$

$\left[\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right]$

## ② Xavier/Glorot Initialization

Researcher $\longrightarrow$ Xavier Glorot

### ① Xavier Normal Init

$W_{ij} \sim N(0, \sigma)$

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

### ② Xavier Uniform

$W_{ij} \sim$ Uniform Distribution $\left[\frac{-\sqrt{6}}{\sqrt{input + output}}, \frac{\sqrt{6}}{\sqrt{i/p + o/p}}\right]$

## ③ Kaiming HC Initialization

### ① He Normal

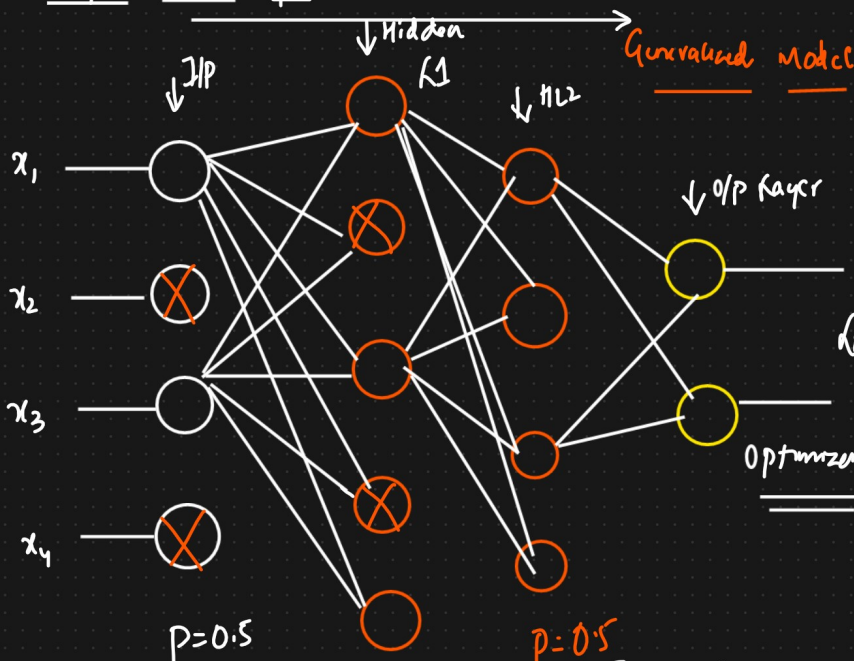
$$W_{ij} \sim N(0, \sigma)$$

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

### ② He uniform

$$W_{ij} \sim \text{Uniform Dimbution} \left[ -\sqrt{\frac{6}{input}}, \sqrt{\frac{6}{input}} \right]$$

