

History of deep learning:-

- i) 1950s:- The question "why don't we mimic the human brain into a computer". The field of AI emerged. Researchers started exploring this field to create intelligent machines. one of the Researchers named "Frank Rosenblatt" developed a simple neural network known as "perceptron" which could learn simple tasks.
- ii) 1960-1980's:- This period is known as AI-Winter; because due to the lack of Computational power researchers unable to train the neural network. Although it trained they are unable to improve the accuracy.
- iii) 1990's:- The concept of backpropagation was discovered by researchers. ~~B~~ which allows neural network to learn by adjusting the weights. This led renewed interest in neural network and its applications.
- iv) 2000's:- Computational advancements, CNN's was developed by geoffrey hinton.

* started exploring on multilayered deep neural network. This allowed applications like image classification.

=
* 2010s:- huge advancements on hardware, larger datasets, novel architectures. Breakthrough like the development of GAN by "Geoffrey Hinton".

And the success of deep learning the image recognition competitions held the fuel the rapid growth of the field.

present:- wide applications:- It achieved remarkable

results in image, speech recognition, NLP, autonomous vehicles, healthcare, finance and more. Deep learning models such as

RNN's (Recurrent neural networks), generative adversarial network (GANs), and transformer

networks have further expanded the capability of deep learning.

DEEP LEARNING

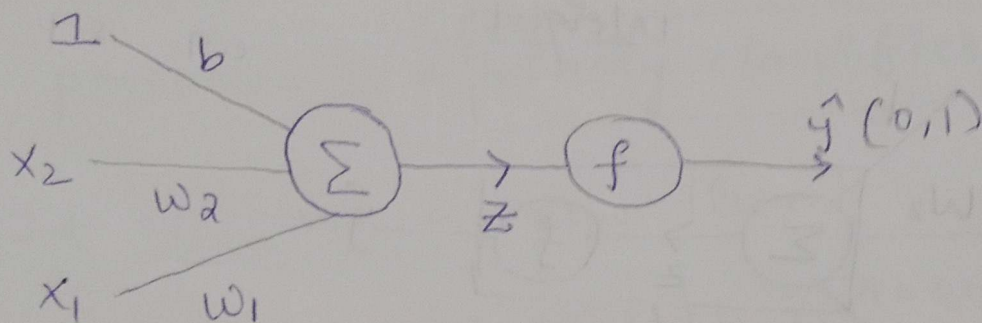
What is perceptron?

* It is an algorithm (like linear regression, RForest)

This is used for supervised ML.

* Its design is there in such a way that it became the building block of deep learning.

* It is also called as mathematical model or function



$$z = x_1 w_1 + x_2 w_2 + b \text{ (summation)}$$

f = Activation function (eg: step, sigmoid, relu)

→ f range: $[-1 \text{ to } 1]$ or $[0 \text{ to } 1]$

Objective:- By using the training data we should find the values of w_1 , w_2 and bias.

Then we can use the test data to predict

Neuron Vs perceptron:

Dendrites = inputs

neucleus = Activation $f^n(z)$

Axon = output

History of Neural Networks:

