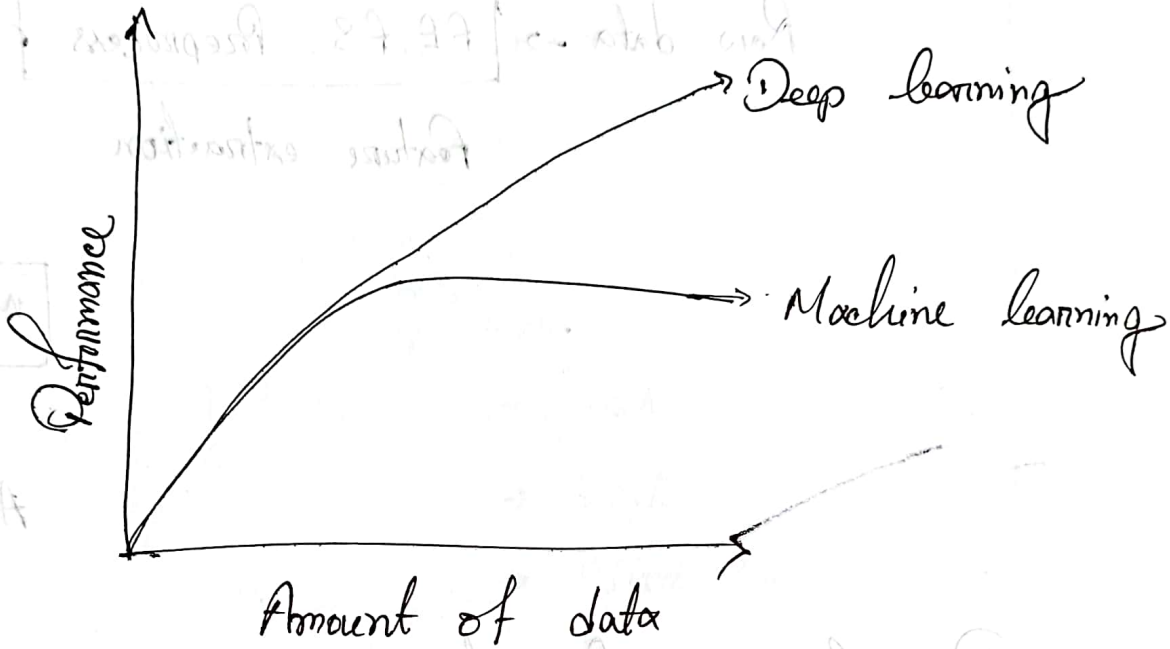
why Deep learning



Exponential growth of data

↳ Deep learning model.

Accuracy ↑

→ Technology Upgradation.

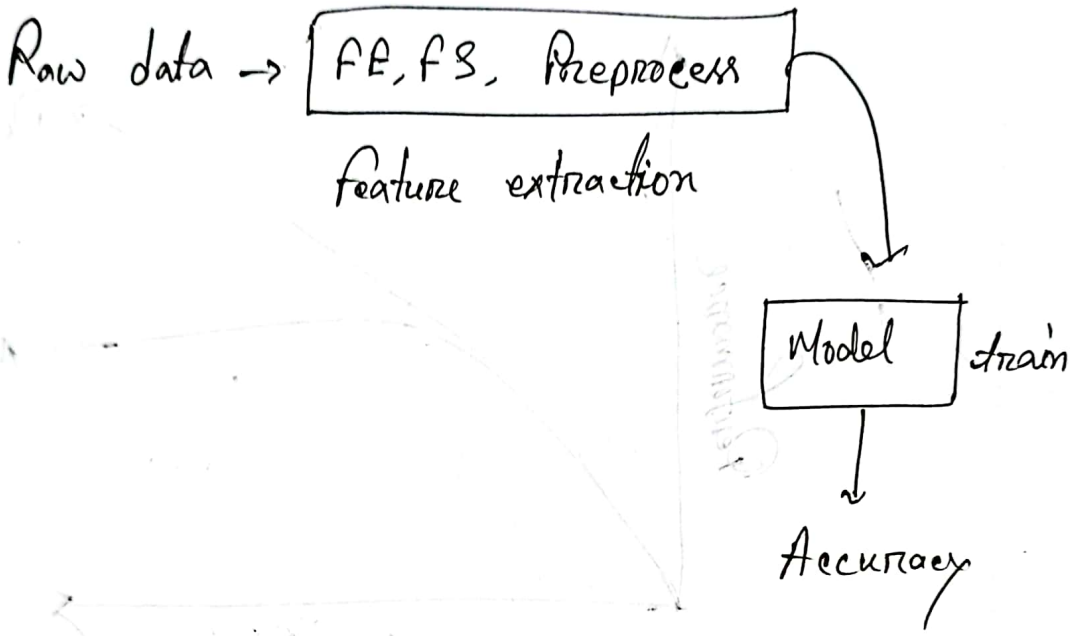
cheap Hardware

NVIDIA → GPU Price ↓

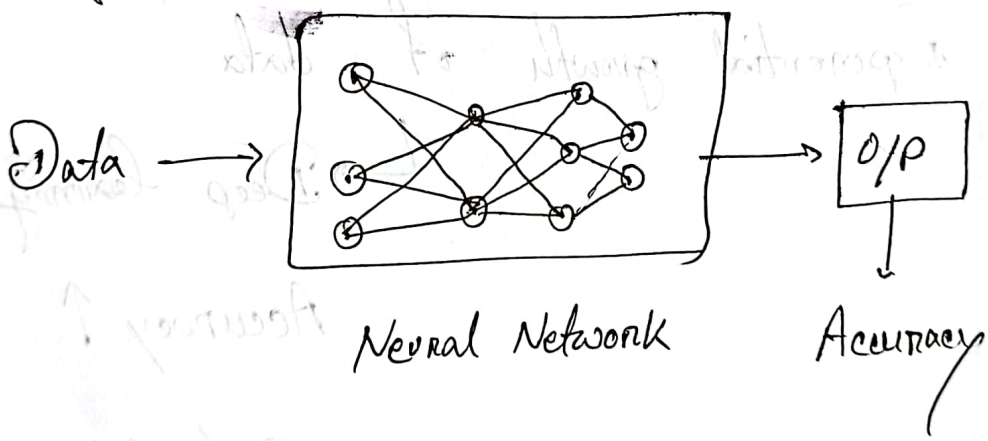
↓

used in cloud.

ML Project



Deep learning Project



-> Deep learning can solve complex problem statement

- -> NLP
- -> Image classification
- -> Speech Recognation.

Deep learning Roadmap

① Introduction to Neural Network.

- ⇒ loss function
  - ⇒ Optimizer
    - → Gradient decent
    - → SGD
    - → Mini-batch GD
    - → Adagrad
    - → RMSProp
    - → Adam
      - ⇒ Activation function
        - → Sigmoid
        - → Relu

→ Global minimal

→ local minimal

→ Convergence point.

② Artificial Neural Network (An like ML)

↳ Deployment project.

- Heroiko
- M Azure
- Flask.

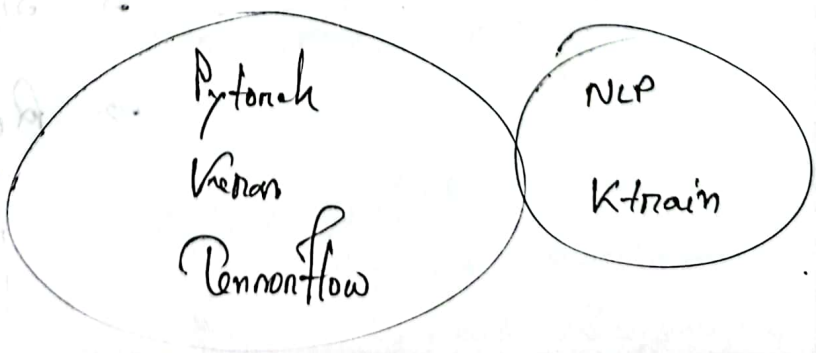
→ weight initialize

→ Parameters

→ Hyperparameter tuning

→ Hidden layer

→ Neuron.



(13) CNN. Convolutional Neural Network.

⇒ 1 in dm done here..

⇒ How does the convolution Actually work.

⇒ Input: Images, videos, Sound.

⇒ Transfer Learning. ResNet, Alexnet, vgg 16.

⇒ Object Detection.

⇒ RCNN, Marked RCNN, SSD, YOLO

⇒ Deploy Project.

NLP Part

④. RNN → Recurrent Neural Network.

⇒ when our input data Sequence to sequence

⇒ LSTM, Bidirectional LSTM

⇒ GRU

⇒ Word Embeding

⇒ Word 2 Vec.

⇒ Attention Models, Encoder, decoder.

Auto encoder.

⇒ Transfer. (NLP)

⇒ BERT (NLP)

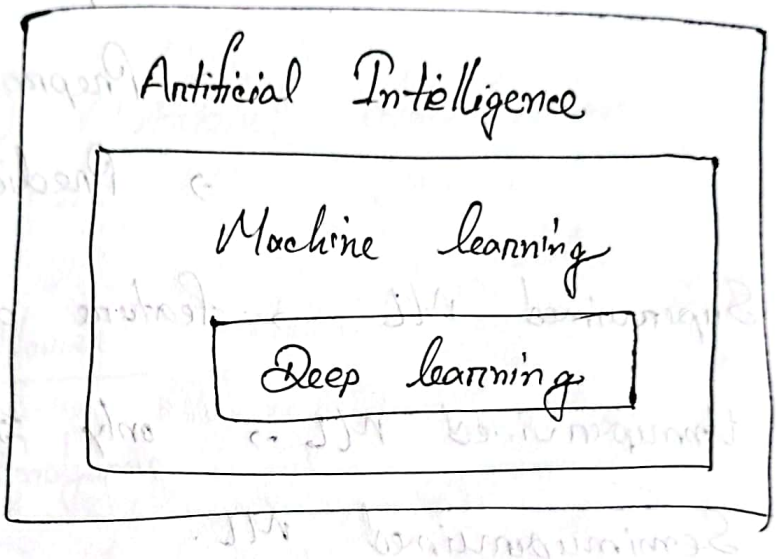
⇒ Hugging face, Ktrain (library)

AI vs ML vs DL vs DS ?

Computer Vision. NLP

AI Application

- Netflix
- Self driving
- Amazon
- Youtube.



- Netflix Recommendation is a part of ML, DL.
- In AI Application not everything in AI

  ↓

  Trying to creating an application. and that application perform any human intervention.

Machine learning Subset of AI

→ Use Stat tool to analyze the data

→ explore the data

→ Preprocess the data.

→ Prediction, forecasting

→ Supervised ML → feature and label

→ Unsupervised ML → only feature. → Dependant feature.

→ Semisupervised ML.

→ Reinforcement learning.

→ Supervised ML

→ Regression : output will be continuous.

→ Classification : classify or category something. discrete value.

classification.

① Binary classification.

② Multiclass classification

③ Multilabel classification

JLL

output

Movies	Action	Romance	Thriller	Suspense
Endgame	1	0	1	1
Drishyam	0	0	1	1

then, recommendation.

forecasting - Regression Problem.

→ Time series analysis.

→ Unsupervised ML

→ Clustering

→ Segmentation

→ Reduce dimension.

→ Semisupervised ML

initially → labeled data - Model (train) → Rec.

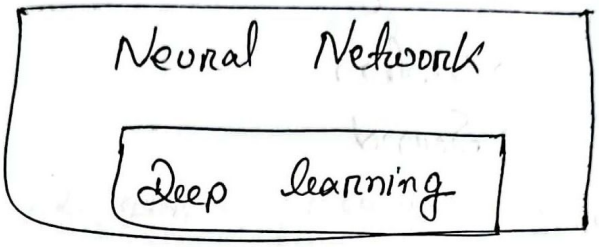
⇒ Based on the user interaction are the next data → train model

Continuously → New Recommendation.

⇒ Reinforcement learning

⇒ New born baby → Young man.

Deep learning - Subset of machine learning.



- How our brain works?
- Main aim of Deep learning is Mimic the human brain.
- ANN - Artificial neural network.

  Input : Tabular kind of dataset
- All the task done by NL can be do it by ANN

CNN → Convolution Neural Network.

Input : → Images (Matrix format)
→ Videos
→ Sound

Application :

1. Image classification ≠ Recognize
2. Object detection.
3. Object de segmentation.
4. Tracking

Recognation vs detection vs Segmentation.

→ Image classification technique. — ~~CNN~~

→ Transfer learning learning Image technique.

CNN (Image Detection)

II. Object Detection.

RCNN, FAST. RCNN, FASTE RCNN

SSD, YOLO, Detectron.

III. Segment, Detect, Recognize, classify.

IV. Tracking : Capture the object and try to track where its moving.

⇒ RNN : Recurrent Neural Network.

Input data : Text data
Time Series
Sequential data.

- -> RNN
- -> Bidirectional RNN
- -> LSTM
- -> B. LSTM
- -> BERT
- -> Transformer
- -> Encoder, decoder.
- -> GAN
- -> GPT1, GPT2, GPT3

Introduction to Neural Network.

- Neural Network : Combination of many neuron build a network means neural Network.

⇒ Perceptron : Single layer neural Network.

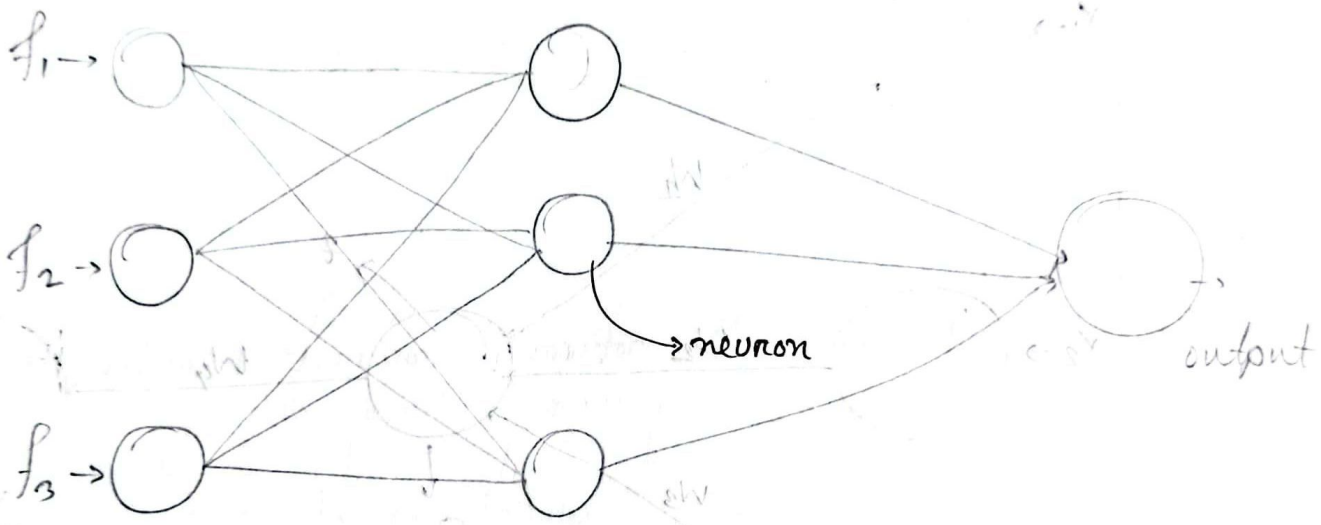
⇒ Multi-layer Perceptron } Neural Network

→ Not able to learn very properly

⇒ In 1986 Geoffrey hinton invented "back-Propagation". Able to learn efficiently.

⇒ Not invented actually. what he claimed in that he was the person to clearly demonstrate that backpropagation could learn interesting

internal representations and that this is what made it popular.



It can any number of feature

Input layer

It can be many layer and any number of neuron

Hidden layer

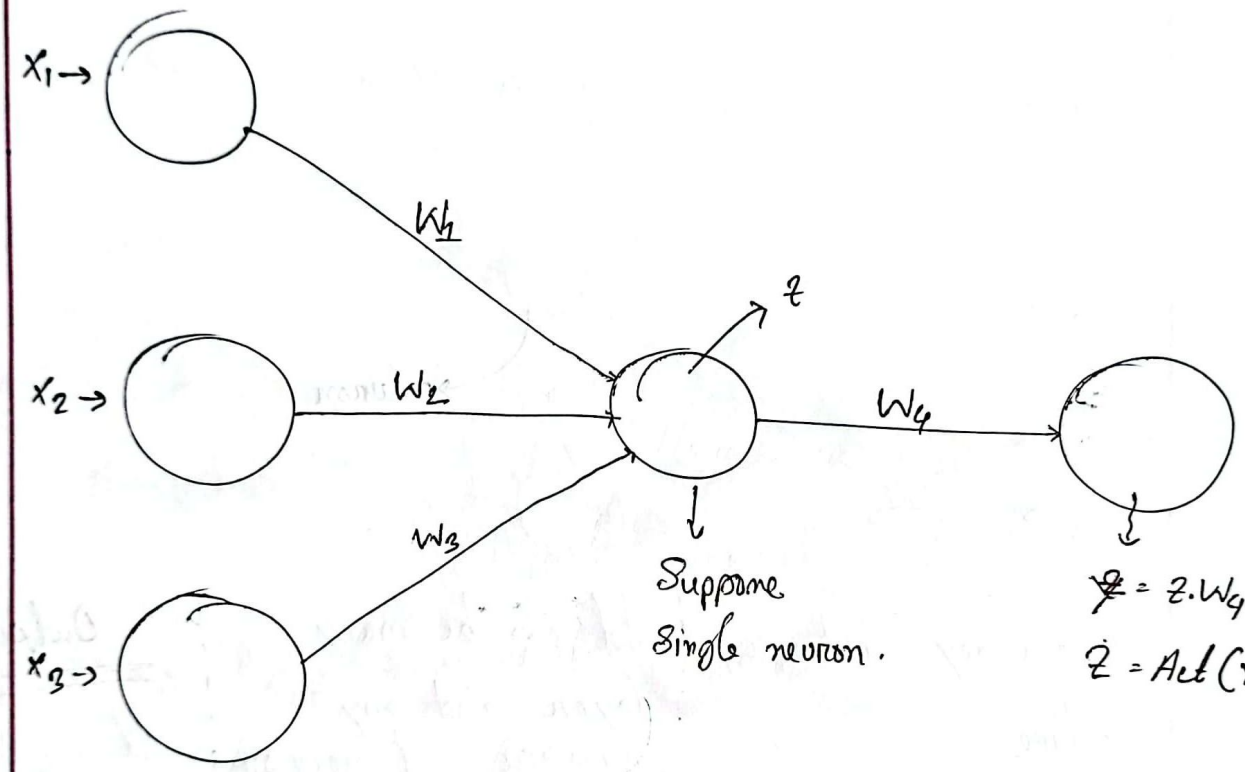
Output layer

It may be may output.

⇒ Hidden layer can have any number of neuron.

Basic NN Architecture

How Neural Network work.



Process

Step 1 : $\sum_{i=1}^{n} W_i x_i$

Step 2 : Activation function $\left(\sum_{i=1}^{n} W_i x_i \right)$

$$y = W_1x_1 + W_2x_2 + W_3x_3 + bias.$$

$$\hat{y} = Act(y)$$

⇒ Sigmoid function like ← Activation function.

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

⇒ Activation function transfor the result between 0 to 1.

Question: why we weight or what are the importance of weight and Activation function.

input ⟶ weight ⟶ bias ⟶ output

(x) (*W) (+b) ⟶ y

⇒ Weight help to determine the importance of any given variable.

→ ① Determine the importance of any given variable
↳ every input.

larger weights. greater importance to the decision or outcome.

→ ① DeepAI.org.

- Within a neural network there an input layer, that takes the input signal and passes them to the next layer.
- Next, the neural network contains a series of hidden layers which apply transformation to the input data. It in within the nodes of the hidden layer the weight are applied.