# Drop Out Layer




Generalized Model ∈ ⇒ Overfitting

Model → Training Acc ⇒ 90%
Test Acc ⇒ 60%

Optimizers

Machine Learning

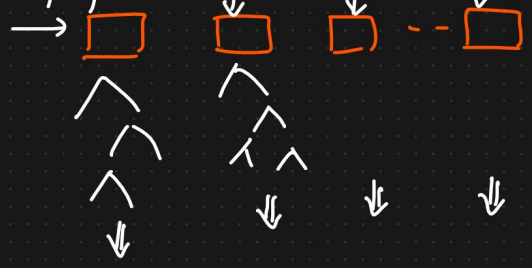
DT

⇒ Overfitting

Generalized Model

Random Forest

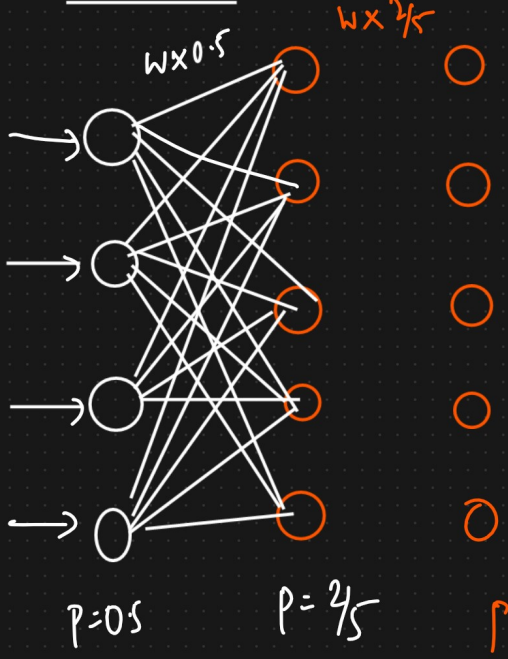
Feature Sampling
Row Sampling



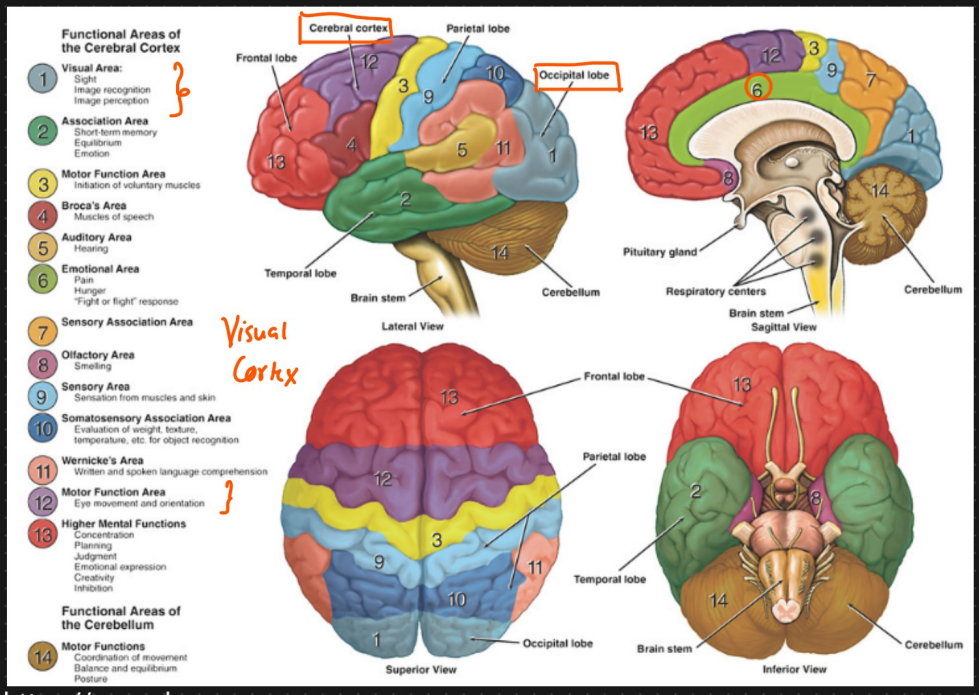
$0 \leq p \leq 1$

Hyperparameter

Test data



# Convolutional Neural N/w




https://www.dana.org

① ANN → Supervised Learning → [Classification, Regression]

Dataset : I/p features O/p

② CNN : I/p => Images Eg: Image classification, Object Detection, Segmentation

## ② Cerebral Cortex And Visual Cortex

Visual Cortex (V1-V5) [Region of the brain that receives, integrates and processes visual information relayed from the retinas].

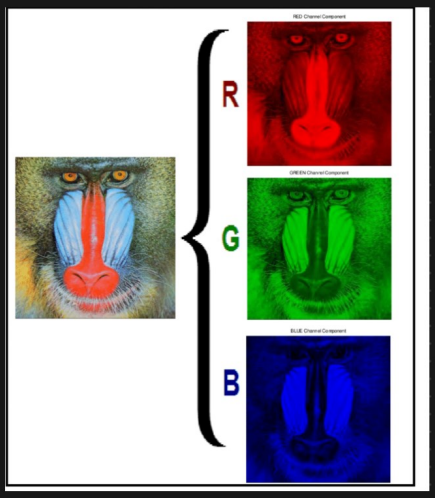
→ V1 → Primary Visual Cortex [Orientation of edges And Lines]