→ Perceptron was built by Rosenblatt's in 1957

* Alan Turing was the father of modern computing

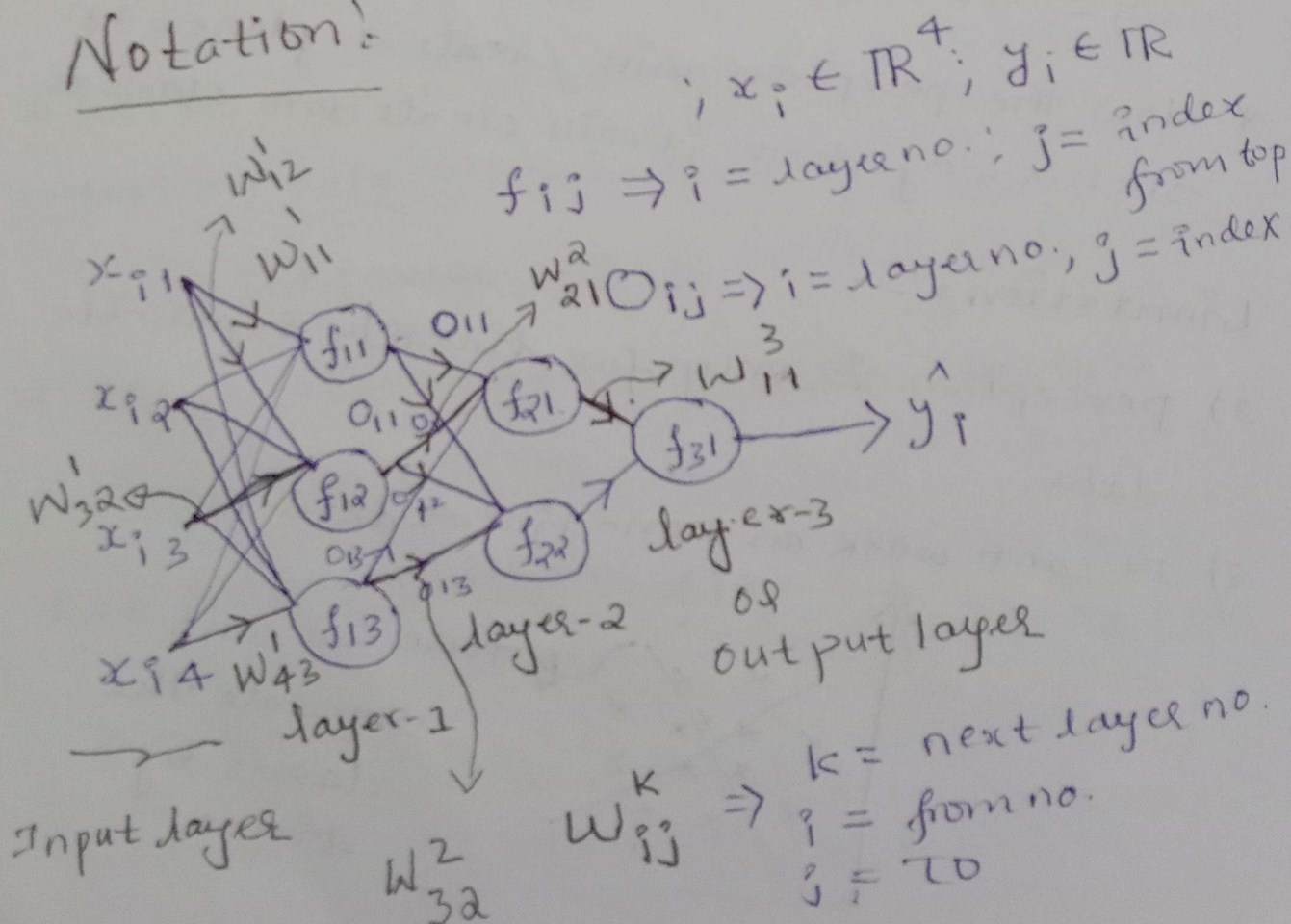
* For all of these have biological inspiration.

* Neuron is a basic building

MLP = graphical way of representing
f o f, g o f

= They can also easily overfitting.

Notation:



* Hence it is called a fully connected MLP.

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix}_{4 \times 3}$$

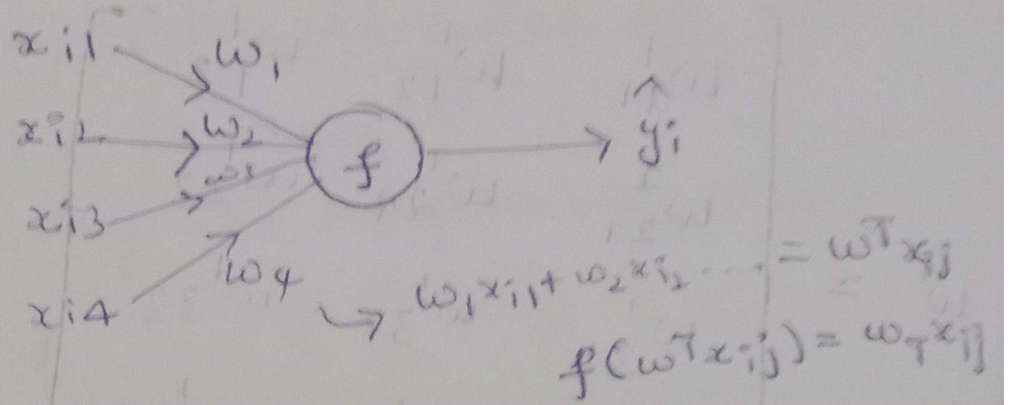
||y there $\exists w_{3 \times 2}^2 \text{ \& } w_{2 \times 1}^3$