⇒ for example, a signal node may take the input data and multiply it by an assigned weight value, then add bias before passing the data to the next layer. The final layer of the neural network also known as the output layer.

⇒ Weight and bias both learnable parameters.

⇒ Initially randomly or any way assign weight and bias.

⇒ An training neural network continues, both Parameters are adjusted towards the device value and the correct output

⇒ "Bias" represent how far off the prediction are from the intended value.

⇒ The weight is connection between two neurons.

⇒ The strength of the connection between two neurons.

⇒ Weight decides how much influence the input will have on the output.

⇒ Weight affects the amount of influence a change in the input will have upon the output.

⇒ A low weight value will have no change on the inpnt, and alternatively a longer weight value will more significantly change the output.

⇒ example : If we connect a neuron to another with a weight, the higher the weight the higher the activation in of the second neuron.

Activation function

① Sigmoid AF

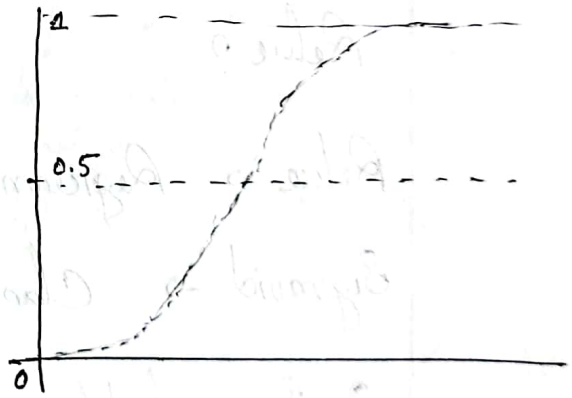
$$\frac{1}{1+e^{-y}}$$

$$y = \sum_{i=1}^{n} w_i x_i + \text{bian}$$

Act (y)

ex: used in logistic Regression.

Transform the result between 0 to 1.



↓

Decide your neuron Active or not.

Active means transfer the signal to the next layer

② Relue Activation function.

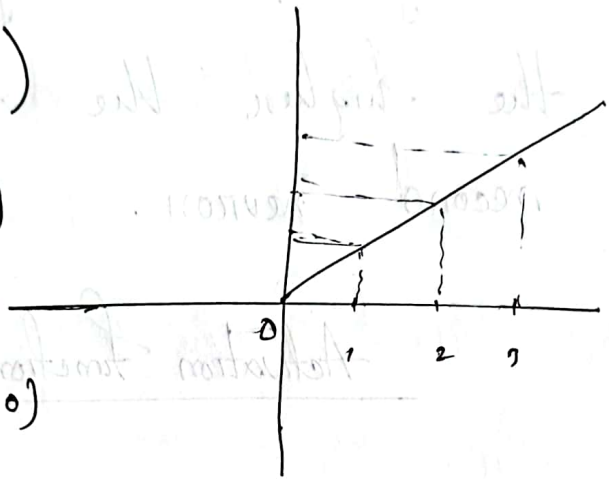
$\max(y, 0)$

-ve → $\max(y, 0)$

$= 0$

+ve → $\max(y, 0)$

⇒ +ve



⇒ when we sigmoid ? when we use Relue ?

Relue → Regression problem.

Sigmoid → Classification problem.

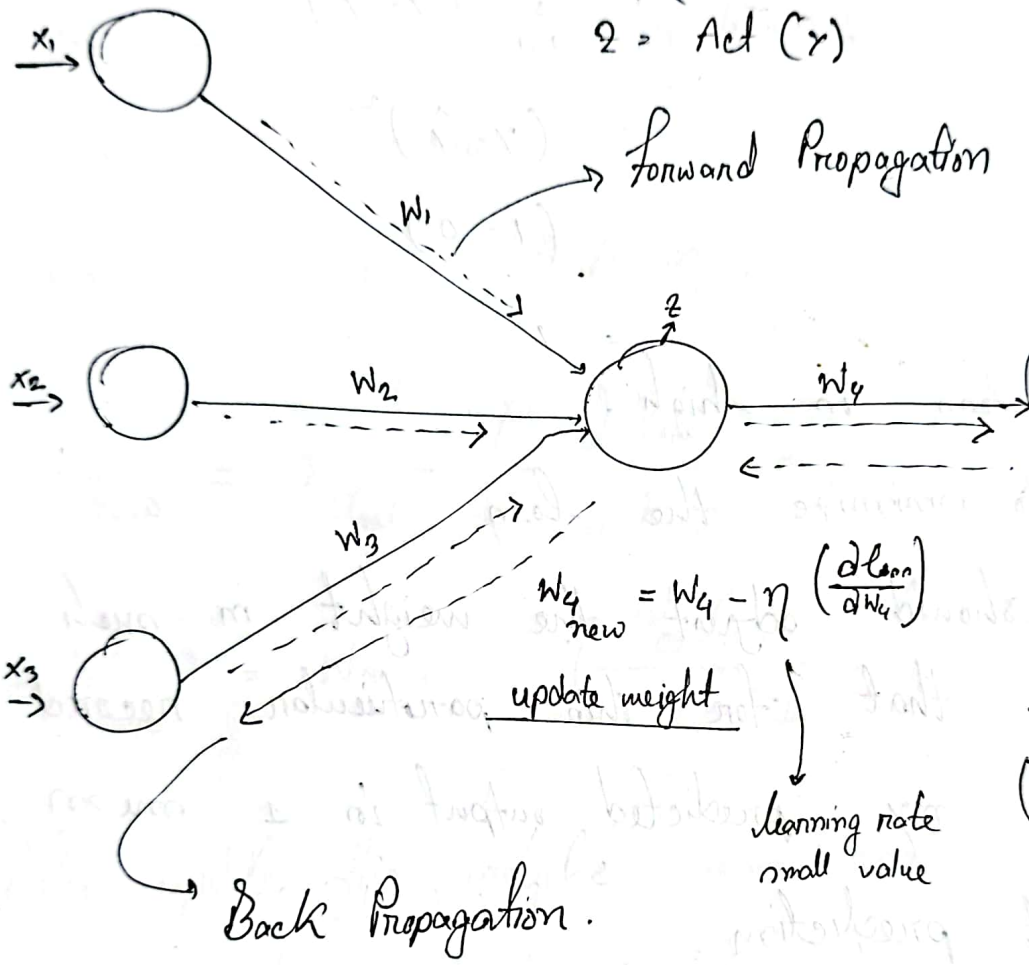
→ In hidden layer we can use relue.

→ But the final output layer, should always use sigmoid.

Neural network training with backpropagation.

$$Y = (x_1 w_1 + x_2 w_2 + x_3 w_3) + \text{bias}$$

$$Z = \text{Act}(Y)$$



$$W_{4\,new} = W_4 - \eta \left(\frac{\partial loss}{\partial W_4}\right)$$

update weight

learning rate
small value

let, $\hat{Y} = 0$

Actually, $Y = 1$

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

$$= (0 - 1) = 1$$

→ normal loss fine

Use Case:

Play	Study	Sleep	(Y) o/p
2h	4h	8h	1 → Pass the exam
			0 → fail the exam

loss function normally

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

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

$$= (1 - 0)$$

$$= 1$$

So loss is high.

→ try to minimize the loss.

→ we should adjust the weight in such a way that for this particular record always my predicted output is 1 mean, correct prediction

→ loss value decreasing (minimize the loss)

→ Accuracy increasing

It can be done by using optimizers

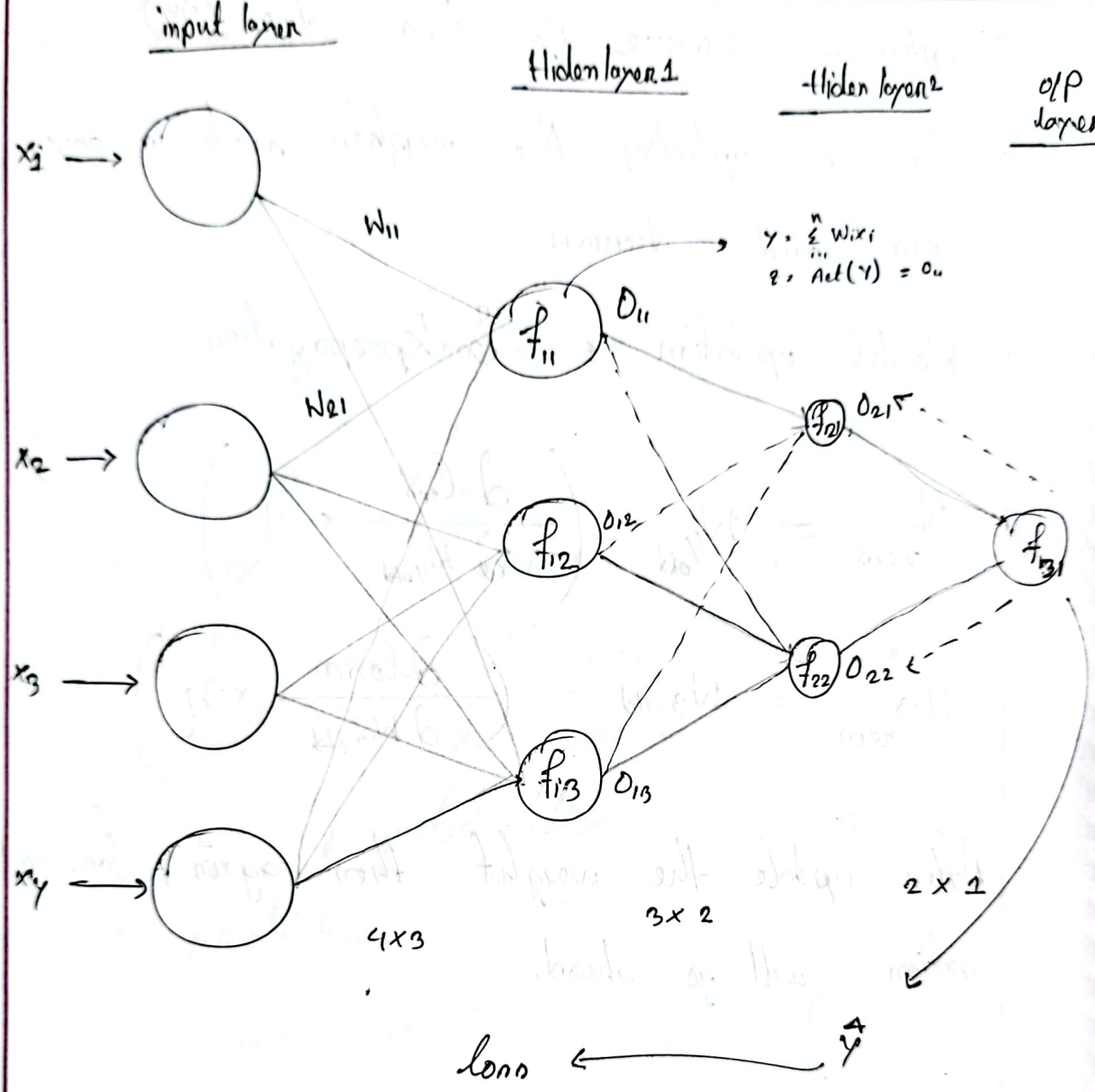
- Optimizer reduce the loss value (true)
- So we updating the weights such a way that loss will decrease.
- Weight updation ← Backpropagation.

$$W_{4_{new}} = W_{4_{old}} - \left(\frac{\partial loss}{\partial W_{4_{old}}} \times \eta \right)$$

$$W_{3_{new}} = W_{3_{old}} - \left(\frac{\partial loss}{\partial W_{3_{old}}} \times \eta \right)$$

After update the weight then again forward propagation will go ahead.

"Multi layer Neural Network"



loss function: $(y - \hat{y})^2$ ↓

→ If we want to reduce loss, we use optimizer

→ Optimizer update the weight such a way that loss will decrease mean y, $\hat{y}$ almost same

⇒ Each and every layer updating the weight on efficient way. how it is done?

"Backpropagation"

Optimizer : Gradient descent, ----------

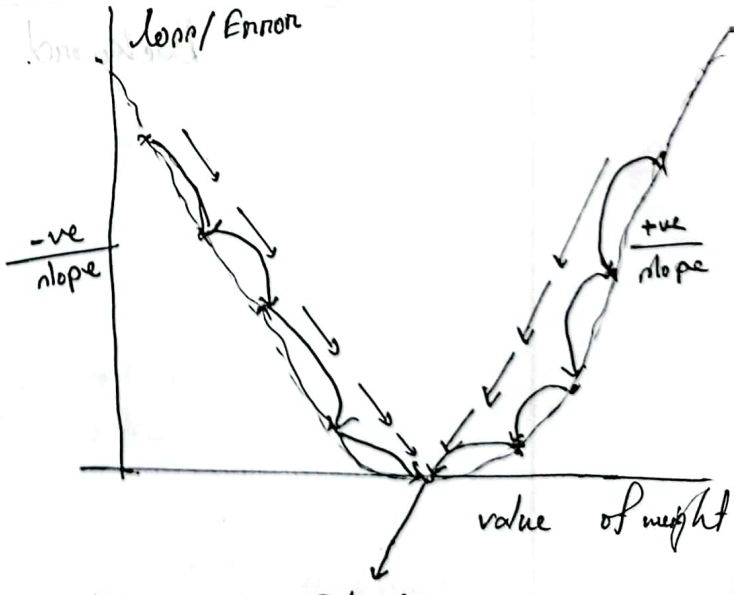
⇒ first weight and biases initialize randomly and then update in Backpropagation.

⇒ Weight updation.

$$W_{new} = W_{old} - \left(\frac{\partial loss}{\partial W_{old}} \times \eta \right)$$

why derevative?

→ Findout the slop
→ Small changes



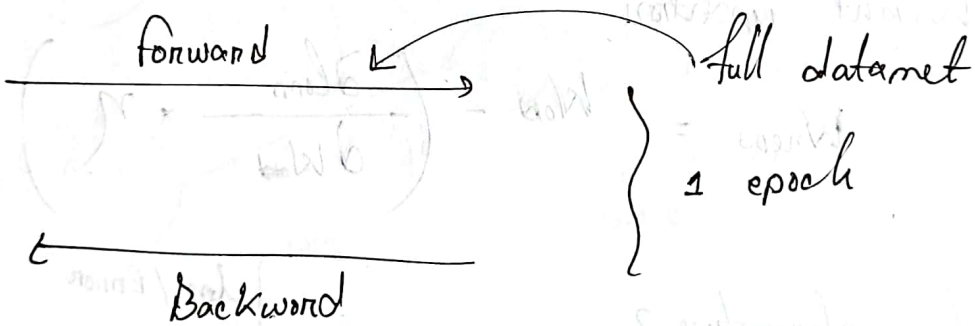
Convergence point ---- Global minima

η ⇒ learning rate is important.

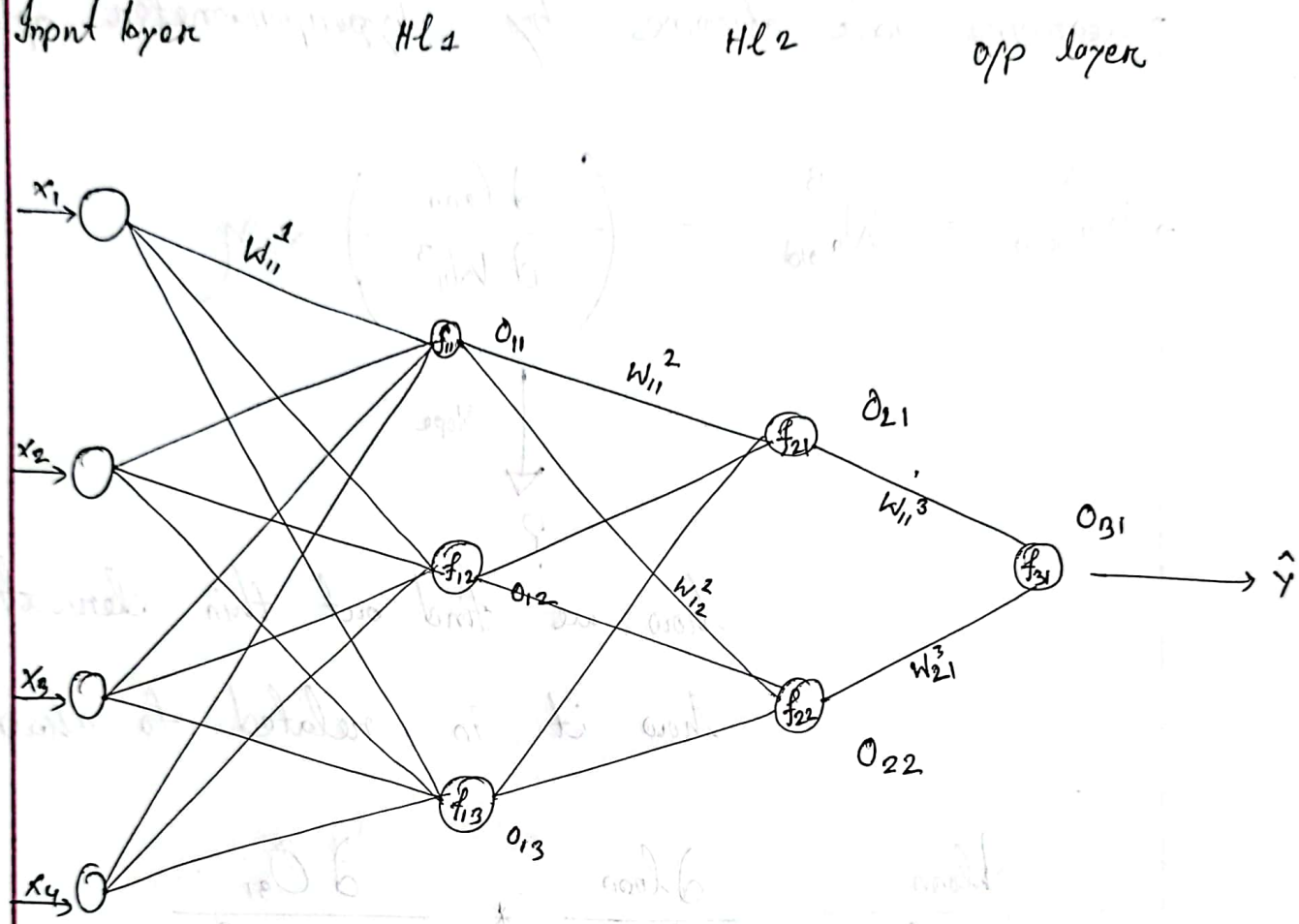
→ should not be very small or should not be very large.

→ How we know that we are on the right way by selecting correct learning rate?

→ After some epoch loss will be decreasing



"Chain Rule of differentiation with Backpropagation:



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

Backpropagation : weight updation process. How?

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

η → learning rate (How to choose it?)

→ learning rate choose by hyperparameter optimization.

$$\Rightarrow W_{11}^{3}{}_{new} = W_{11}^{3}{}_{old} - \left(\frac{\partial loss}{\partial W_{11}^{3}}\right) \times \eta$$

slope

?

how we find out this derivative

how it is related to chain Rule.

$$\frac{\partial loss}{\partial W_{11}^{3}} = \frac{\partial loss}{\partial O_{31}} * \frac{\partial O_{31}}{\partial W_{11}^{3}}$$

$$\Rightarrow W_{21}^{3}{}_{new} = W_{21}^{3}{}_{old} - \frac{\partial loss}{\partial W_{21}^{3}} \times \eta$$

$$\frac{\partial loss}{\partial W_{21}^{3}} = \frac{\partial loss}{\partial O_{31}} * \frac{\partial O_{31}}{\partial W_{21}^{3}}$$

$$W_{11}^2 = W_{11_{old}}^2 - \frac{\partial L_{oss}}{\partial W_{11}^2} \times \eta$$

?