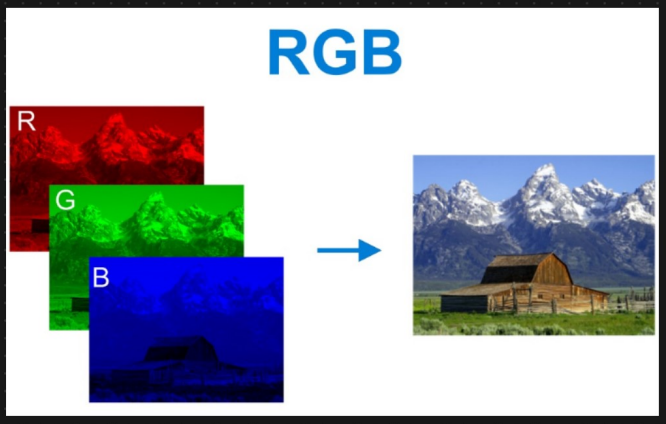
V2 [Differences in color, complex patterns, Object orientation]

V3 V4 V5 [Object Recognition]

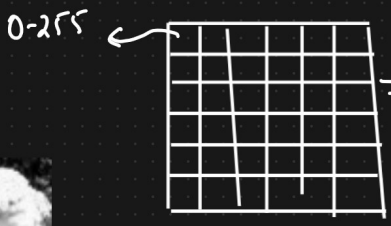
Visualize the Image

## ③ RGB Images And Gray Scale Images

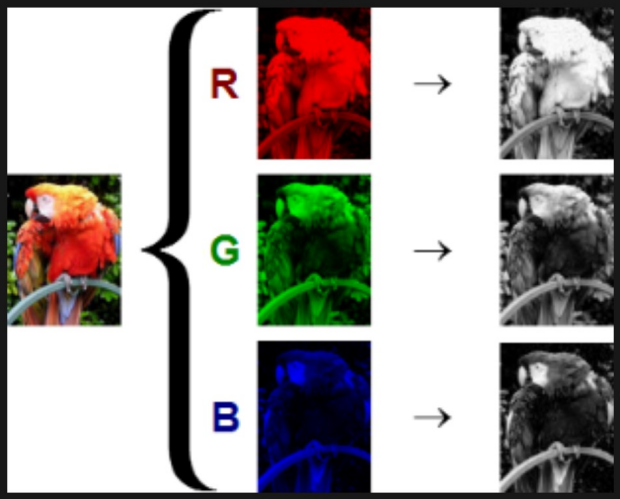


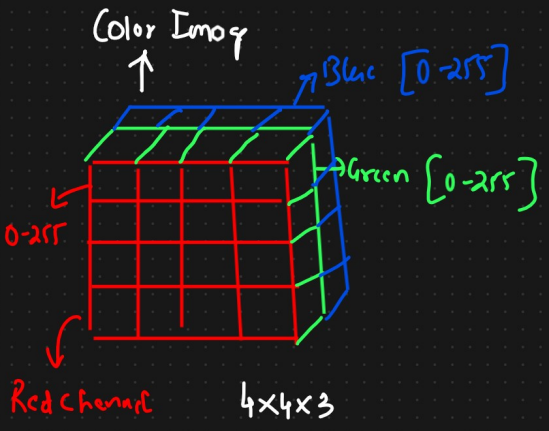

https://www.researchgate.net/




6x6x1




https://commons.wikimedia.org/



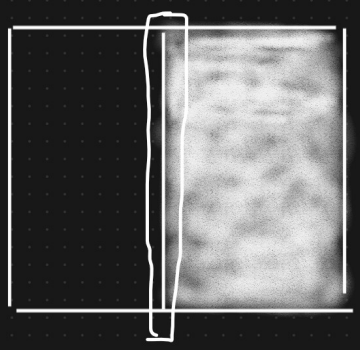

4x4x3

## ④ Convolution Operation In CNN

→ (0,1)