Train a single neuron model:

↓
finding the best edge weights using training data. (D_{train})

- * perceptron and Logistic Regression are single neuron models for classification.
- * Same Linear Reg is also a single neuron model for regression.

$$D_T = \{x_i, y_i\} \quad ; \quad x_i \in \mathbb{R}^{\text{dimension}} \quad ; \quad y_i \in \mathbb{R}$$



In Linear Reg.

$$\hat{y}_i = \sum_{j=1}^n w_j x_{ij} = w^T x_i$$

then $f(\cdot)$ is called Identity function

Result: In case of Linear Reg function acts as Identity function.

* In case of Logistic Reg f act as

$f = \text{logistic or sigmoid}$

To solve an optimization problem:-

① Define the Loss function

$$L = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \text{reg}$$

$$L = (y_i - \hat{y}_i)^2 \quad (\text{ignore reg})$$

② Define optimization function

As $x \in \mathbb{R}^4 : w \in \mathbb{R}^4$

$$w^* = \arg \min_w \sum_{i=1}^n (y_i - f(w^T x_i))^2 + \text{reg}$$

$$\therefore w^T x_i = w_1 x_{i1} + w_2 x_{i2} + \dots$$

* If f is identity then $\Rightarrow$ Linear optimization fn

* If f is logistic or sigmoid $\Rightarrow$ logistic optimization

* If f is threshold function like

$$f = \begin{cases} 1 & w^T x_{ij} \geq 0 \\ 0 & \text{else} \end{cases}$$

* is a perceptron activation optimization function.

③ Solve the optimization problem

① Randomly initialize the weights

$$\textcircled{b} (\nabla L)_w = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_k} \end{bmatrix} \quad \begin{matrix} \text{As } x_i \in \mathbb{R}^4 \\ L \in \mathbb{R}^4 \end{matrix}$$

$$\textcircled{c} w_{\text{new}} = w_{\text{old}} - \eta \left[\nabla_w L \right] w_{\text{old}}$$

$$(w_i)_{\text{new}} = (w_i)_{\text{old}} - \eta \left[\frac{\partial L}{\partial w_i} \right]_{(w_i)_{\text{old}}}$$

update eqⁿ

For eg:- Let us compute $(w_1)_{\text{new}}$

$$\therefore (w_1)_{\text{new}} = (w_1)_{\text{old}} - \eta \left[\frac{\partial L}{\partial w_1} \right]_{(w_1)_{\text{old}}}$$

$$\left[\frac{\partial L}{\partial w_1} \right]_{(w_1)_{\text{old}}} = \frac{\partial L}{\partial f} \times \frac{\partial f}{\partial w_1}$$

$$L = \sum_{i=1}^n (y - f)^2; \quad f = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + w_4 x_{i4}$$

$$f(w^T x_{ij}) = \hat{y}$$

$$\therefore \frac{\partial L}{\partial f} = -2(y - f); \quad \frac{\partial f}{\partial w_1} = x_{i1}$$