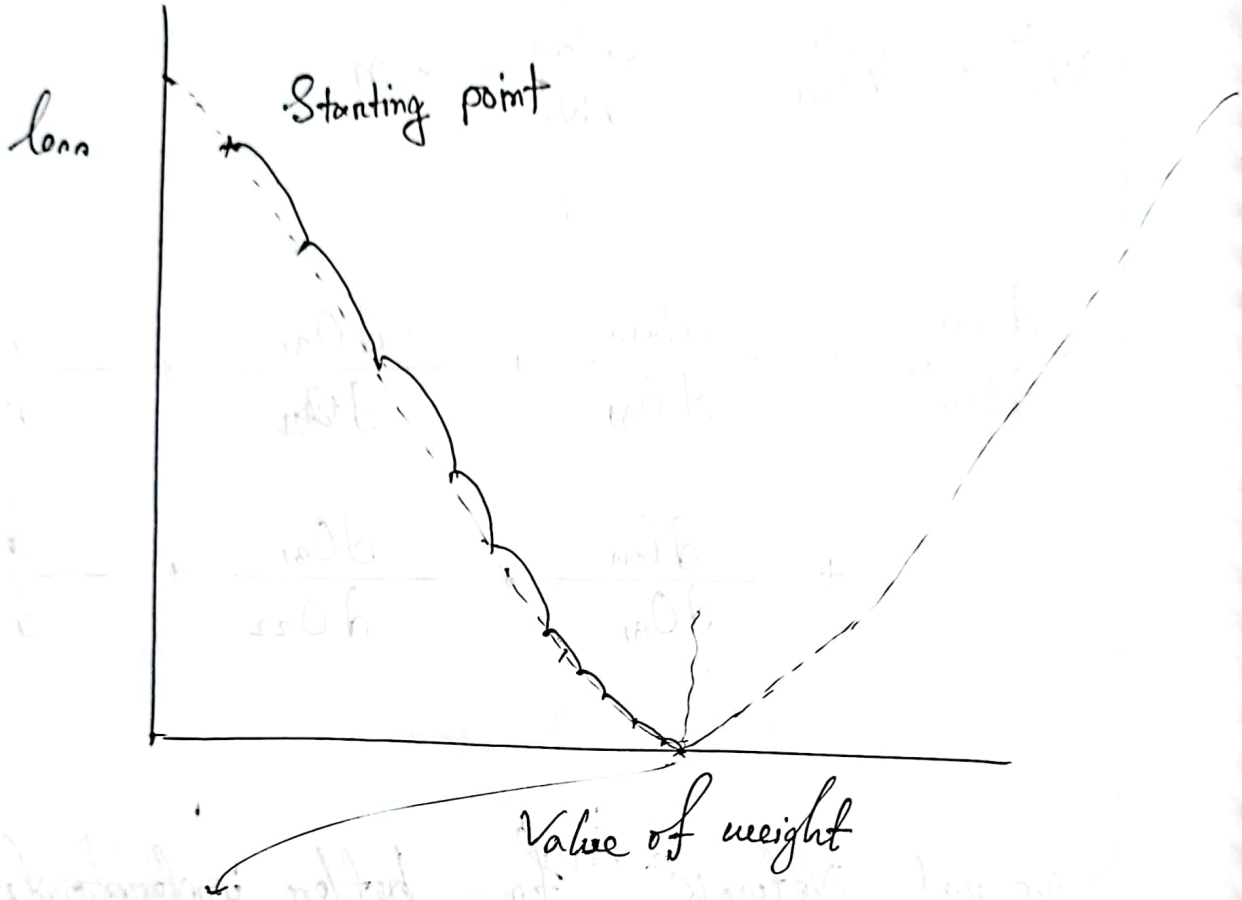
$$\frac{\partial L_{oss}}{\partial W_{11}^2} = \frac{\partial L_{oss}}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial W_{11}^2} \rightarrow (old)$$

$$+ \frac{\partial L_{oss}}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{22}} * \frac{\partial O_{22}}{\partial W_{12}^2} \rightarrow (old)$$

⇒ Neural Network for better understanding.

→ 3 blue 1 brown

→ IBM

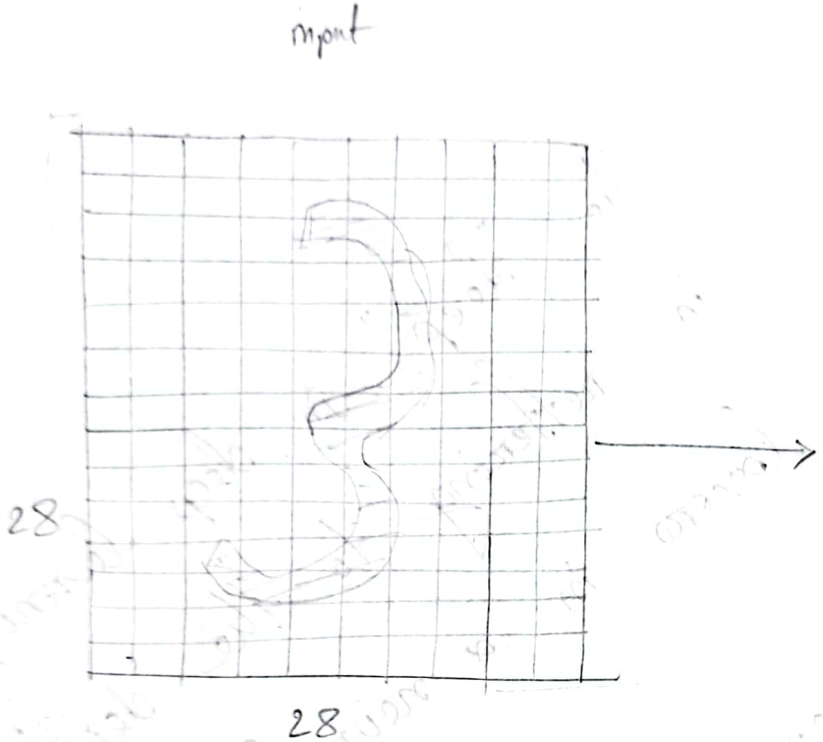


Point of convergence where the Cost function is at its minimum.

The "deep" in deep learning refferring to the depth of layers in a neural network.

A neural network that comprised of more than three layers - which would be inclusive of the "inputs" and "Output can be considered a deep learning algorithm

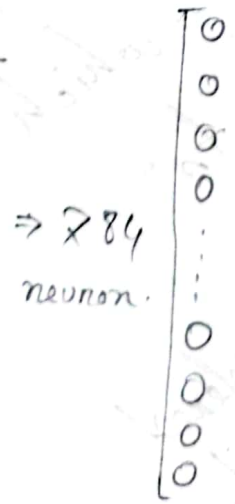
① Digit Recognize

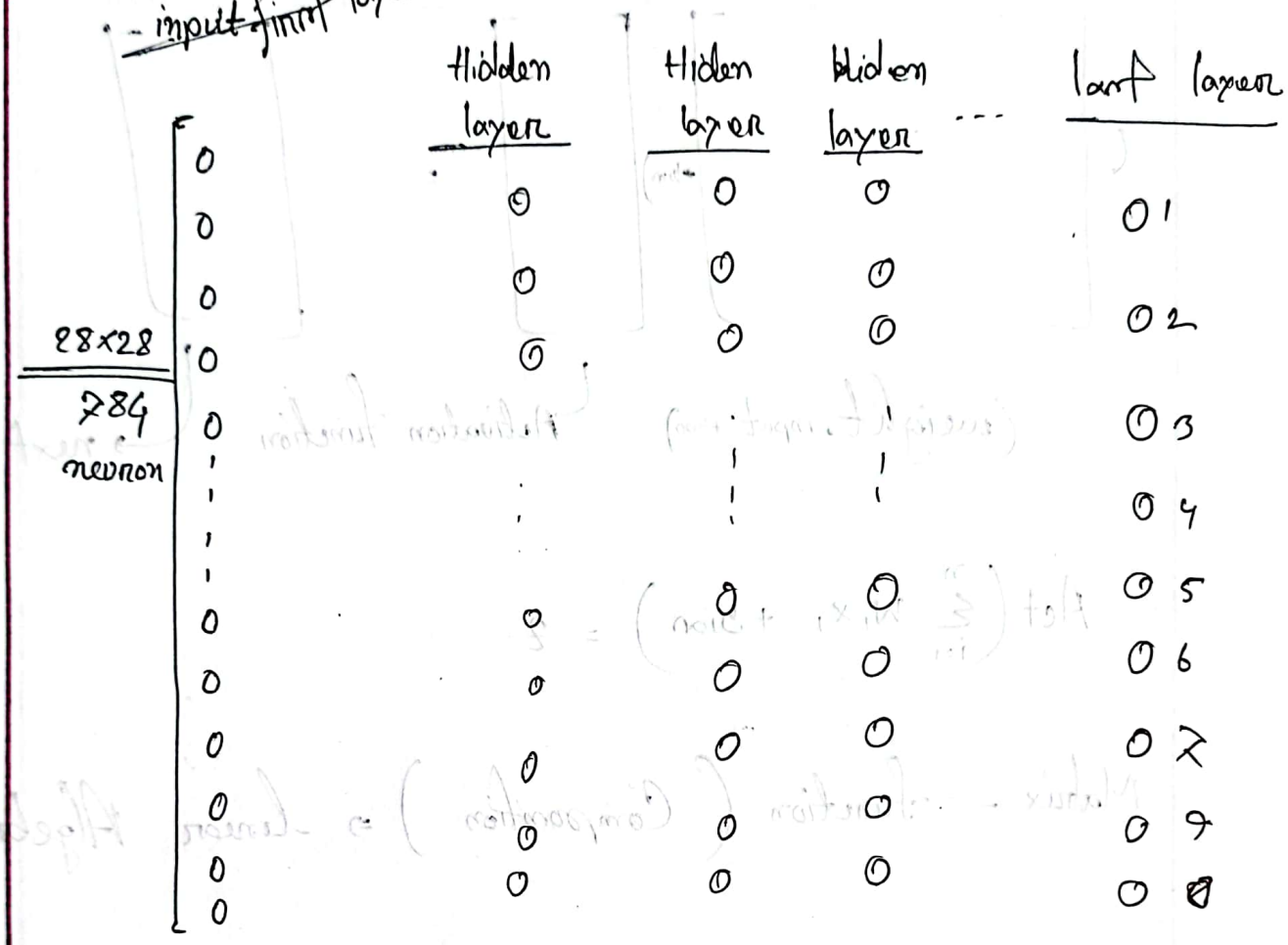


LSTM - for speech recognize
CNN → Image recognize } Good Performance

28 × 28 = 784 neurons

Flatten





⇒ Suppose 2 hidden layer

Hidden layer 1 = 16 neuron

Hidden layer 2 = 16 neuron

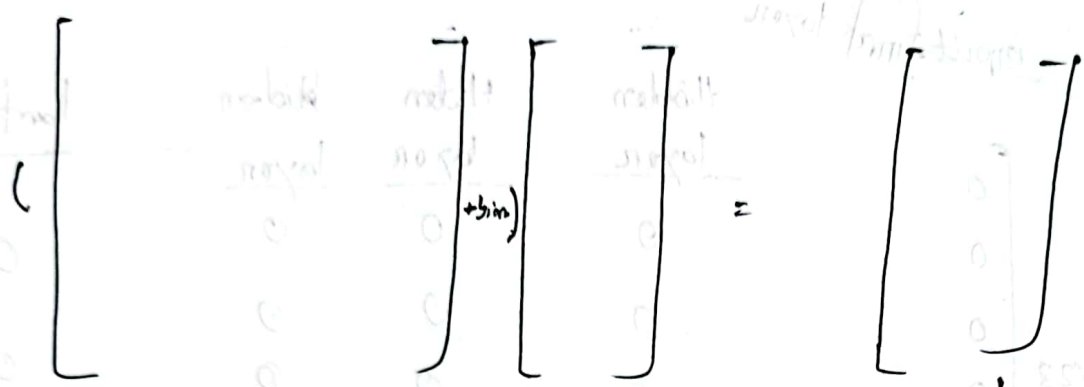
$784 \times 16 + 16$

$16 \times 16 + 16$

$16 \times 10 + 10$

weight and biases

3blue 1 Brown

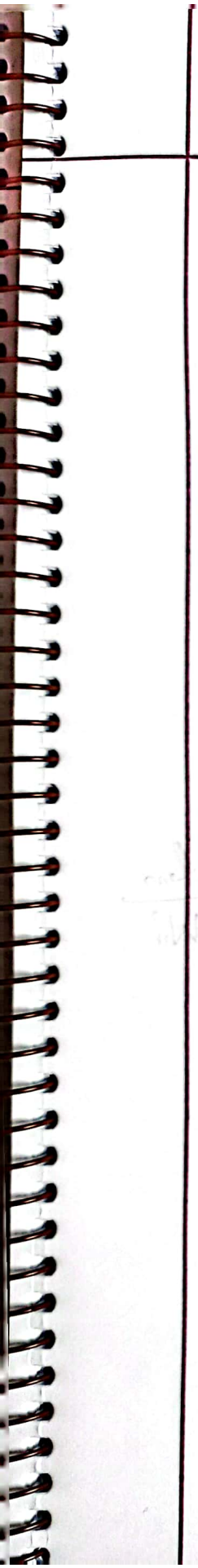


(weight . input + bias) → Activation function → next layer.

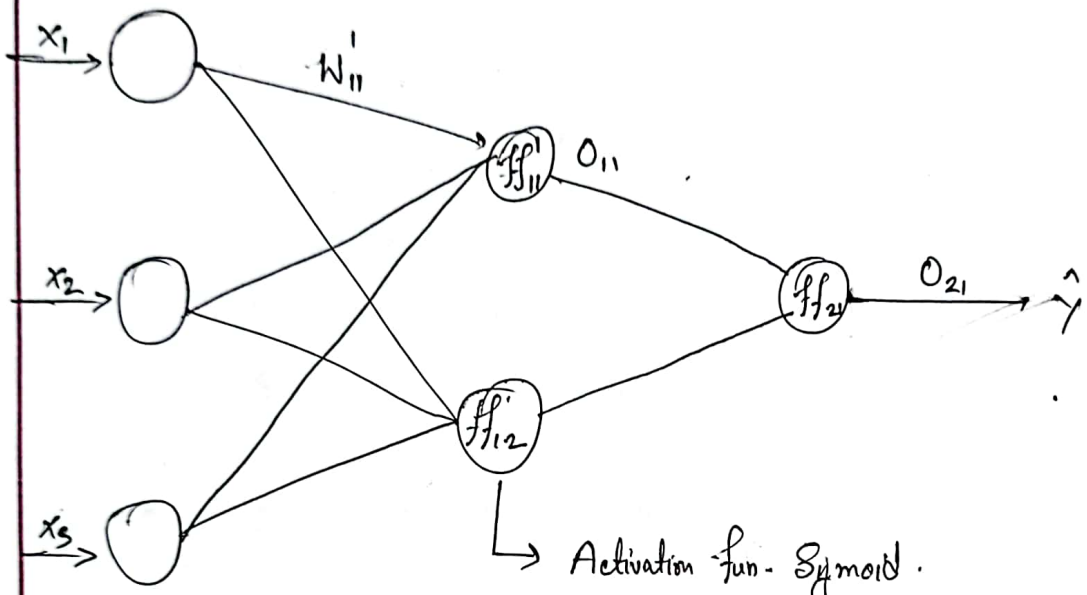
$$Act\left(\sum_{i=1}^{n} w_i x_i + bias\right) = z$$

Matrix - function (Composition) ⇒ linear Algebra.

Gradient Descent → IBM Article.



Vanishing Gradient Problem.



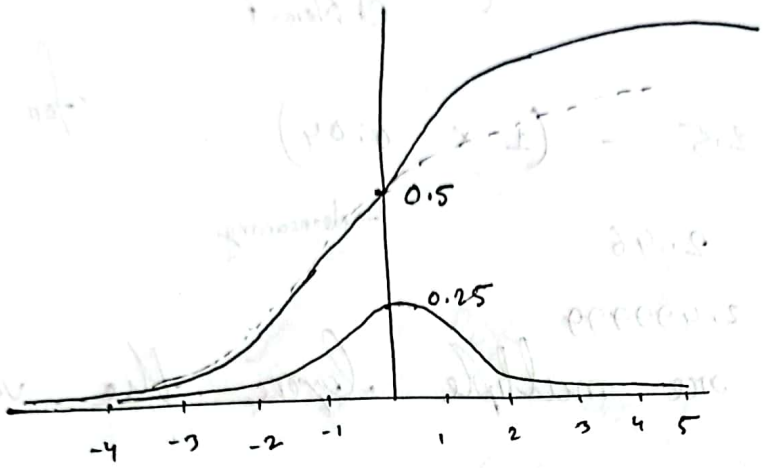
weight updation =

$$W_{11_{new}}^1 = W_{11_{old}}^1 - \eta \frac{d L_{on}}{d W_{11}^1}$$

$$\frac{d L_{on}}{d W_{11}^1} = \frac{d O_{21}}{d O_{11}} \cdot \frac{d O_{11}}{d W_{11}^1}$$

Sigmoid function convert output 0 - 1

derivative of sigmoid : 0 - 0.25



$z = \sum W x + \text{bian}$ Sigmoid $f = \frac{1}{1+e^{-z}}$

$\frac{d\sigma}{dz} = 0 - 0.25$

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

$\Rightarrow \frac{dl}{dW_{11}'} = \frac{dO_{21}}{dO_{11}} \cdot \frac{dO_{11}}{dW_{11}'}$ //chain Rule//

$(0 - 0.25)$ $[0 - 0.25]$

bdn, 0.2 0.2 $= 0.04$

new weight $= 2.5 - 1 \times 0.04$

$= 2.46$

$$W_{11}^{3}{}_{new} = Weight_{old} - \eta \frac{d\,loss}{d\,Weight}$$

$= 2.5 - (1 \times 0.04)$ for a single layer

$= 2.46$ ↳ decreasing

$\vdots$

$= 2.499999$

- If there are multiple layer the value of derivative is decreasing.
- So once a time old weight and new weight have very small change (may be unchanged).
- Approximately old and new weight will be matching.

⇒ That is the reason the Sigmoid is bad

And that's reason "Sigmoid" is not used "each and every" layer. but can used in the Last layer.

" ReLU Activation function solve this.

As a result the gradient descent never converge to the optimum. This is known as vanishing gradient problem.

- Sigmoid function.
- tanh also.

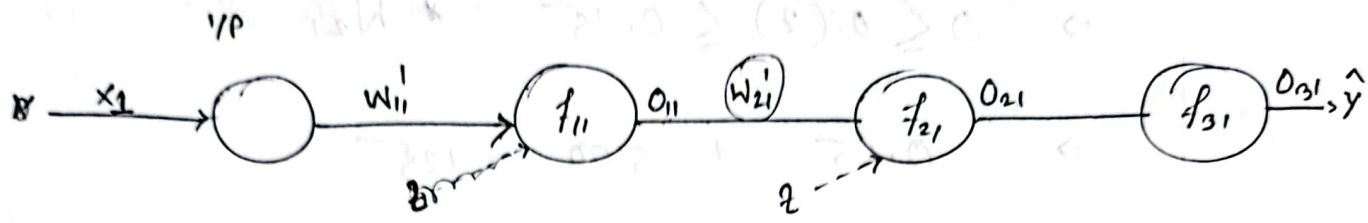
$weight_{old} \approx weight_{new}$

almost same.

- It occurs when gradient is too small [derivative]

$$\frac{\partial Loss}{\partial Weight}$$

Exploding gradient Problem.



$$W_{11_{new}}' = W_{11_{old}}' - \frac{\partial L_{oss}}{\partial W_{11}'} * \eta$$

$$\frac{\partial L_{oss}}{\partial W_{11}'} = \frac{\partial L_{oss}}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial O_{11}} * \frac{\partial O_{11}}{\partial W_{11}'}$$

Main Reason.