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |

6x6x1



Step 1

(1) Normalize

Divide by 255

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

Convolution operation

Stride=1

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |

6 x 6 x 1

n = 6

*

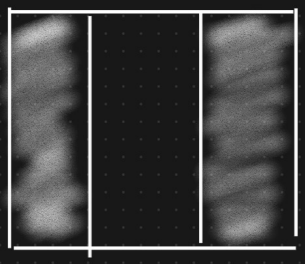
f = 3

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

3 x 3

filters

Vertical edge filters



⇐

| | | | |
|---|---|---|---|
| 2rr | 0 | 0 | 2rr |
| 2rr | 0 | 0 | 2rr |
| 2rr | 0 | 0 | 2rr |
| 2rr | 0 | 0 | 2rr |

=

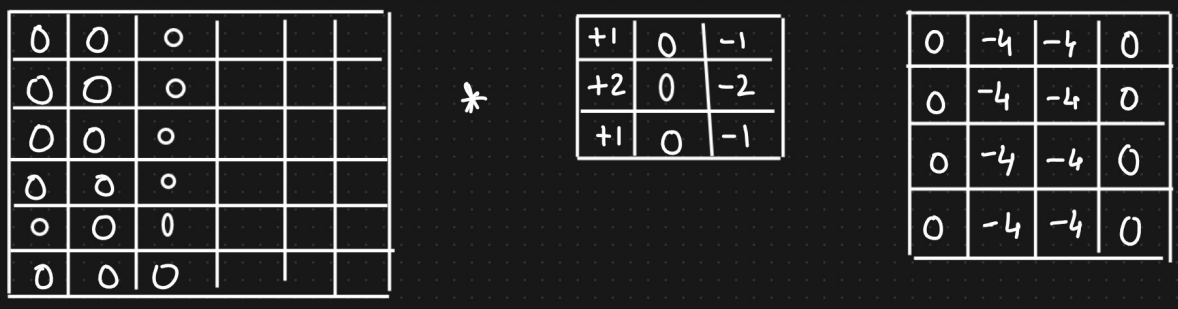
| | | | |
|---|---|---|---|
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |

4 x 4

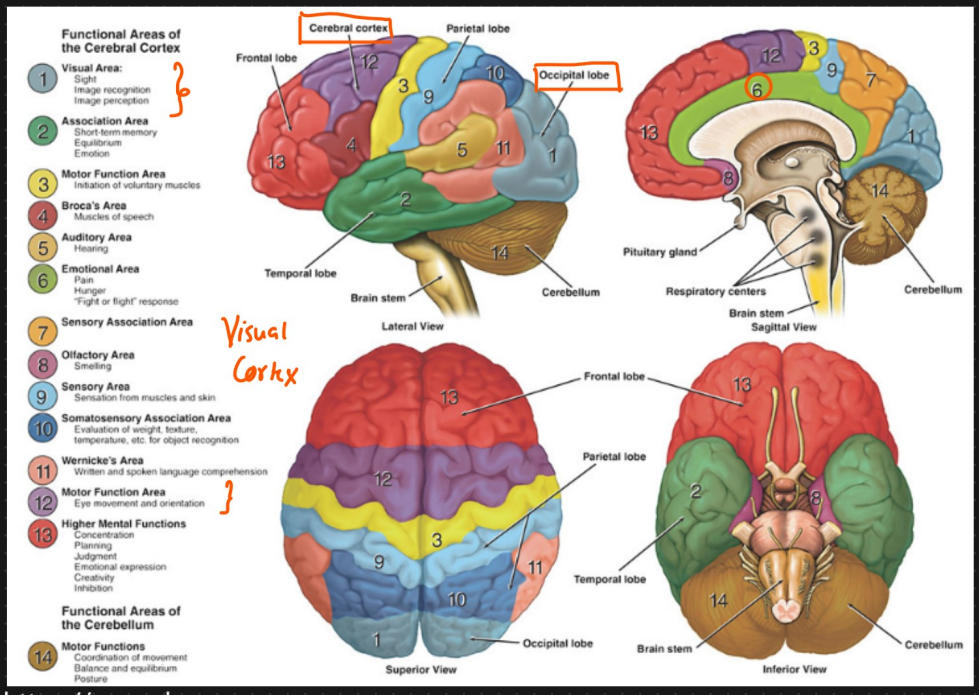
O/p = 4

$n - f + 1 =$

$= 6 - 3 + 1 = 4$



# Convolutional Neural N/w




https://www.dana.org

① ANN → Supervised Learning → [Classification, Regression]

Dataset : I/p features O/p

② CNN : I/p => Images Eg: Image classification, Object Detection, Segmentation

## ② Cerebral Cortex And Visual Cortex

Visual Cortex (V1-V5) [Region of the brain that receives, integrates and processes visual information relayed from the retinas].