$$\therefore \frac{\partial L}{\partial w_1} = -2x_{i1}(y - f(w^T x_{ij}))$$

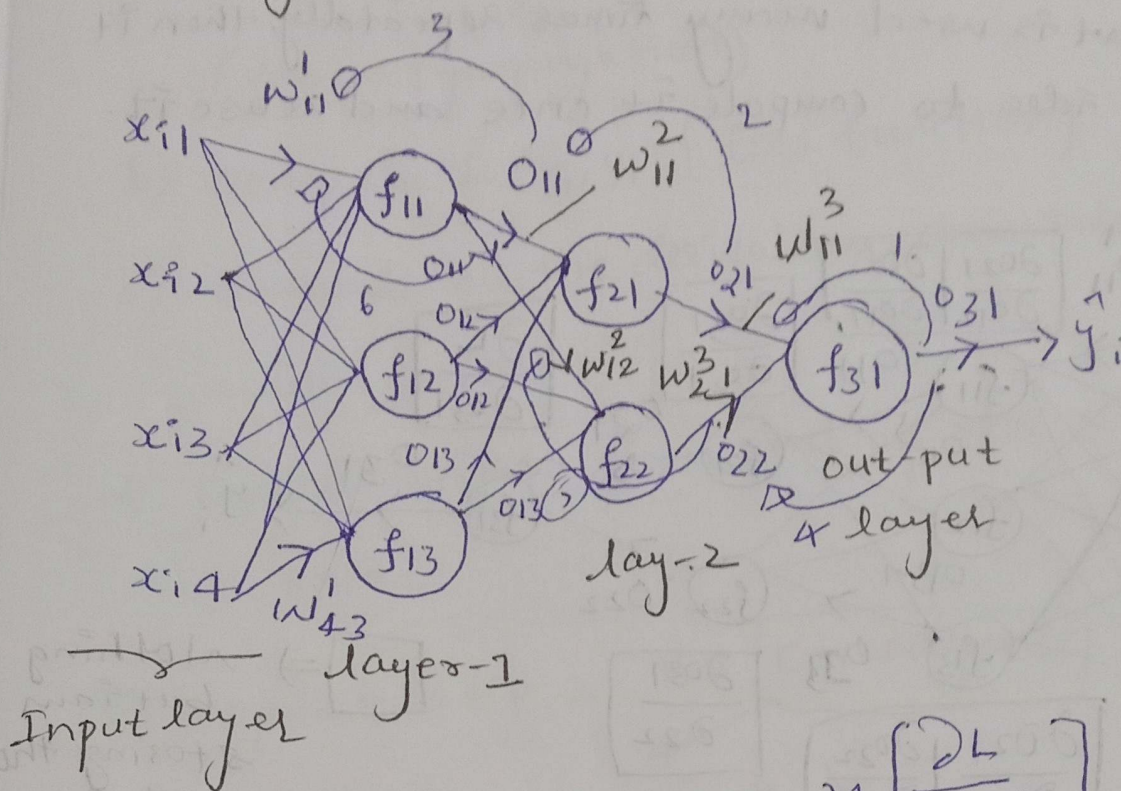
* The above update eqⁿ can be solved by using:

1) Gradient descent, where we take all the train data in one iteration

2) SGD: $\left\{ \nabla_w L \approx \text{one pt of } x_i, y_i \right\}$

Batch SGD $\left\{ \begin{array}{l} \text{we take batches of} \\ \text{points in 1 iteration} \end{array} \right\}$

Training a multi neuron model



$$(w_{11}^2)_{\text{new}} = (w_{11}^2)_{\text{old}} - \eta \left[\frac{\partial L}{\partial w_{11}^2} \right]_{\text{old}}$$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w_{11}^2}$$

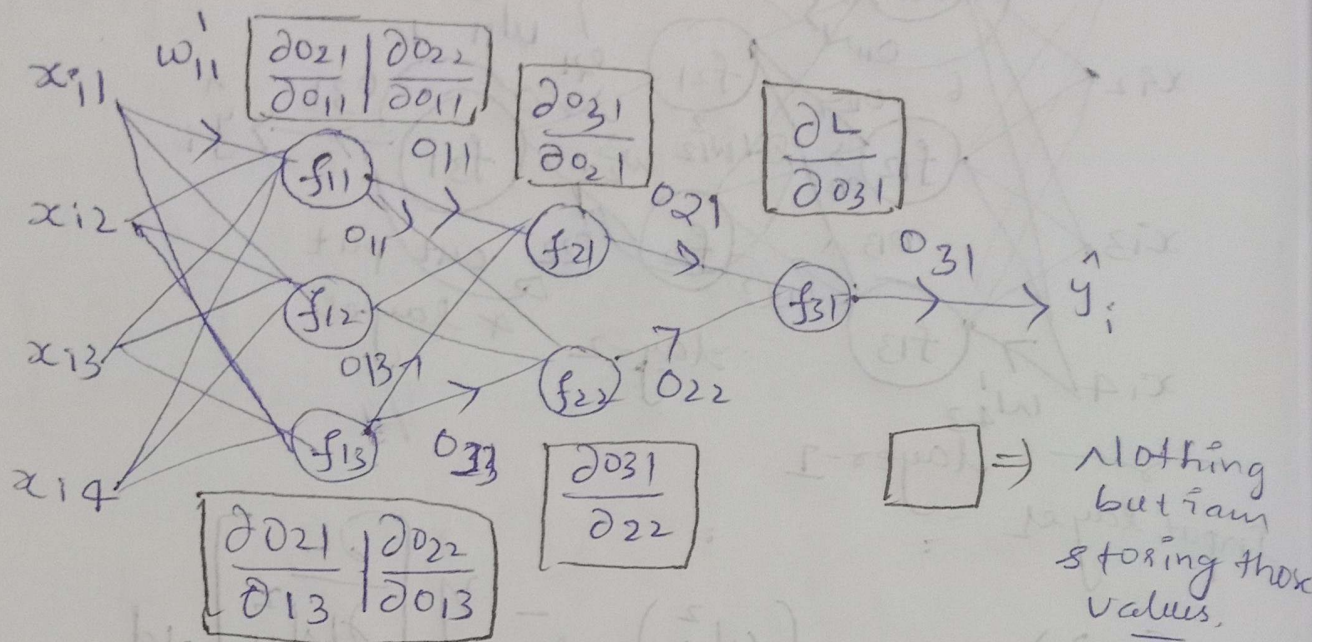
Now find $\left[\frac{\partial L}{\partial w_{11}^2} \right]_{\text{old}}$