Sigmoid and "Weights"

$$O_{21} = \varphi(z) \longrightarrow \text{Activation function } / \sigma$$

$$z = W_{21} O_{11} + b_2$$

$$\frac{\partial O_{21}}{\partial O_{11}} = \frac{\partial \sigma(z)}{\partial z} * \frac{\partial z}{\partial O_{11}}$$

$$= 0 \leq \sigma(z) < 0.25 * \frac{\partial (W_{21} \cdot O_{11} + b_2)}{\partial O_{11}}$$

$$= 0 \leq \sigma(z) < 0.25 * W_{21}$$

$\Rightarrow \quad 0 \leq \sigma'(z) \leq 0.25 \quad * N_{21}$

$\Rightarrow \quad 0.25 * 500 = 125$

$$\frac{dL_{oss}}{dW_{11}'} = \frac{dO_{31}}{dO_{21}} \cdot \frac{dO_{21}}{dO_{11}} \cdot \frac{dO_{11}}{dW_{11}'}$$

$$200 \times 125 * 100$$

large value.

$$\text{Weight}_{new} = W_{u\,old}' - \eta \frac{dL}{dW_u'}$$

[- large value]

New weight and old weight difference will be very high.

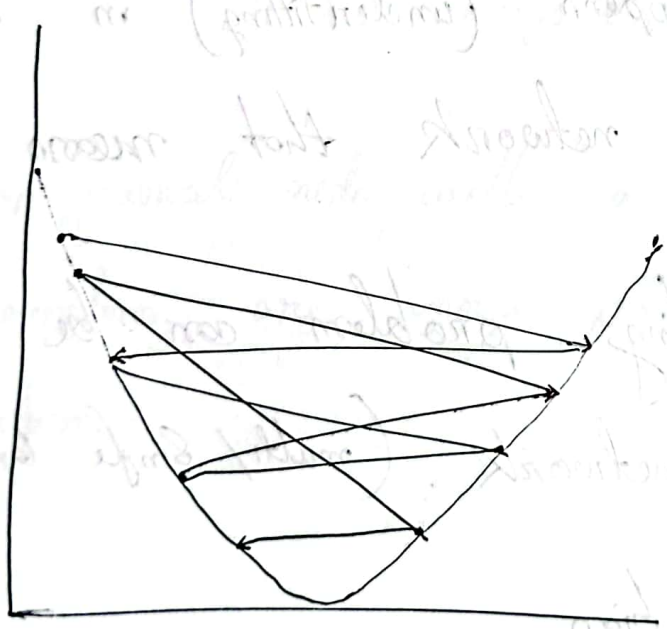
* Gradient descent will never converge. it will jumping here and there.

This happens when the gradient is too large.

→ derivative is large

→ Weight difference is high.

→ The challenges of vanishing / exploding gradients in deep neural network - Analytics vidhya.



convex

exploid gradient problem

vanishing gradient Problem.

but for non-convex ?

Dropout layers.

→ Huge data.

Huge amount of weight and biases Parameters., NN tends to overfit to the particular use-case.

⇒ We will never face underfitting problem in multilayer neural network.

⇒ "May be" happens (underfitting) in single layer neural network that means perceptron.

⇒ But overfitting problem can be happens in neural network (multi/single layer).

low bias

High variance.

- large neural nets trained on relatively small datasets can overfit the training data.

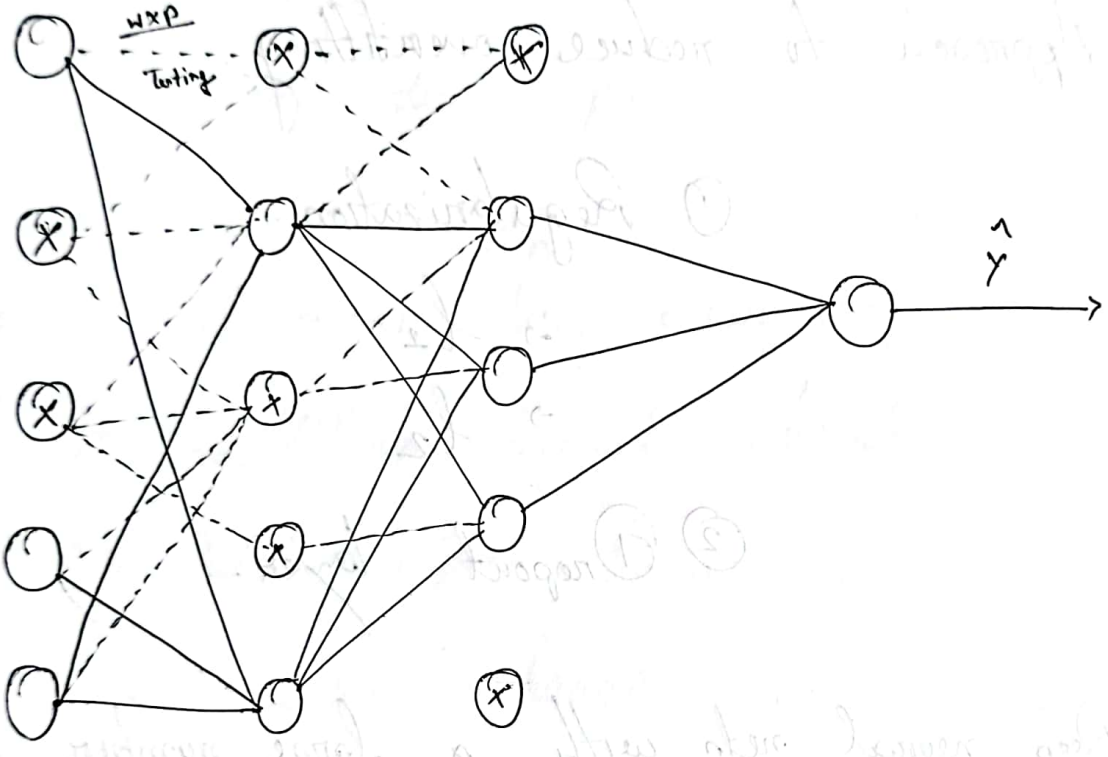
- Approach to reduce overfitting.

  ① Regularization..
  - → l_1
  - → l_2

  ② ① Dropout layer.

- Deep neural nets with a large number of parameters are very powerful machine learning system.

Dropout Ratio $0 \leq p \leq 1$



↳ "Randomly selected" some neuron and rest of them will be "deactivate".

training data → activating
→ deactivating

what about my test data. ?

for my [test data] every neuron will be connected

Additional work.

All these weight during training which are fixed. in multiply by probability

[w×p] weight and probability

→ dropout ratio.

Q: How do we select the p value means ration?

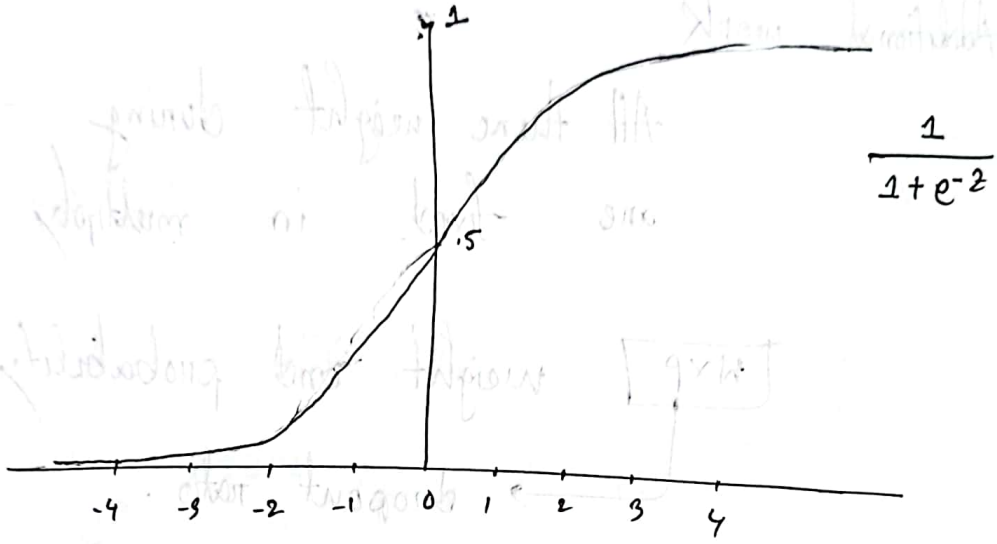
→ Hyperparameter optimization

→ if high overfitting problem p should be higher then 0.5.

Question: During forewards propagation are these dropped out neuron also zeron (deactive / turned off) during back-prop?

Activation function / transfer function.

① Sigmoid Activation function.



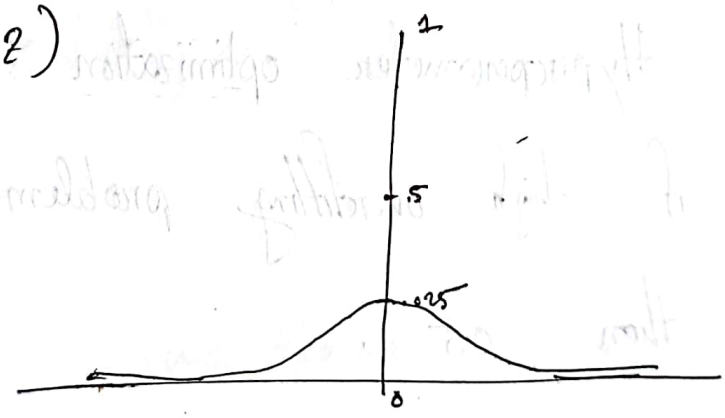
$$\frac{1}{1+e^{-z}}$$

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

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

$$z = Act(z)$$

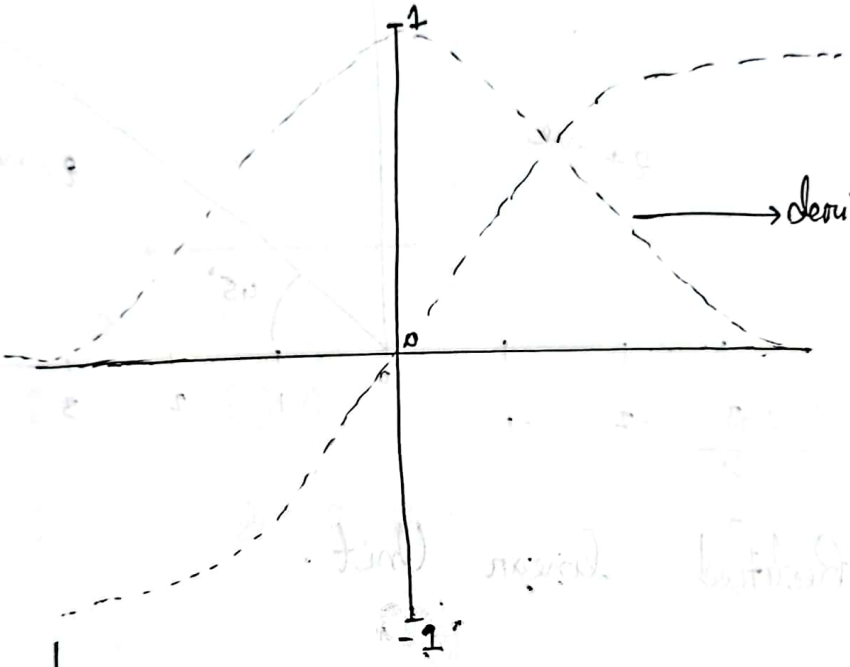
$$\frac{d(f)}{dz}$$



derivative of Sigmoid $0 \leq d\sigma(z)/dz \leq 0.25$

② Threshold Activation function,

$$\tanh = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

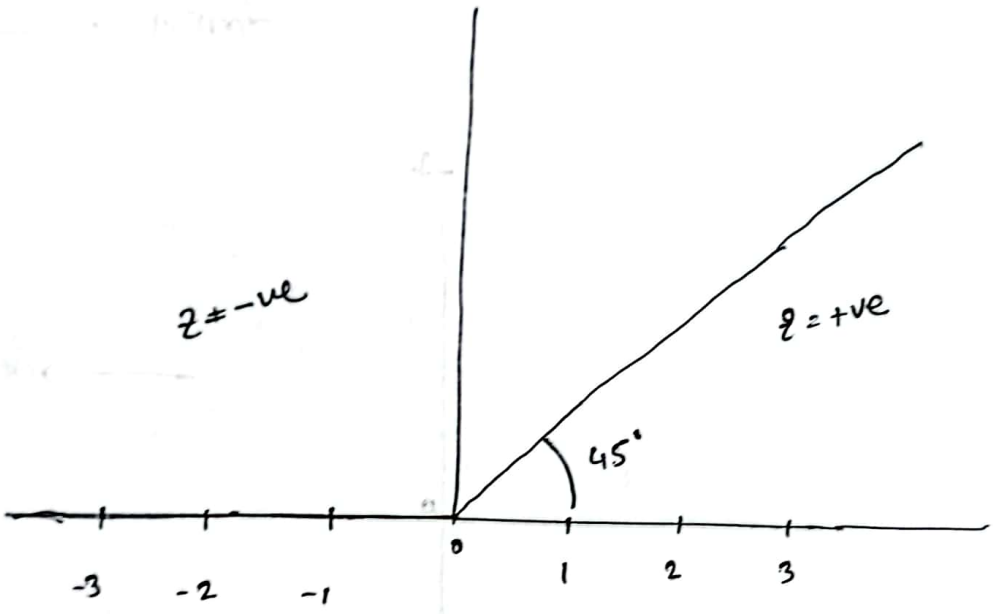


→ derivative range 0 - 1

↳ Zero centric thin function range -1 to 1

Both Sigmoid and tanh face vanishing gradient Problem.

③ ReLU activation function.



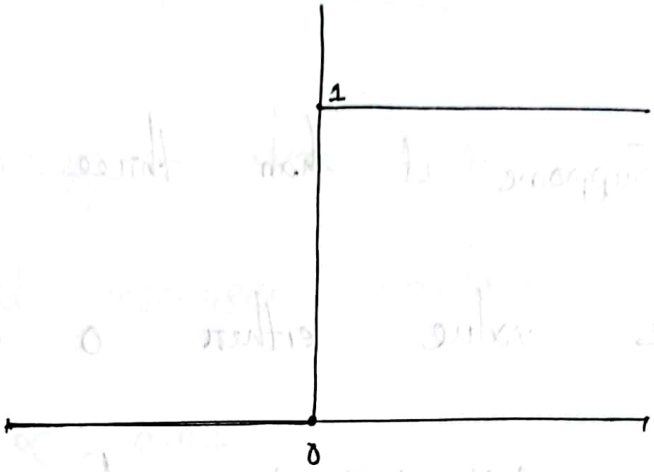
Rectified linear Unit.

$\max(0, z)$

$$R = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$$

⇒ The main catch here is that the Relu function doesn't activate all the neurons at the same time.

Derivative of ReLU



$$R = \begin{cases} z & z > 0 \\ 0 & z < 0 \end{cases}$$

$$\frac{d(0)}{dz} ?$$

** Zero has no derivatives.

→ Only certain numbers of neurons are activated.

→ Computationally more efficient than sigmoid, tanh.

→ limitation.

Dying relu.

$$Weight_{new} = W_{old} - \frac{d Loss}{dW} * \eta$$

→ Suppose it has three derivative.

derivative value either 0 or 1.

$1 \times 1 \times 1$ its ok

but,

$0 \times 1 \times 1 = 0$

weight are not updated and neuron it in not activate

⇒ Dead neuron

This is called Dying ReLU

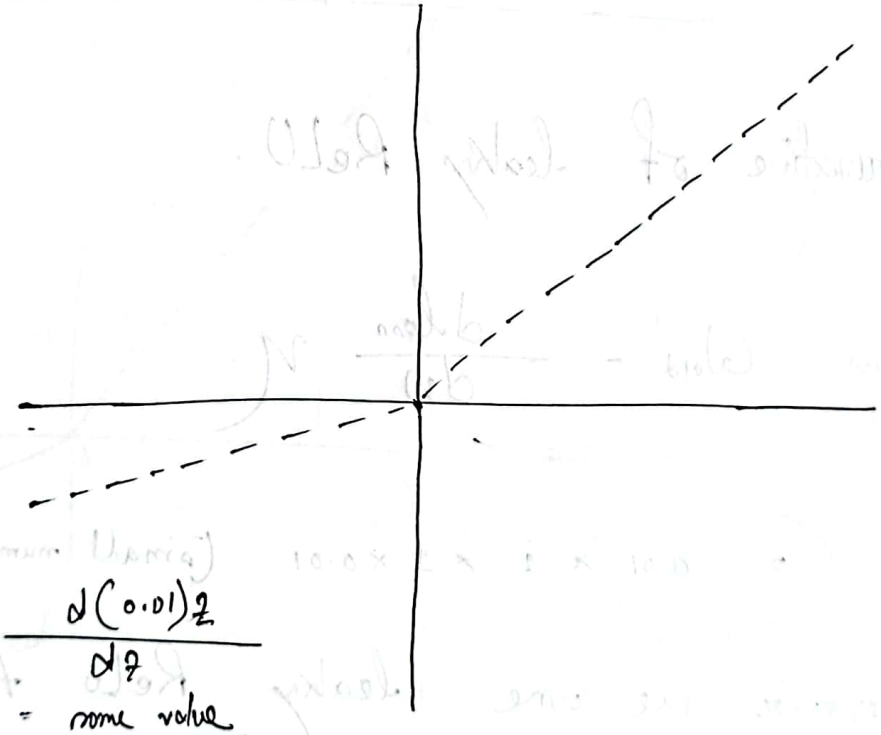
To fix this problem we use Leaky ReLU.

$$R = \begin{cases} z & z > 0 \\ \alpha z & z <= 0 \end{cases}$$

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

Instead of being 0, when $z < 0$, a leaky ReLU "allows" small non-zero constant gradient α.

Normally $\alpha = 0.01$

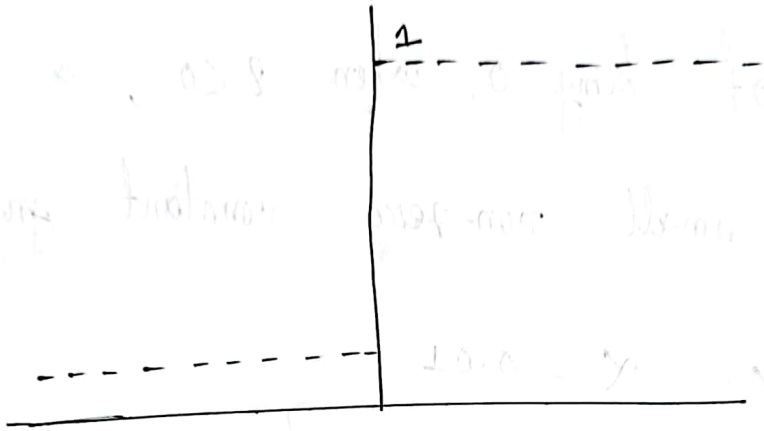


$$\frac{d(0.01)z}{dz}$$

= some value, not zero

= 0.01

$$R' = \begin{cases} 1 & z > 0 \\ \alpha & z < 0 \end{cases}$$



derivative of leaky ReLU.

$$W_{new} = W_{old} - \frac{d\,loss}{dw}\,\eta$$

$\rightarrow$ $0.01 \times 1 \times 1 \times 0.01$ (small number)

Whenever we use leaky ReLu there will be a chances of vanishing gradient problem.