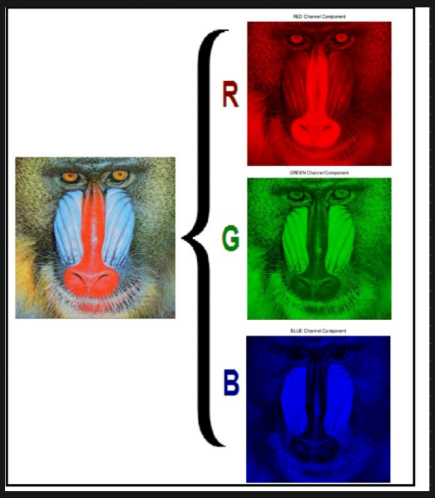
→ V1 → Primary Visual Cortex [Orientation of edges And Lines]

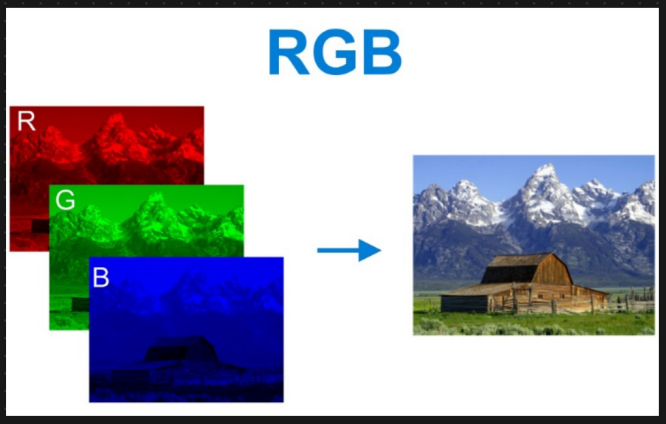
V2 [Differences in color, complex patterns, Object orientation]

V3 V4 V5 [Object Recognition]