$$\therefore \frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial w_{11}^2}$$

Now $\frac{\partial o_{31}}{\partial w_{11}^2}$ = has two paths like 1, 2, 3 $\rightarrow$ 4 $\rightarrow$ 5 $\rightarrow$ 6

$$\therefore \frac{\partial o_{31}}{\partial w_{11}^2} = \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_{11}^2} + \frac{\partial o_{31}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_{11}^2}$$

Memoization; If there is any operation that is used many times repeatedly then it is good idea to compute it once and reuse it.



* Memoization is very imp technique for design and analysis of algorithms.

Backpropagation; It is a very simple mathematical trick of using chain rule of differentiation and memoization, and with some optimizations sprinkled in.

Let $D = \{x_i, y_i\}$

① initialize w_{ij} 's

② for each x_i in D

- pass x_i forward through the network (forward propagation)
- compute $L(\hat{y}_i, y_i)$ [loss function]
- Compute all the derivatives, chain rule & memoization
- update weights from end of the network to start.

③ repeat step 2 till Convergence

$$(w_{ij}^k)_{\text{new}} \approx (w_{ij}^k)_{\text{old}}$$

epoch: Using all the all the datapoints of a training set D_{train} . In real world we do multiple epoch's for convergence.

In Back prop:-

① Initialize weights

② epoch:- i) forward propa

ii) compute loss

iii) compute derivatives

(chain rule + memoization)

iv) update weights

(back propagation)

* Why one should learn DL?

Do you want teach computer how humans are thinking and how they recognise people!

Don't you want an AI assistant for you to help in daily goals, like in a day you have to shortlist 10 resumes out of 1000.

History of DL:- (How we can provide the intelligence of human to computer)?

* Rosenblatt said that I have created a perception which is very very similar to a single neuron. In 1957

* Cold war: 1945: 1990

It is b/w USA & Soviet Union these ideologies are Capitalism & Democratic. In the war it is imp for analysing the image of a military person & pattern recognition, we say as for military application. US govt encouraged the researchers and funded the development

of high performance computers and computing networks, which eventually facilitated the computational requirements for training and deploying deep learning models.

* During the time the greatest minds like "Alan Turing": father of modern computers and others said "how can we create intelligence in the computer".

Activation functions - Discussed in previous book and had a Jupyter notebook to study

Vanishing Gradient descent

Let's try to update w_{11} weight.

$$(w'_{11})_{\text{new}} = (w'_{11})_{\text{old}} - \eta \left(\frac{\partial L}{\partial w'_{11}} \right)$$

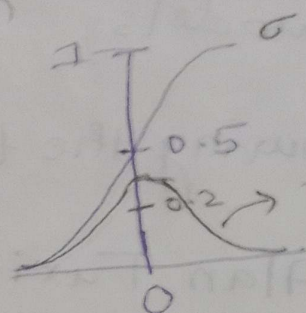
$$\frac{\partial L}{\partial w'_{11}} = \frac{\partial z_1}{\partial z_1} \frac{\partial L}{\partial z_1} \left[\frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w'_{11}} \cdot \frac{\partial L}{\partial z_1} + \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial w'_{11}} \cdot \frac{\partial L}{\partial z_2} \right]$$