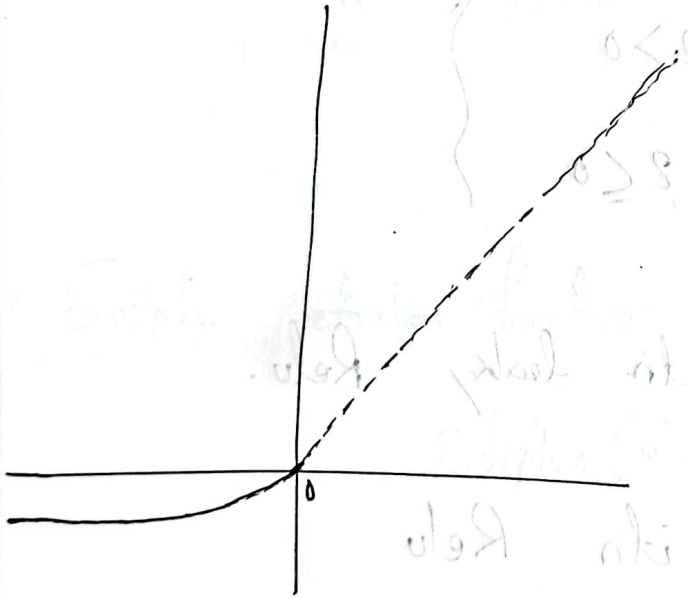
$\rightarrow$ if maximum number of weight is -ve.

Exponential linear Unit.

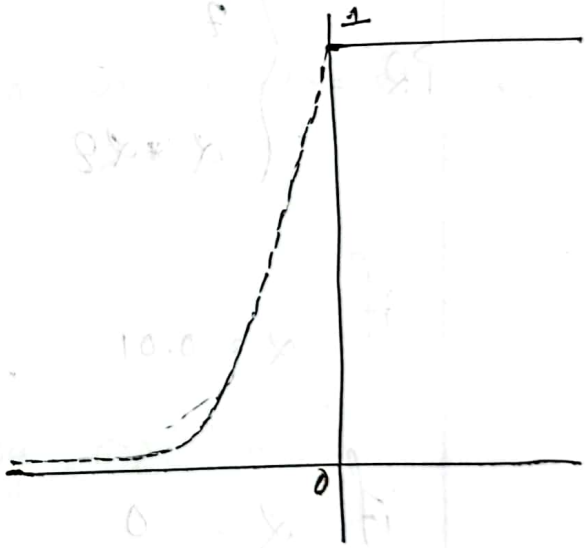
ELU in very similar to RELU except negative inputs.

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

↳ learnable parameter or can be fixed.



zero centered



derivative

$\frac{d\alpha(e^x - 1)}{dx}$ ↳ it's computationally expensive.

derivative is not possible for 0.

$$E'(z) = \begin{cases} 1 & z > 0 \\ \alpha \cdot e^{z} & z < 0 \end{cases}$$ derivative of Relu, function.

PReLU (Parametric ReLU)

$$PR = \begin{cases} z & z > 0 \\ \alpha * z & z \leq 0 \end{cases}$$

if $\alpha = 0.01$, itn leaky Relu.

if $\alpha = 0$, itn Relu

if α = anyvalue , can be value any kind of Problem. itn learnable parameter.

→ it become PRelu.

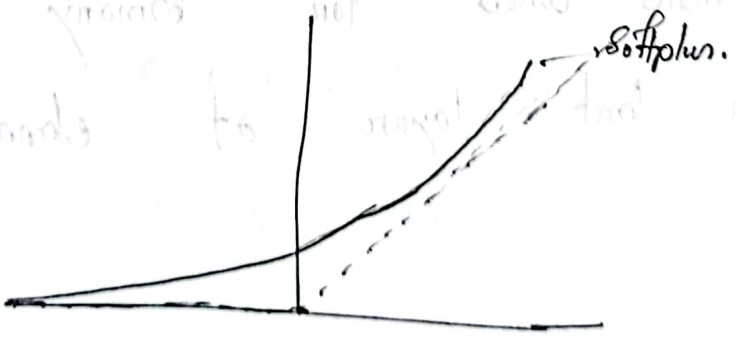
→ Swish activation function.

- → Swish (x) = x * sigmoid (x)
- → Used in LSTM
- → Also called Self-gating function.

whenever NN layers > 40.
then we should use swish.

⇒ Softplus activation function:

$$\text{Softplus}(x) = \log(\exp(x) + 1)$$



⇒ Softmax Activation function.

→ Used as the activation for the last layer of a classification - network.

→ Used for multiclass classification.

→ Converts a vector value to a Probability distribution.

→ Return tensor.

→ Used in a Dense layer.

⇒ Sigmoid used for Binary classification in the last layer of classification.

Softmax

Suppose output

[40, 30, 10, 5]

$$\frac{e^{40}}{e^{40}+e^{30}+e^{10}+e^{5}} =$$

$$\frac{e^{30}}{e^{40}+e^{30}+e^{10}+e^{5}} =$$

$$\frac{e^{10}}{e^{40}+e^{30}+e^{10}+e^{5}} =$$

$$\frac{e^{5}}{e^{40}+e^{30}+e^{10}+e^{5}} =$$

Dog

cat

monkey

Cow

Tensor

[0.6, 0.2, 0.1, 0.1]

= 1

it's dog

highest probability that in the final output

Sigmoid

Dog 0.6

Cat 0.4

} it's dog

⇒ Activation function for hidden layer.

→ ReLU

→ Swish if neural network having a depth greater 40 layers.

⇒ Activation function for "Output layer."

→ Regression : linear Activation function.

→ Binary classification : Sigmoid

→ Multiclass classification : Softmax.

→ Multilabel classification : Sigmoid.

Dense layer

Activation function in "hidden layers"

CNN → ReLU Activation

RNN → Sigmoid / tanh

Multilayer perceptron → ReLU

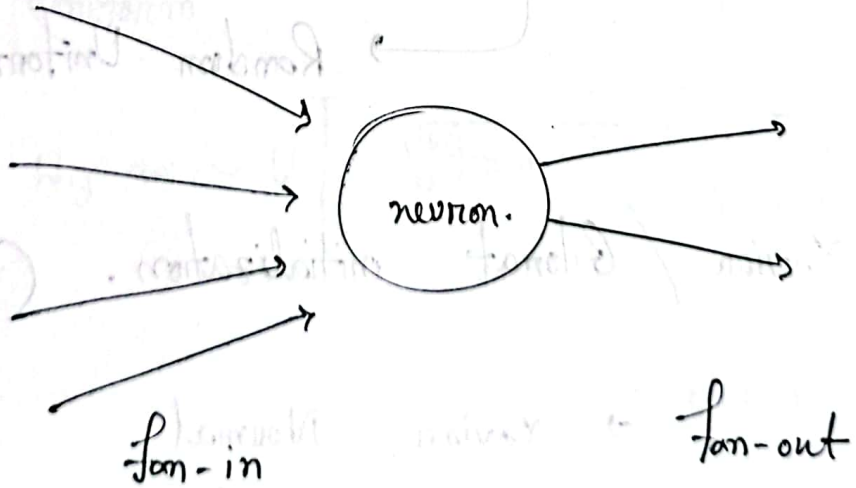
All activation function

1. Sigmoid
2. Threshold / tanh
3. Relu
4. Leaky Relu
5. PRelu
6. ELU
7. Softplus
8. Swish
9. Softmax
10. linear

Weight Initialization.

- -> Weight should be small. (not very small or big)
- -> Weight should not be same.
- -> Weight should have good variance.

Technique : None of the technique can say best.
Some work for some dataset.
Some of the work for some of activation function properly



① Uniform distribution.

$$W_{i,j} \sim \text{Uniform}\left[\underbrace{\frac{-1}{\sqrt{fan\text{-}in}}}_{a}, \underbrace{\frac{1}{\sqrt{fan\text{-}}}}_{b}\right]$$

lower maximum

→ Not Zero initialize

② Random initialize

- → Random Normal.
- → Random Uniform.

③ Xavion / Glonat initialization. (Sigmoid)

- → Xavion Normal
- → Xavian Uniform

→ Xavier Normal (Suitable for Sigmoid.)

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

$$\sigma = \sqrt{2/(\text{fan-in} + \text{fan-out})}$$

→ Xavier Uniform.

$$W_{i,j} \sim U\left[\frac{-\sqrt{6}}{\sqrt{\text{fan-in} + \text{fan-out}}}, \frac{\sqrt{6}}{\sqrt{\text{fan-in} + \text{fan-out}}}\right]$$

→ He. Uniform.

$$W_{ij} \sim U\left[-\sqrt{6/\text{fan-in}}, \sqrt{\frac{6}{\text{fan-out}}}\right]$$

Activation function = Relu.

⇒ He Normal.

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

$$\sigma = \sqrt{2/\text{fan-in}}$$

Suitable for Relu, leaky Relu

⇒ Gradient Decent
Stocantic GD.
Mini-batch GD
Batch GD

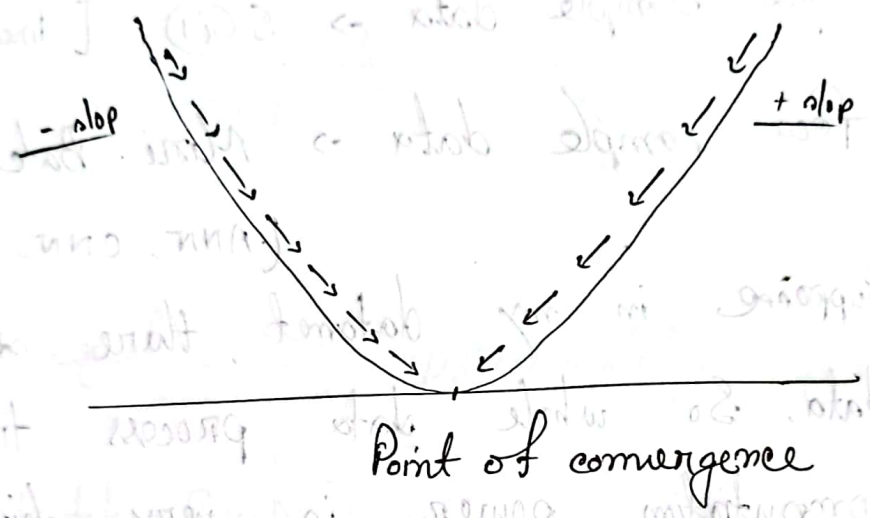
Same ?

Stocadic GD with momentum.

local minima

Global miniana.

→ Gradient Descent - IBM Article.



$$W_{new} = W_{old} - \frac{\partial L_{oss}}{\partial W} \eta$$

$\sum_{i=1}^{n} (y-\hat{y})^2$ ⇒ All the data points(n) of the dataset → Gradient Descent.

$(y-\hat{y})^2$ ⇒ 1 data at a time → Stocastic gradient Descent.

$\sum_{i=1}^{k} (y-\hat{y})^2$ ⇒ k data points [$k < n$] ⇒ Mini-batch SGD

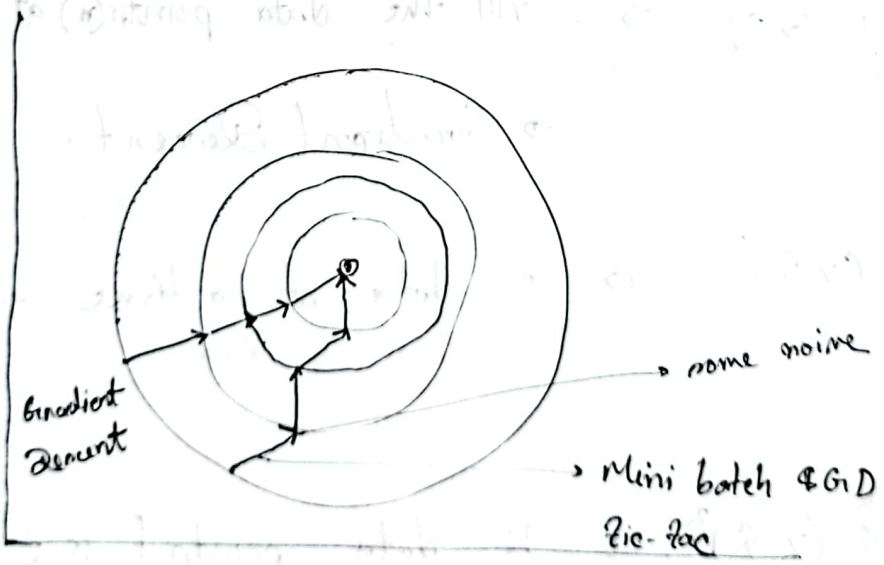
① All data points -> Gradient Descent.

② Single Sample data -> SGD. [linear Regression]

③ few sample data -> Mini-Batch SGD
[ANN, CNN, RNN]

- Suppose in my dataset, there are 1M data. So whole data process time and computation power in very high.

⇒ for that case we use Mini-Batch SGD



Sample

Population

$$\left[\frac{d\,loss}{d\,W_{old}}\right]_{\text{Mini-batch SGD}} \approx \left[\frac{d\,loss}{d\,W_{old}}\right]_{\text{SGD}}$$

↓

Problem higher Computational memory

→ Never equal . but approximately equal.

→ Mini-batch SGD noise solution : Stocastic Gradient Descent with momentum.

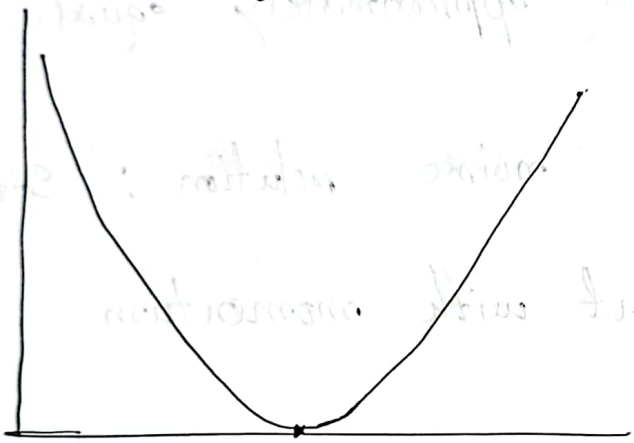
Global minima . local minima

- MB- SGD

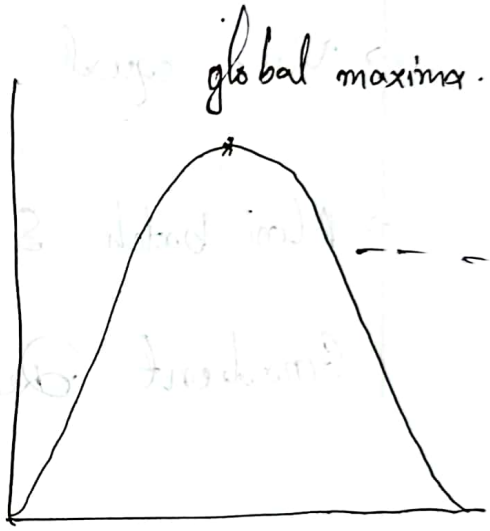
$$loss(w) = \sum_{i=1}^{k} (y - \hat{y})^2$$

↓

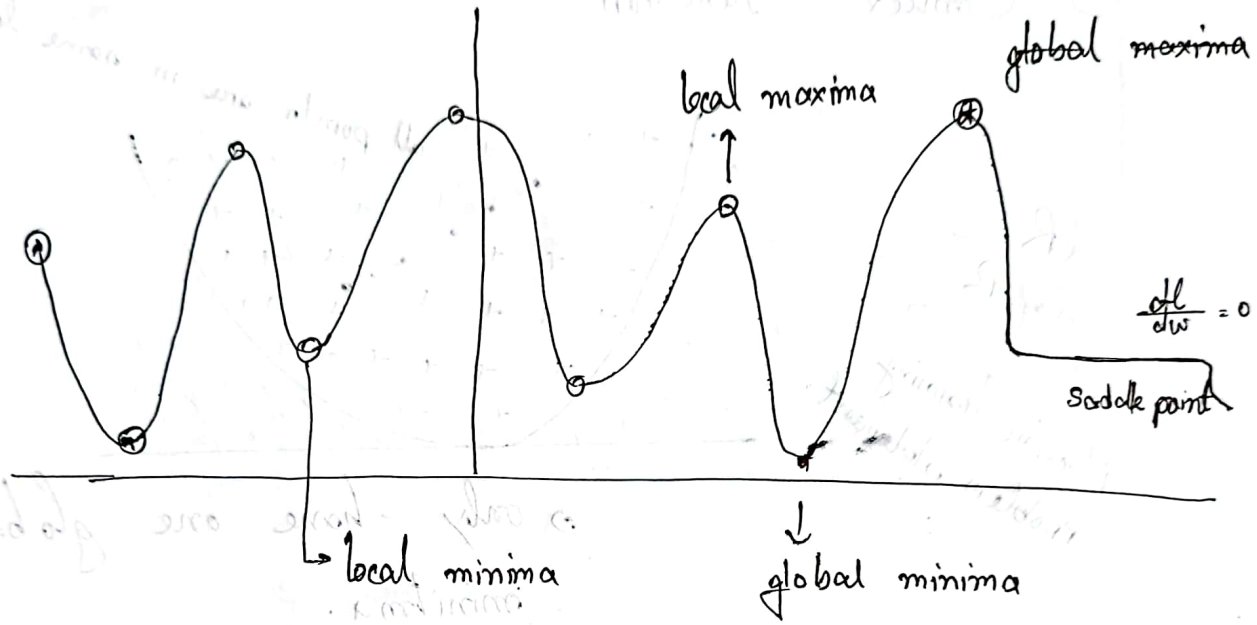
Mean square error



global minima



But in deep learning there are different loss function and higher dimention value. Curve are not like that.



→ derivative of point = 0

$W_{new} = W_{old} - 0$

$W_{new} = W_{old}$ ← Convergence point

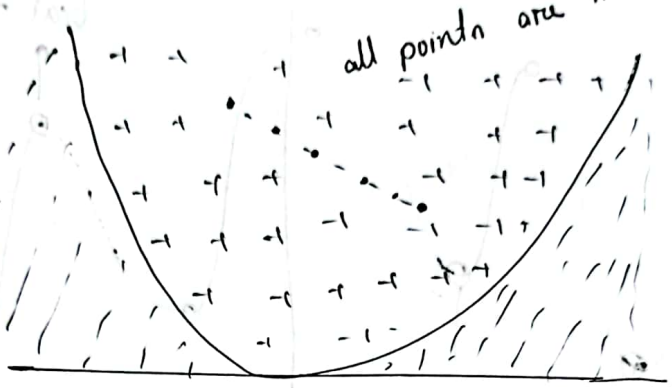
Gradient Descent → Problem

for this type of curve.

not reach the convergence point for non-convex function.

① Convex function

all points are in some line Region.



⇒ only have one global minima.

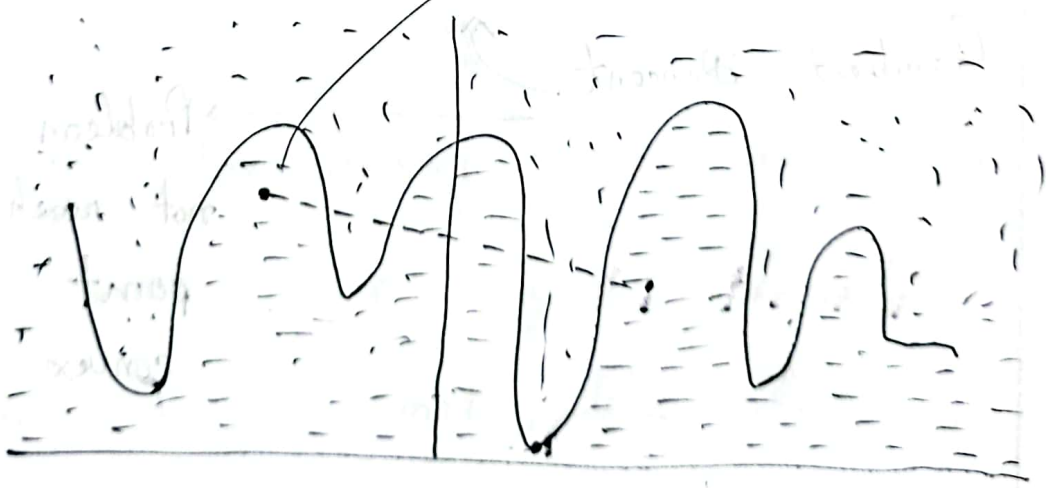
LR.
Log R

Machine learning
Problem statement.