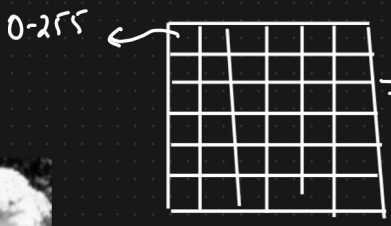
Visualize the Image

## ③ RGB Images And Gray Scale Images

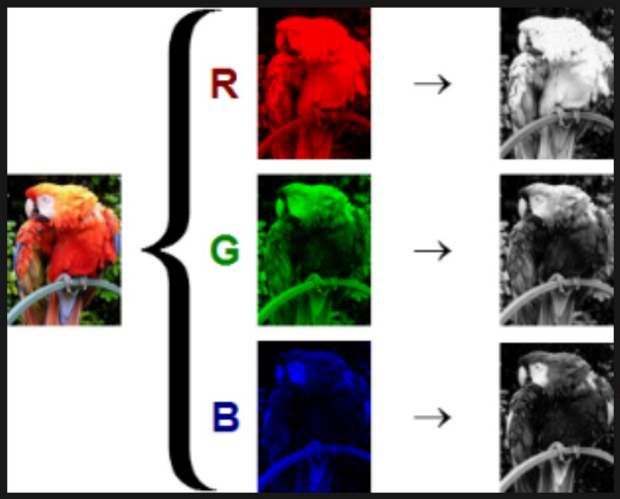


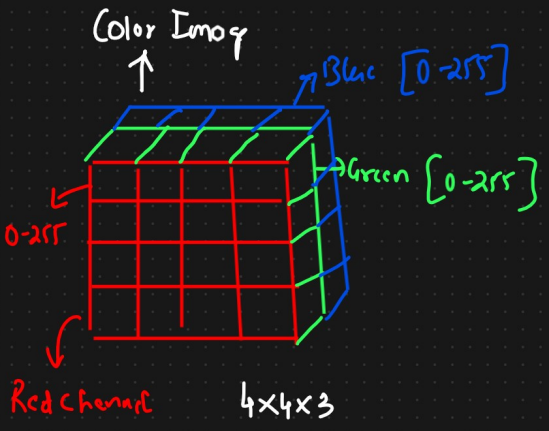

https://www.researchgate.net/




6x6x1




https://commons.wikimedia.org/



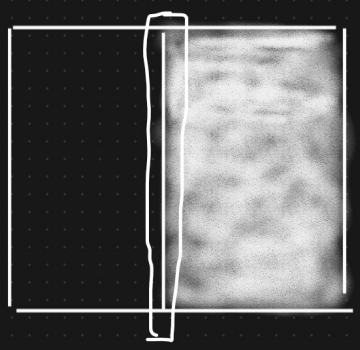

4x4x3

## ④ Convolution Operation In CNN

→ (0,1)

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |
| 0 | 0 | 0 | 255 | 255 | 255 |

6x6x1



Step 1

(1) Normalize

Divide by 255

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

Convolution operation

Stride=1

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |

6 x 6 x 1

n = 6

*

f = 3

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

3 x 3

filters

Vertical edge filters

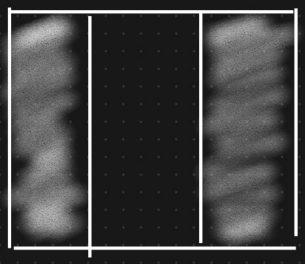
=

o/p = 4

| | | | |
|---|---|---|---|
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |

4 x 4

$n - f + 1 = 6 - 3 + 1 = 4$



⇐

| | | | |
|---|---|---|---|
| 2rr | 0 | 0 | 2rr |
| 2rr | 0 | 0 | 2rr |
| 2rr | 0 | 0 | 2rr |
| 2rr | 0 | 0 | 2rr |

## ⑤ Padding In CNN

① Zero padding

② Neighbour padding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 0 | | | | | | | |

6x6 n=6 8x8

*

f = 3

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

=

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 0 | -4 | -4 | 0 |
| | 0 | -4 | -4 | 0 |
| | 0 | -4 | -4 | 0 |
| | 0 | -4 | -4 | 0 |
| | | | | |

6x6

$n - f + 2p + 1$

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

$3 + 2p + 1 = 6$

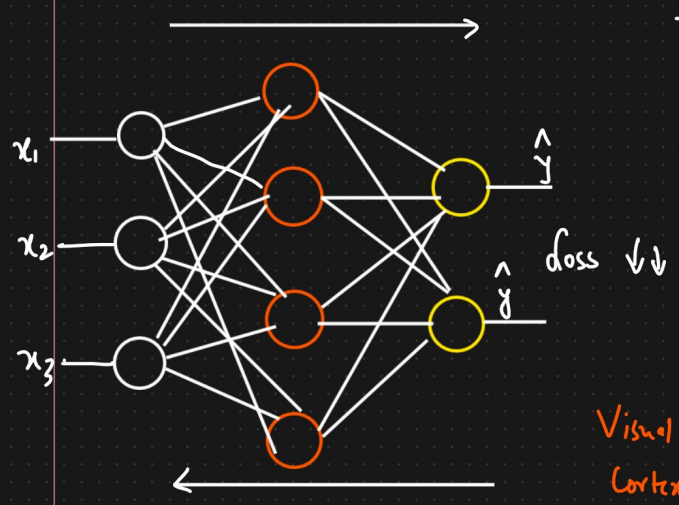
$2p = 6 - 4$

$p = 2/2 = 1$

7x7 3x3 7x7

How much padding you need to apply?