② non-convex function

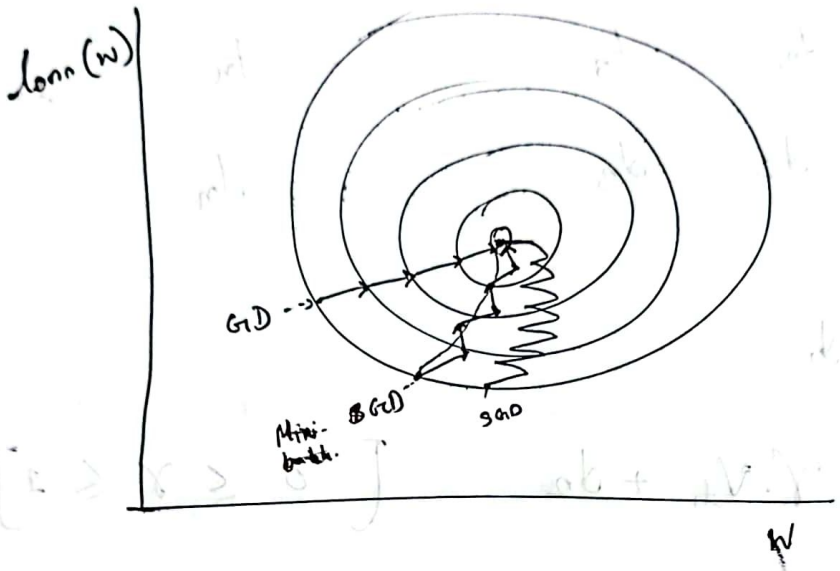
Deep learning

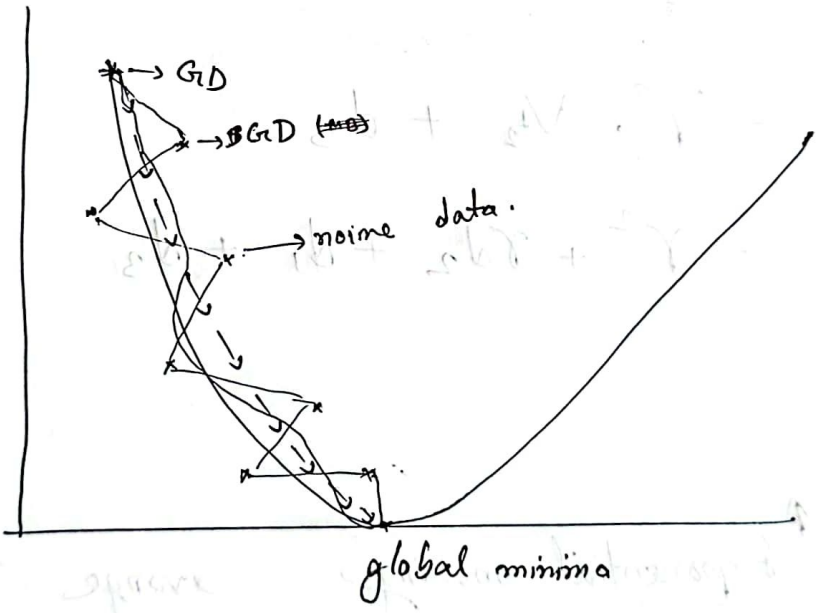
more than two region.



Occured in mostly deep learning technique.

SGD with Momentum.





SGD ⇒ Reach the global minima take large amount of time. So, we try to remove those noise.

How? : Exponential moving avarage.

t_1 . t_2 t_3 ---- t_n ...

d_1 . d_2 d_3 ---- d_n

$V_{t_1} = d_1$

$V_{t_2} = \gamma \cdot V_{t_1} + d_2 \quad [0 \leq \gamma \leq 1]$

$V_{t_3} = \gamma \cdot V_{t_2} + d_3$

$= \gamma^2 + \gamma d_2 + d_1 + d_3$

↑↑ Exponential moving avarge formula.

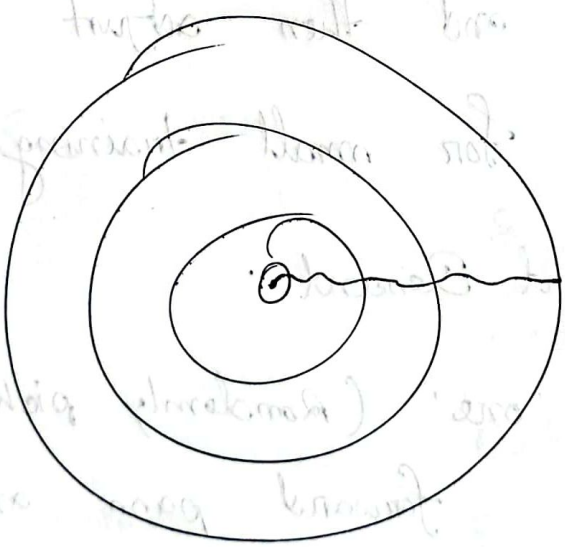
"Batch gradient Dercent and gradient dercent are same!"

GD / SGD

$$W_{new} = W_{old} - \eta \frac{d loss}{d W_{old}}$$

By applying moving avarage (momentum)

$$\Rightarrow W_{old} \left[\gamma V_{t-1} + \eta \frac{d loss}{d W_{old}} \right]$$



Smooth line GD with momentum.

$$V_{t-1} = 1 \left[\frac{dl}{dW_{old}} \right]_t + \gamma \left[\frac{dl}{dW_{old}} \right]_{t-1} + \gamma^2 \left[\frac{dl}{dW_{old}} \right]_t$$

Analytics vidhya ->

Batch Gradient Descent.

Stocastic " "

Mini Batch Gradient Descent.

Batch Gradient Descent

-> Use all training sample for one forward pass and then adjust weights.

-> Good for small training set.

Stcastic Gradient Descent

-> Use 'one' (Randomly picked) sample for a forward pass and then adjust weight.

-> Good when training set is very big and we dont want too much computation.

→ Batch gradient Descent

Batch Size = Size of training set.

→ Stocastic Gradient Descent

Batch Size = 1

→ Mini-Batch Gradient Descent

1 < Batch size < Size of training set.

Sample of dataset.

→ Epoch

→ Batch Size

→ MD-GD

Epoch,

whole data 1 in time training.

Suppose, A dataset 2000 data point.

Batch Size = 200

So,

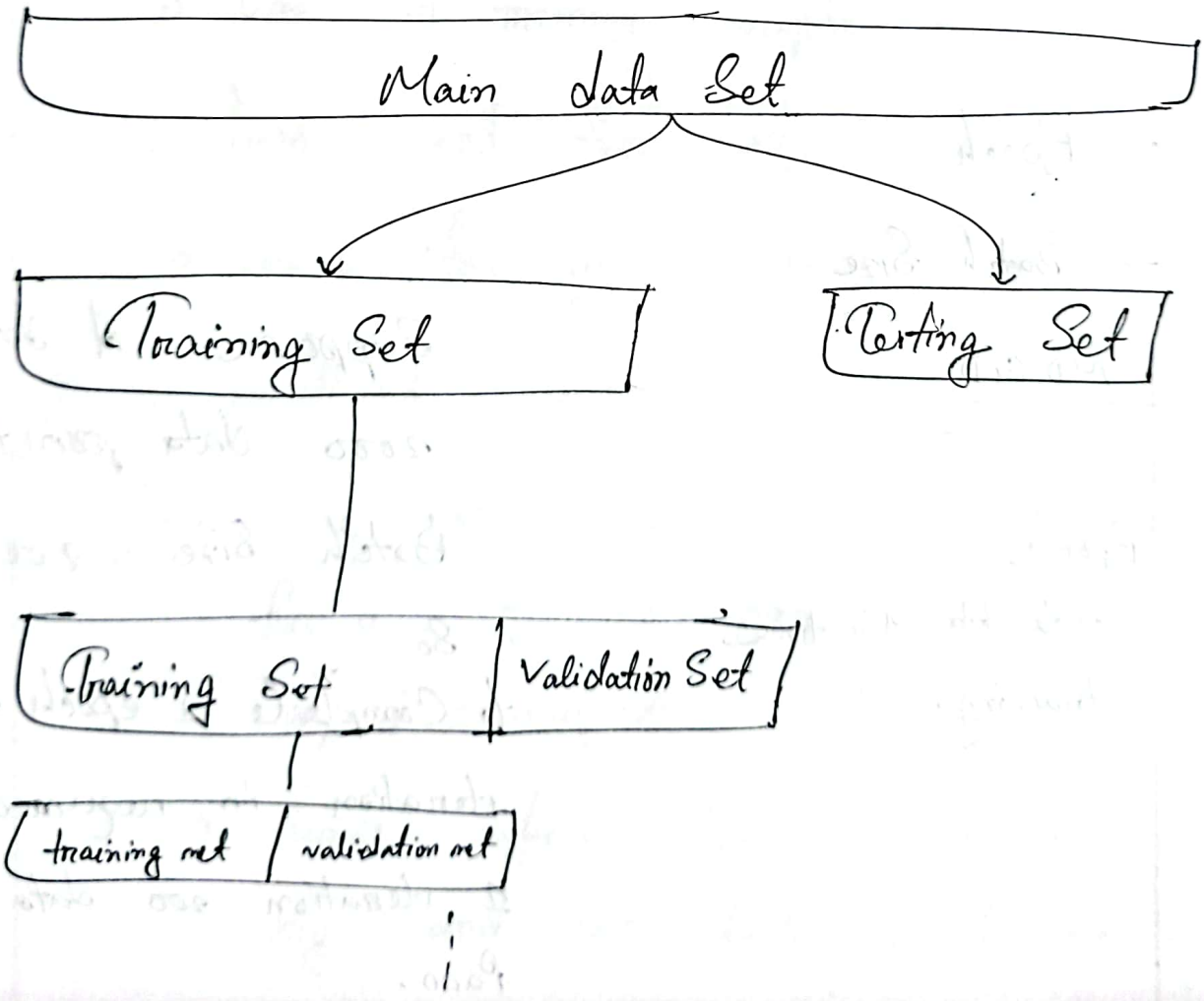
to Complete 1 epoch = 10 iteration in required.

1 iteration 200 data in pass.

→ Training Set : Used to train the model.

→ Validation Set : Used to optimize model parameters.

→ Testing Set : Used to get an "unbiased" estimate of the "final model" "performance".



Adagrad Optimizer

Adaptive gradient descent.

$$W_{new} = W_{old} - \eta \left(\frac{d\,loss}{d\,W_{old}} \right)$$

$$W_t = W_{t-1} - \eta \left(\frac{d\,loss}{d\,W_{t-1}} \right)$$

↳ This learning rate used to be same in GD, SGD, MB-GD, for all the neurons for all the hidden layers.

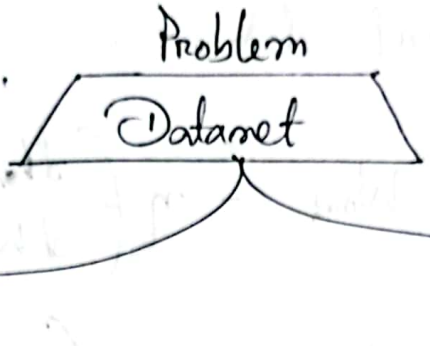
η - learning rate dynamically change.

In adaptive GD, "learning rate" should be different

① for each and every neuron.

② . . . . . hidden layer.

③ based on different iteration.



Solve by (NN)

Dense feature

→ Most of the feature are non zero.

Sparse feature [B.o.W]

→ Most of the feature/value are → 0 (zeroes).

↓

here we can't use same learning rate?

Initialize $\eta = 0.01$;

Adaptive optimizer main concept in that.

Can we use different learning rate with respect to.

Different feature,
Weight,
layer,
Neuron,
iteration.

Dense/Sparse feature ← use different learning rate (Adoptive)

Adagrad Optimizer formula,

$$W_t = W_{t-1} - \eta_t' \left(\frac{d\,loss}{d\,W_{t-1}} \right)$$

$$\eta_t' = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$ → ~~Constant~~ learning rate.
initial.
0.01

Epsilon always small positive number.

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{d\,loss}{d\,W_i} \right)^2$$

$t = 3$ [3rd iteration] First calculate α_t.

$t = 2$

$t = 1$

α_t will high number.

So, η_t' will be small

when, $\alpha_t \uparrow$

η_+^1 will be low:

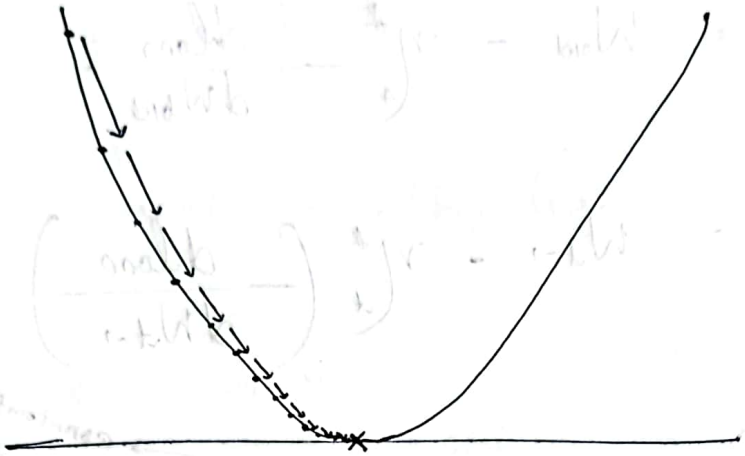
updation.

So, Weight will be decreasing slowly. and it will convergence very nicely.

first, little bigger → (learning rate)

then small → Smaller → Smaller 2 ---

So, Adagrad optimizer changing the learning rate slowly and reach the convergence point very nicely.



One ① inadvantages.

As the number of iteration increases

Some point of time α_t is very high. [$w_{new} \approx w_{old}$]

The Solution : Adadelta and RMS Prop optimizer.

Prevent α_t higher value.

"define iteration step"

Adadelt and RMSProp

$$W_{new} = W_{old} - \eta_t^* \frac{d\text{loss}}{dW_{old}}$$

$$W_t = W_{t-1} - \eta_t^* \left(\frac{d\text{loss}}{dW_{t-1}}\right)$$

$$\eta_t^* = \frac{\eta}{\sqrt{W_{avg(t)} + \varepsilon}}$$

η → constant value dynamically change. initially assign some.

$W_{avg(t)}$ → exponential moving average.

$$W_{avg(t)} = \gamma \cdot W_{avg(t-1)} + (1-\gamma)\left(\frac{dl}{dW_t}\right)^2$$

$$\gamma = 0.95 \quad (\text{most cases})$$

→ Here not summation, so we restrict it restrict the number (t).

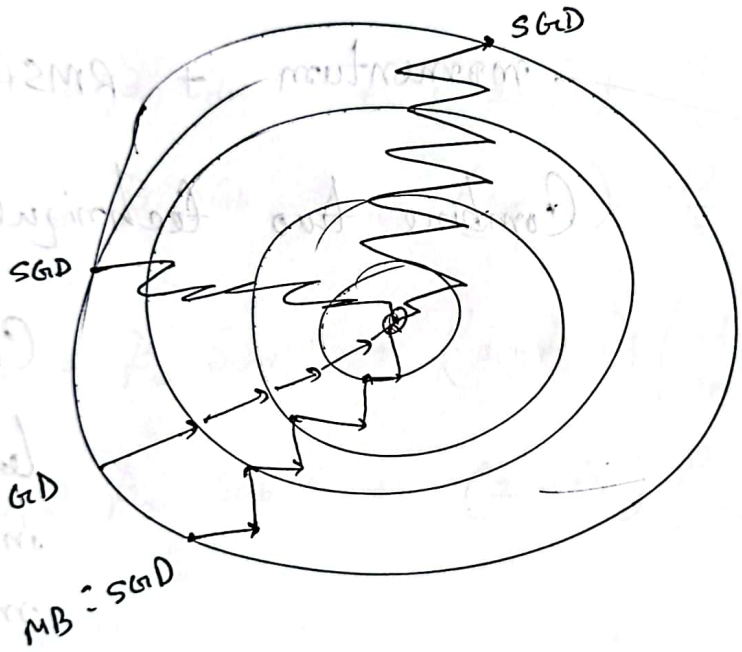
$$\eta_t' = \frac{\eta}{\sqrt{SdW_t + \varepsilon}}$$

SdW_t → exponential moving avg.

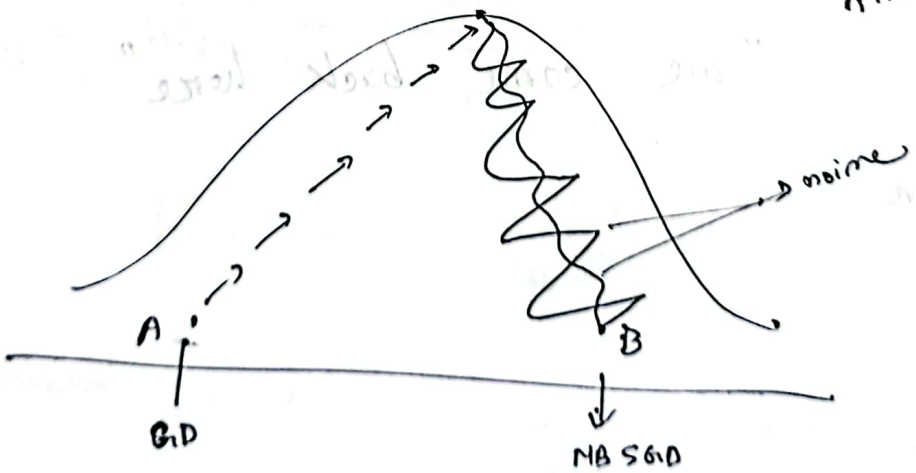
$\beta SdW_{t-1} + (1-\beta)\left(\frac{dl}{dW_t}\right)^2$ } Same

→ Iteration = $\frac{\text{Dataset size}}{\text{Batch Size}}$

Gradient Descent ← Wikipedia.



Reduce noise with momentum.



check it in last

Adam optimizer.

Best optimizer algorithm → "Adaptive Moment Estimation"

Adam optimizer = Gradient Descent with momentum + RMSProp

momentum → Smoothing

Combine two technique.

RMSProp → Change the learning rate in a efficient manner.

"we come back here"

~~Adam~~

Momentum, RMSProp

$V_{dw}, V_{db} = 0$ $S_{dw}, S_{db} = 0$ [initially]

On iteration t, compute $\frac{dl}{dw}$, $\frac{dl}{db}$ using the [mini batch]

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)\frac{dl}{dw}$$
$$V_{db} = \beta_1 V_{db} + (1-\beta_1)\frac{dl}{db}$$
} Momentum.

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\left(\frac{dl}{dw}\right)^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\left(\frac{dl}{db}\right)^2$$
} RMSProp

$$W_{new_t} = W_{t-1} - \frac{\eta * V_{dw}^{correction}}{\sqrt{S_{dw}^{correction} + \epsilon}}$$

η ← initial learning rate.

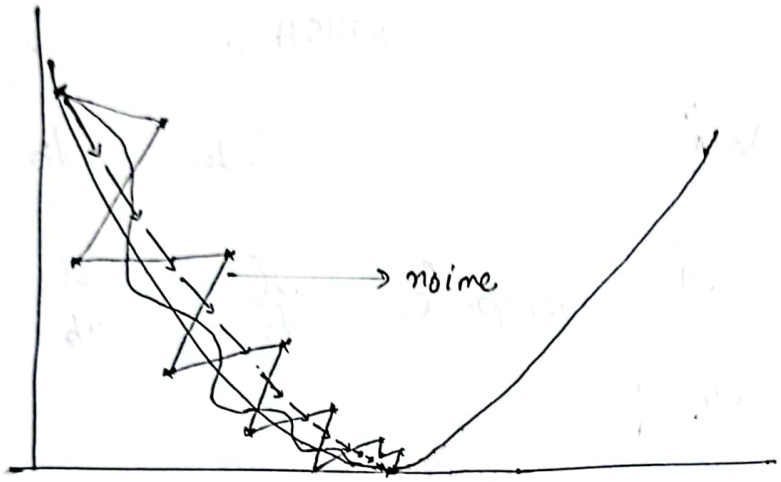
$$b_t = b_{t-1} - \frac{\eta * V_{db}}{\sqrt{S_{db}^{correction} + \epsilon}}$$
} Adam

Bias Correction

$$V_{dw}^{correction} = \frac{V_{dw}}{(1-\beta_1^t)} \qquad S_{dw} = \frac{S_{dw}}{(1-\beta_2)}$$

$$V_{db} = \frac{V_{db}}{(1-\beta^t)} \qquad S_{db} = \frac{S_{dw}}{(1-\beta_2)}$$

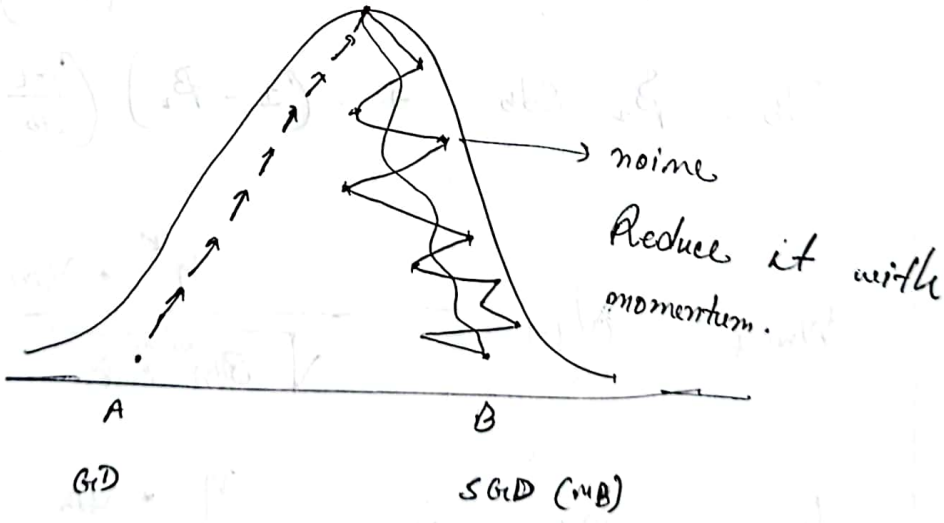
Back in SGD with Momentum.



→ GD

→ MB SGD

↳ apply momentum.



$$W_{new} = W_{old} - \eta \frac{d\,loss}{d\,W_{old}}$$

$$W_t = W_{t-1} - \eta \frac{d\,loss}{d\,W_{t-1}}$$

Instead of ... computing gradient descent, we calculate, Exponential Weighted avarage

→ Let's a time series problem.

Time,	t_1	t_2	t_3	----	t_n
data.	a_1	a_2	a_3	----	a_n

Smoothing Concept.

→ initially

$$V_{t_1} = a_1$$

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

$$= \beta a_1 + (1-\beta) * a_2$$

$$= 0.95 a_1 + (1-0.95) * a_2$$

momentum / velocity / weight decay

→ Learnable parameters / hyperparameter tuning.

If, $\beta = 0.1$

$\beta = 0.95$ most cases.

$$V_{d_2} = 0.1\, a_1 + (0.9) * a_2$$

$$V_{d_3} = \beta V_{d_2} + (1-\beta) * a_3$$

$$= \beta\left(\beta * V_{d_1} + (1-\beta) * a_2\right) + (1-\beta) * a_3$$

Exponential Weighted avarage; weight updation is.

$$W_t = W_{t-1} - \eta \frac{d\,loss}{d\,W_{t-1}}$$

$$= W_{t-1} - \eta \cdot V_{dw} \longrightarrow ?$$

For bias

$$b_t = b_{t-1} - \eta\, V_{db} \longrightarrow ?$$

$$V_{dw} = \beta V_{dw_{t-1}} + (1-\beta) \frac{d\,l}{d\,W_{t-1}}$$

$$V_{db} = \beta V_{db_{t-1}} + (1-\beta) \frac{d\,l}{d\,b_{t-1}}$$

Initially V_{db} and V_{dw} will be zero (0).

Implementation details.

Initially $V_{dw} = 0 \qquad V_{db} = 0$.

On iteration t inside epoch.

Compute dw, db on the current mini-batch.

updation:

$$V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) \frac{d\text{Loss}}{dw_{t-1}}$$

$$V_{db_t} = \beta V_{db_{t-1}} + (1-\beta) \frac{d\text{Loss}}{db_{t-1}}$$

$$w = w_{t-1} - \eta V_{dw_{t-1}}$$

$$b = b_{t-1} - \eta V_{db_{t-1}}$$

Smoothing of convergence.

SGD with momentum.

Adadelta implementation details (explain before)

- On iteration t in minibatch Compute

$$\frac{\partial loss}{\partial w}, \quad \frac{\partial loss}{\partial b}$$ on current mini-batch.

$$S_{dw} = \beta S_{dw} + (1-\beta)\left(\frac{\partial l}{\partial w}\right)^2$$

$$S_{db} = \beta S_{db} + (1-\beta)\left(\frac{\partial l}{\partial b}\right)^2$$

$$w_t = w_{t-1} - \eta_t' \frac{\partial l}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta_t' \frac{\partial l}{\partial b_{t-1}}$$

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

$$= \frac{\eta}{\sqrt{S_{db} + \varepsilon}}$$

Loss function in Deep learning.

→ Loss function / Cost function.

Error for 1/Single record. like in SGD first pass single record and calculate the error then backpropagate.

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

→ In Mini Batch gradient descent. we pass batch of record, or in gradient descent pass whole record. then we calculate loss function called.

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

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

Check Keras losses

⇒ Regression Problem losses.

① Squared Error loss

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

② Mean Square Error

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

/ Actually cost $= \sum_{i=1}^{n} (y - \hat{y})^2 * \frac{1}{n}$

③ Mean Absulate Error loss

$$loss = |y - \hat{y}|$$

$$= abs(y - \hat{y}) * \frac{1}{1}$$

$$= \sum_{i=1}^{n} abs(y - \hat{y}) * \frac{1}{n}$$

④ Huber loss

Combine MSE + MAE

↳ Quadratic equation.

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

↳ linear equation.

δ ⇒ delta value, hyperparameter.

⇒ Classification problem losses,

• Cross Entropy

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

↳ for Binary cross entropy.

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

⇒ Solve only Binary classification

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

⇒ Multiclass Cross entropy loss. → category

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

O/P → Multiclass.

Good, bad, Neutral.

f_1	f_2	f_3	O/P	y_{i1} y_{i2} y_{i3} y_1 y_2 y_3
2	4	2	Good	[1, 0, 0]
5	6	9	bad	[0, 1, 0]
3	2	5	neutral	[0, 0, 1]

3 Bits

→ That we called "Categorical Cross Entropy".

y_i is one hot encoded target vector.

$y_i = [y_{i1}, y_{i2}, \ldots, y_{in}]$ i. Row number.

y_{ic} → category.

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

$\hat{y}_{ij}$ → use Softmax activation function.

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

⇒ Categorical cross entropy vs Sparse categorical cross entropy.

⇒ both have same loss function,

$$J(w) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}) + (1-y) \log(1-\hat{y}_i) \right]$$

y_i = true value

$\hat{y}_i$ = predicted value

w = weight.

⇒ if your Y_i are one hot encoded, use categorical_crossentropy. Examples,

for a 3 class classification:

[1, 0, 0], [0, 1, 0], [0, 0, 1]

⇒ if your Y_i are integers, use sparse_categorical_crossentropy. Example for 3 class classification,

[1], [2], [3]..

Most Common loss function.

① Probabilistic / classification losses.

→ Binary Crossentropy class.

→ CategoricalCrossentropy class.

→ SparseCategoricalcrossentropy class.

② Regression losses.

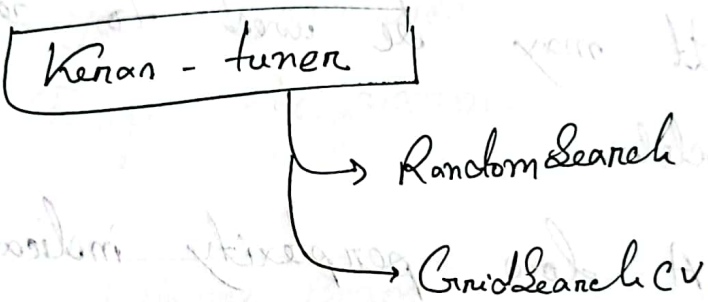
→ Meansquared error.

→ Mean Absolute error.

→ Huber class.

Hyperparameter tuning

1. How many numbers of hidden layer we should have?
2. How many numbers of neurons each hidden layers?
3. What should be the learning - Rate.



→ Go to the keras tuner site.

→ Embeding dimension.

→ Dense function → create neuron.

Dense layer ↔ Dense function

Keras/api/layers/core_layers/dense.py

Best understanding.

→ Perplexity is a measurement of how well a probability distribution or probability model predicts a sample.

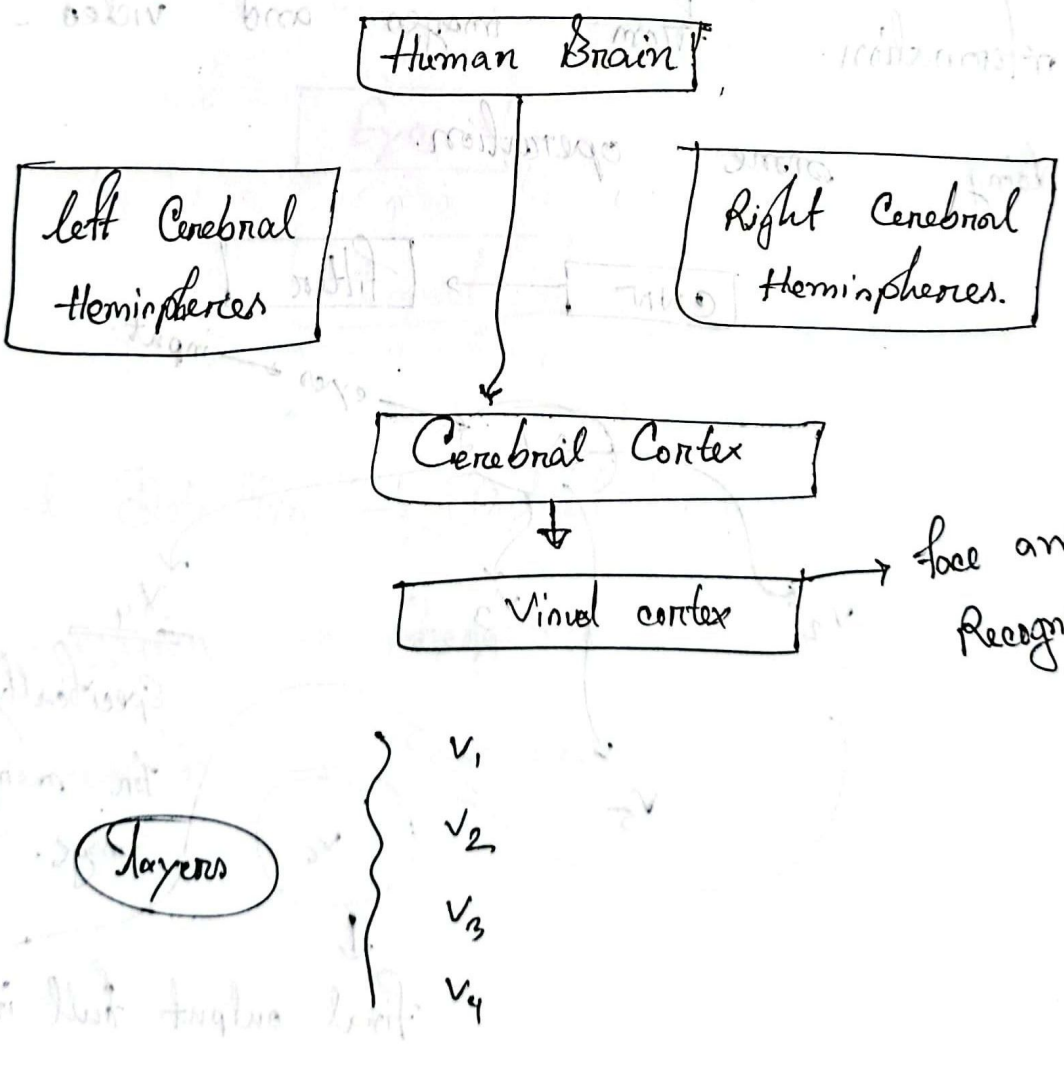
→ It may be used to compare probability models.

→ A low perplexity indicates the probability distribution is good at predicting the sample.

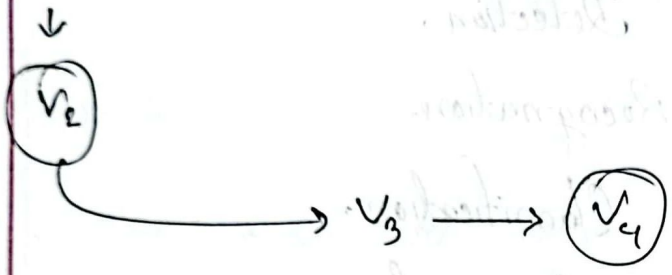
Convolution Neural Network.

Input format : Image, video.

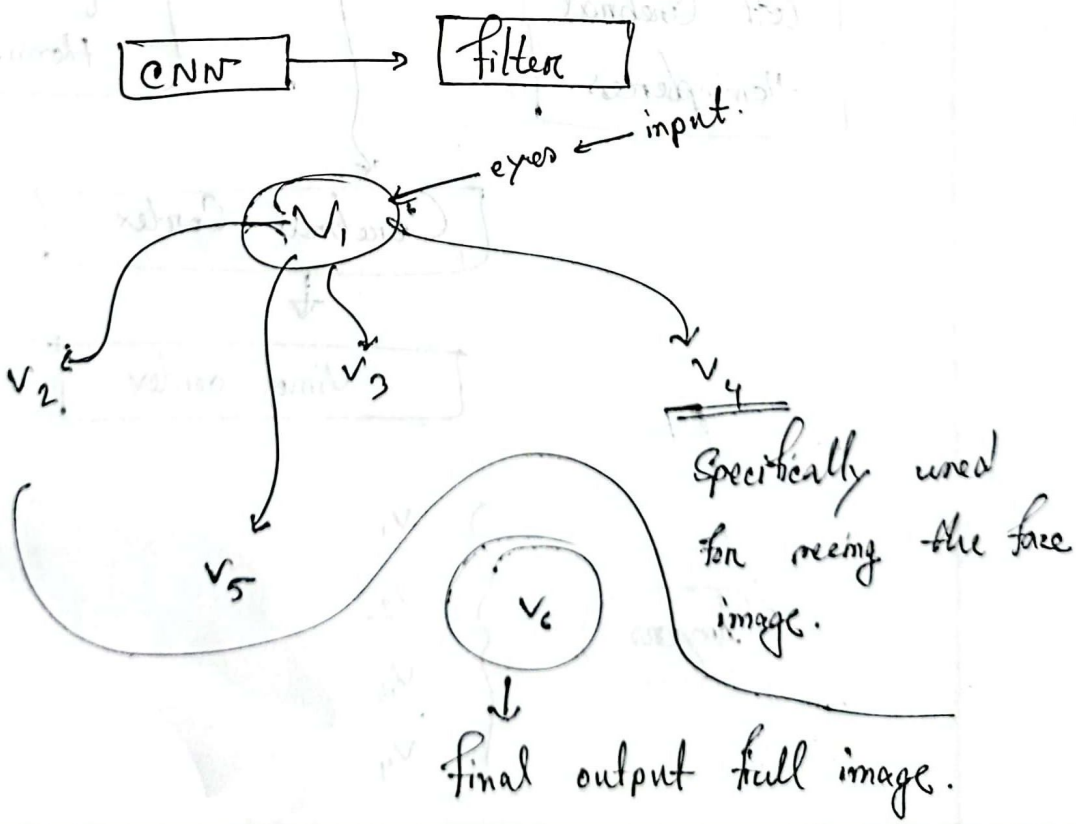
-> Object Detection.
-> face Recognation.
-> Object Classification.
-> Image Recognation.

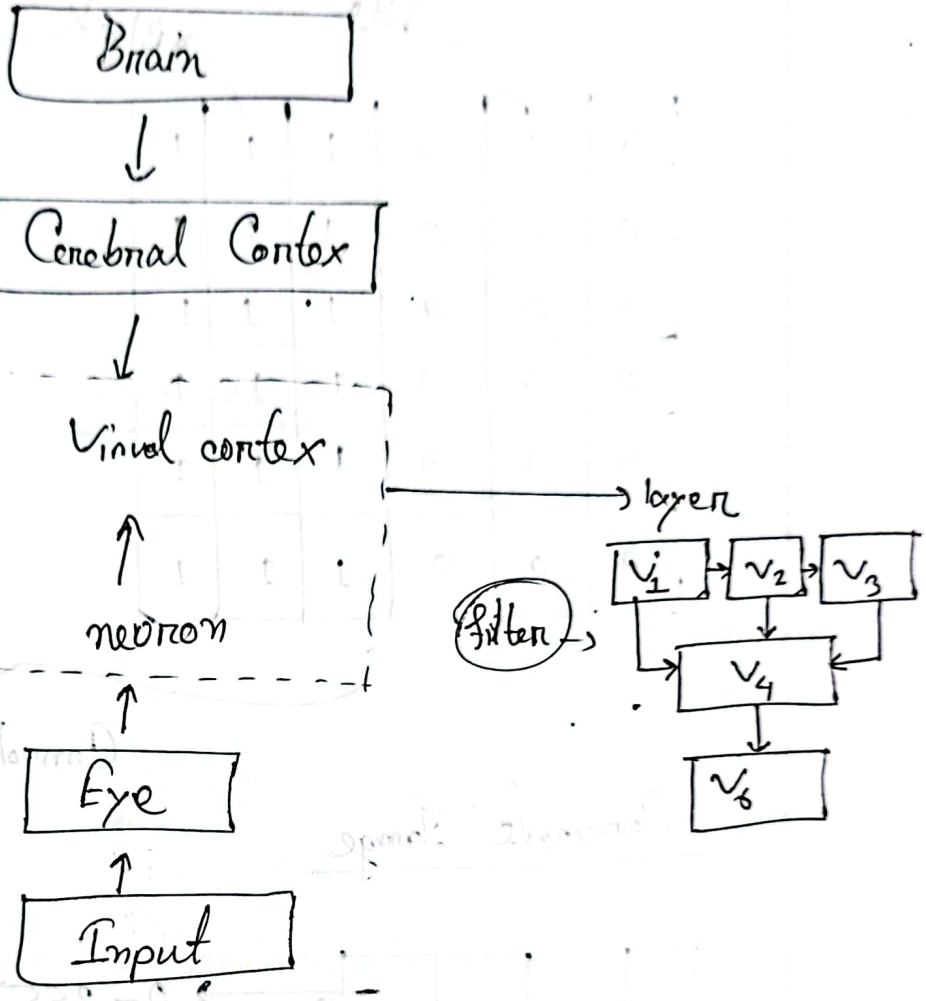


V_1 layer may be responsible for finding the edges of images (like cat, dog)



Each layers is responsible for Extract the information from images and video after doing some operation.