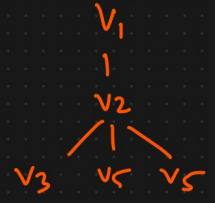
## ⑥ Operation of CNN Vs ANN




$$z = w^T x_i + b$$

$$relu(z)$$

Visual Cortex




Image

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |

$*$

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

$f_1$

$f_2$

$f_3$

$f_n$

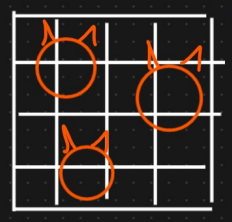
$=$

→ Relu operation $max(0, x)$

| | | | |
|---|---|---|---|
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |

Convolution Layer

## ⑦ Max Pooling, Min Pooling, Mean Pooling



$\circledast$

2x2

P=1

S=1

filter

$f_2$

$f_3$

$f_n$

Convolution Layer

$=$

Relu

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 3 | 6 |
| 2 | 8 | 4 |

3x3

Stride = 2

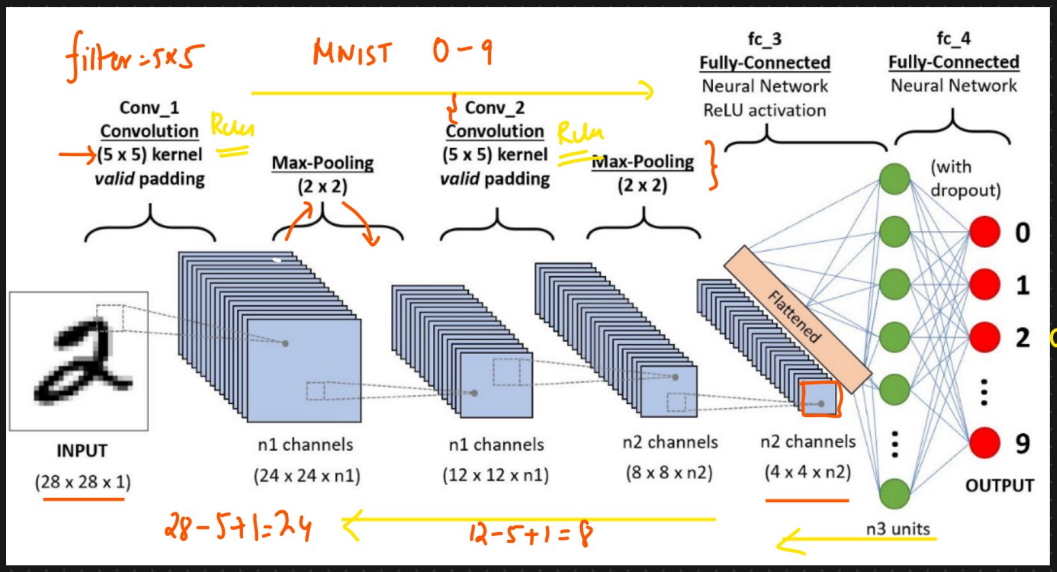
Max Pooling

2x2

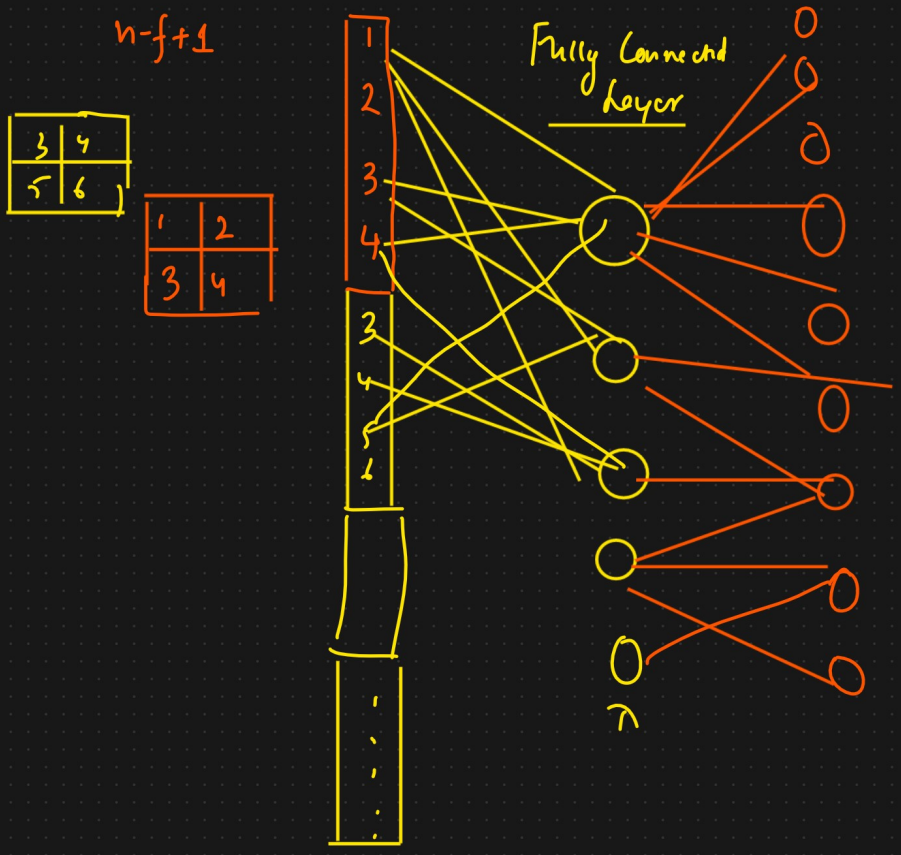
| | |
|---|---|
| 4 | 6 |
| 8 | 4 |

⑦ Location Invariant

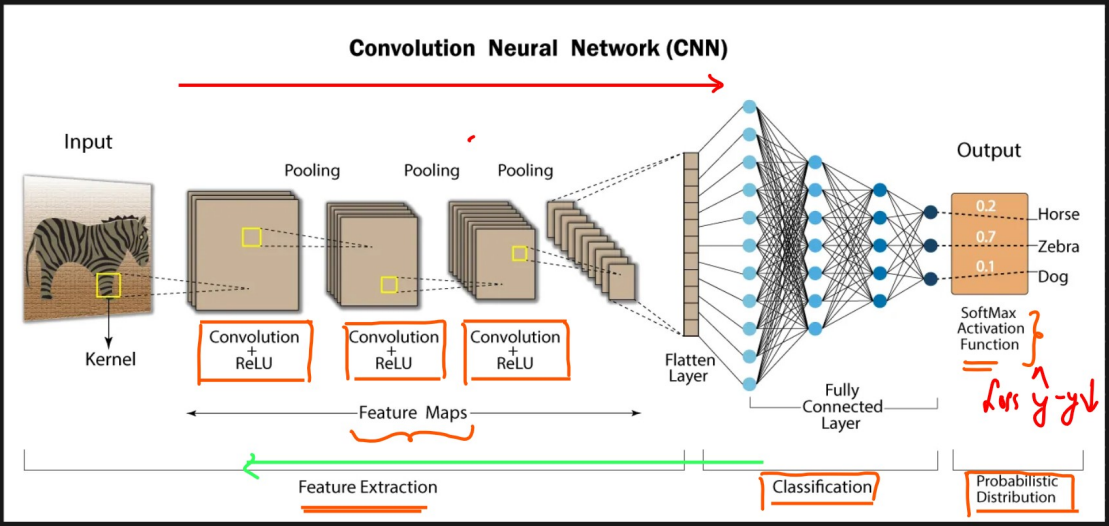
⑧ Fully Connected Layer In CNN [Flattened Layer]




Loss function ↓↓
Optimizer



# ⑨ CNN Complete Example




https://developersbreach.com/convolution-neural-network-deep-learning/

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | | | |
| 0 | 0 | 0 | | | |
| 0 | 0 | 0 | | | |
| 0 | 0 | 0 | | | |
| 0 | 0 | 0 | | | |
| 0 | 0 | 0 | | | |

*

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

| | | | |
|---|---|---|---|
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |
| 0 | -4 | -4 | 0 |