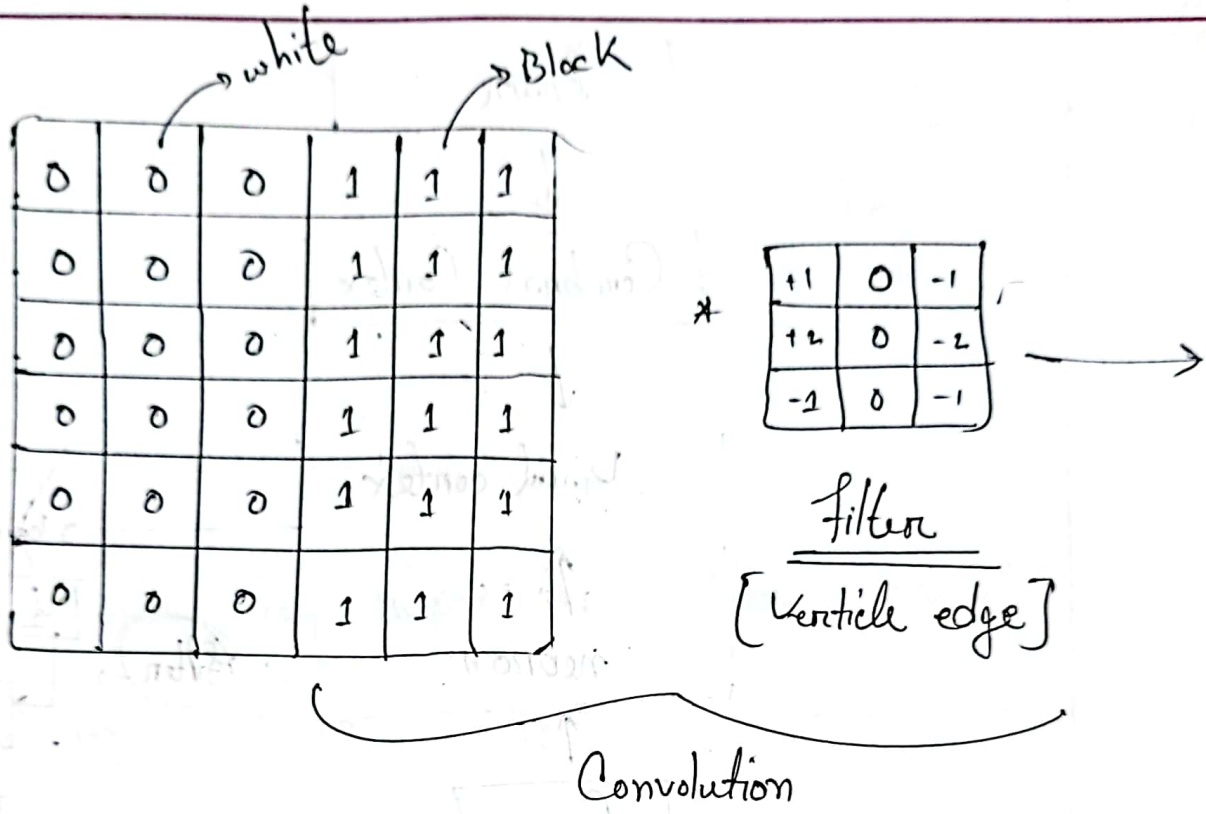




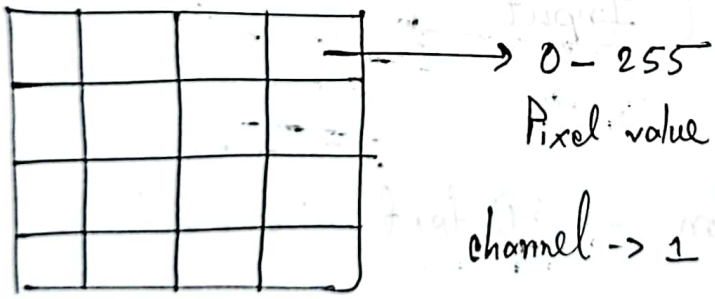
Object Detection → Detect

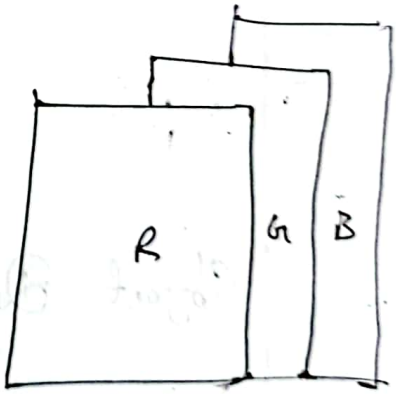
Recognization → Recognize.

Segmentation → Segment.



Grayscale Image





0 – 255

↓

normalize (0-1) if we use minmax.

Vertical edge filter or detector

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

+ve

-ve

Horizental edge detection,

+ve

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

-ve

input matrix * filter matrix = ?

$$n_{out} = \left[\frac{n_{in} + 2P - K}{S} \right] + 1$$

n_{in} = number of input feature / input size

n_{out} = number of output feature / output size

P = Padding Size

K = filter size / kernel size

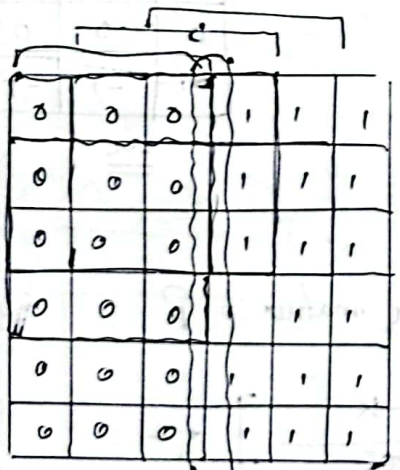
S = Stride size.

CNN

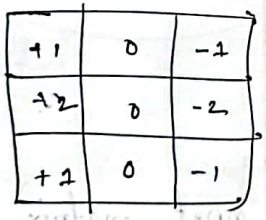
→ Convolution layer.

→ Pooling layer.

→ fully connected layer.



6×6



3×3

Vertical edge detection.

→ Thin edge

if Stride = 1

1 step forward - - - -

1 step down - - - - -

$$= \frac{6+0-3}{1} + 1$$

$$= 4$$

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

4×4

→ normalize

minmax

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

Black or Dark side

White side

Dark side

Different filter detect the different edge / different feature.

Padding in CNN

input image * filter = output

6×6 * 3×3 = 4×4

here losing some amount of data/info.

$6 \times 6 \quad * \quad 3 \times 3 \quad = \quad 4 \times 4$

but it should be 6×6, because of input size 6×6

$$n_{out} = \left(\frac{n_{in} + 2P - k}{s} \right) + 1$$

$$6 = \left[\frac{n_{in} + 2P - k}{1} \right] + 1 \qquad \text{[if i want output } 6 \times 6\text{]}$$

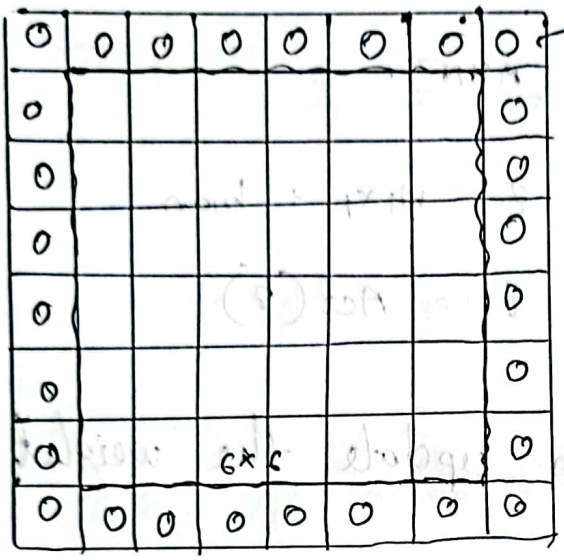
$n_{in} = 6 \times (n_{in}$

$n_{in} + 2 \times 0 - 3 = 5$

$n_{in} = 5 + 3$

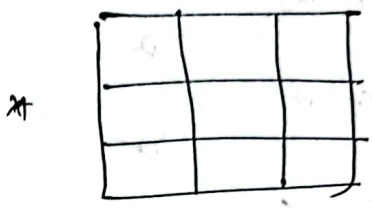
$n_{in} = 8$

[input size should be $(6 \times 6) \rightarrow 8 \times 8$ do it by padding]



→ 0 padding.

8×8



= 6×6

$$n_{out} = \frac{6 + 2 - 3}{1} + 1$$

$$= 6$$

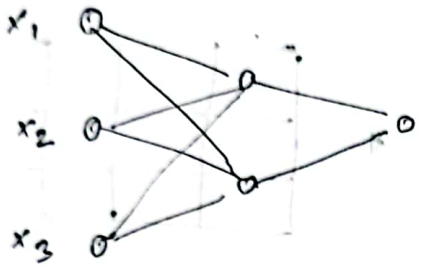
input → 6×6 → by padding (P = 1) → 8×8

output

6×6

0 padding means fill every pixel with 0.

ANN - CNN



ANN

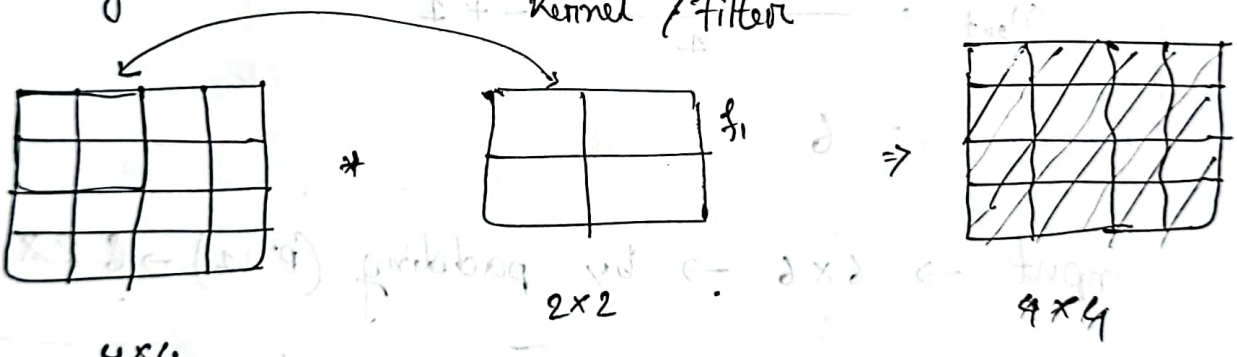
$$z = w_i x_i + bian$$

$$z' = Act(z)$$

In Backpropogation update the weight.

Image

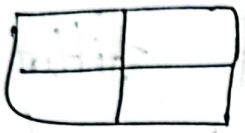
Kernel / filter



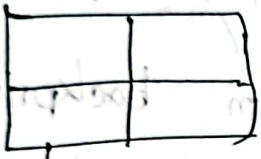
$$n_{out} = \frac{4+0-2}{2} + 1$$

$$= 3$$

filter 2

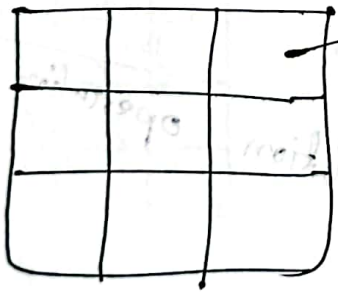


filter 3



filter 4

after every filter the output



apply activation function. (relu) each every field.

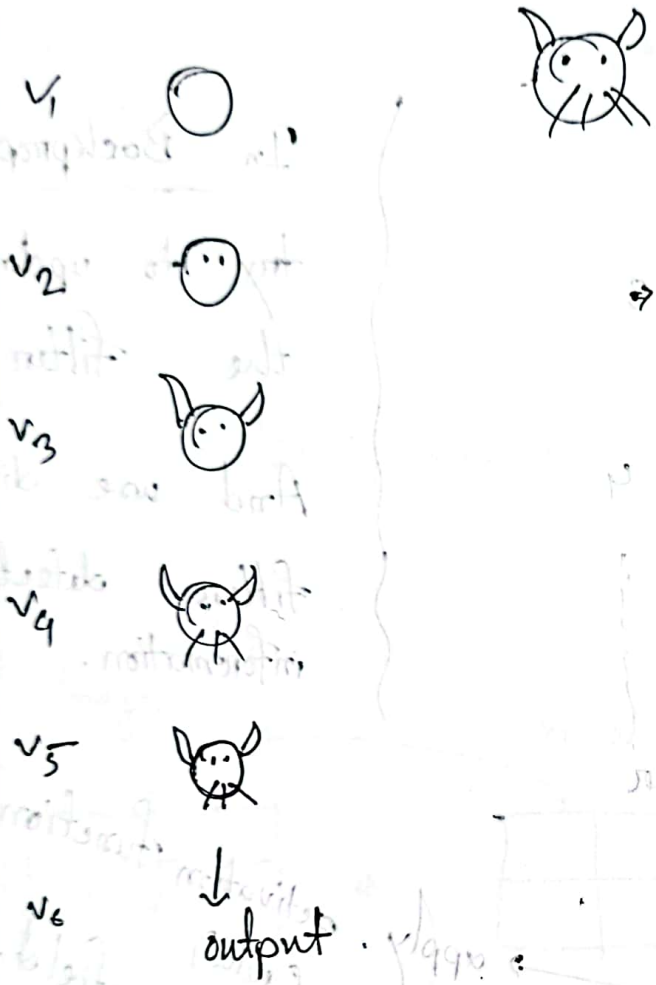
In Backpropagation

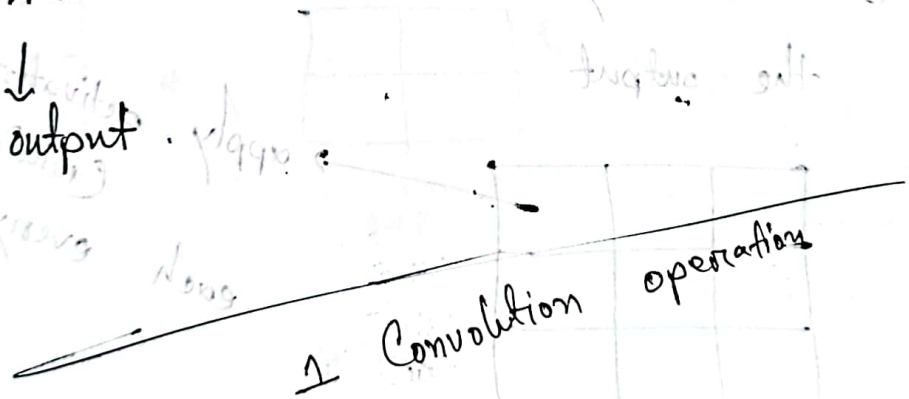
try to update the filter value.

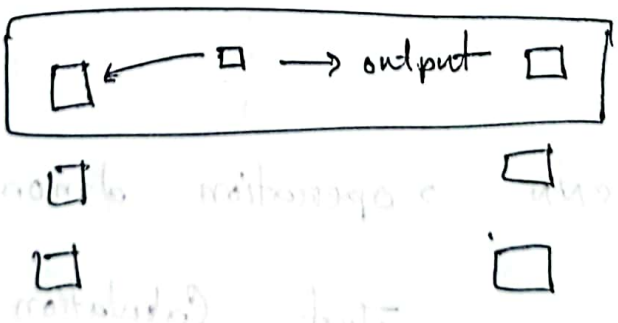
And use different filter detect different information.

ANN - CNN → operation almost same (Process)

Just Calculation is diff.







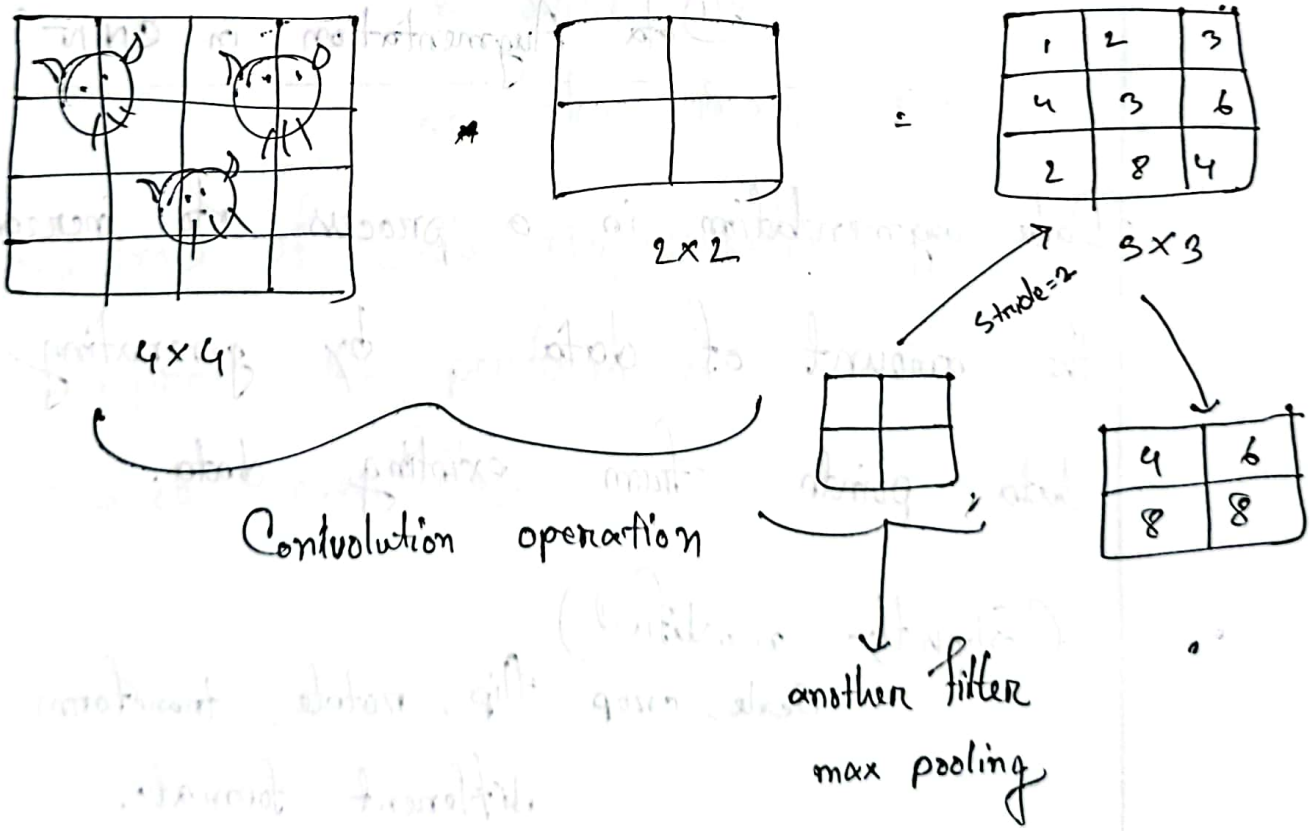
→ in first filter value randomly selected and in backpropagation kernel / filter are updated.

Term: location invariant

When human Brain see some faces some of the neuron automatically triggered.

→ Convolution NN do that / done by

Max Pooling layer



Max pooling → clearly / more precisely detect.
(maximum intensity value)
→ Properly detected.

There is also **Average pooling**

Data Augmentation in CNN

Data augmentation is a process to increase the amount of data by generating new data points from existing data.

(Slightly modified)
Scale, crop, flip, rotate transform different formats.

"Synthetic data": When data in generated artificially without using real world images. Produced by (GAN)

"Augmentation data": Derived from "orginal images" with some sort of minor geometric transformation like (flipping, translation, rotation, or the addition of noise) in order to increase the diversity of the traing image.

"Importance of Data Augmentation"

⇒ Improve the performance.

⇒ Reduce overfitting.