In early 80's 90's & 2000's most of them are using sigmoid and tanh as the activation functions.

o_{31} is nothing but f_{31} output

$$\therefore \frac{\partial o_{31}}{\partial a_{22}} = \frac{\partial f_{31}}{\partial o_{21}} \quad \text{--- span}$$

where $f_{31} = \text{sigmoid}$



The output of derivative of sigmoid function

$$\leq 0.25$$

$$\text{Now } \frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial o_{31}} \left[(0.2)(0.1)(0.05) + (0.15)(0.06)(0.1) \right]$$

$$\frac{\partial L}{\partial w_{11}} \leq 10^{-3}$$

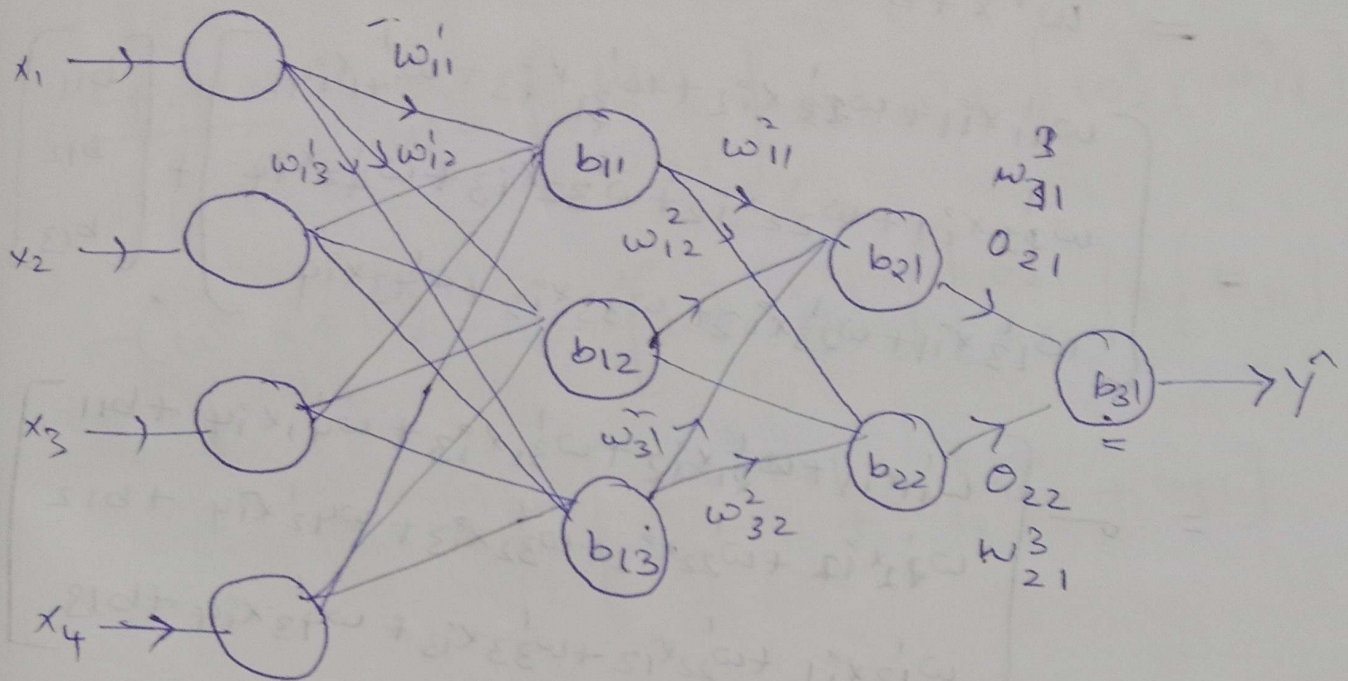
$$\text{Then: } \partial (w_{11})_{\text{new}} = 2.5 - (1)(10^{-3})$$

$$= 2.49$$

almost same as original weight.

Although we have taken only 2 hidden layers means we can call it as a deep nln. But if we 10 to 20 hidden layers. The change almost negligible.

Forward propagation!



perception $\Rightarrow \sigma(w^T x + b)$

$$W^T = \begin{bmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 & w_{41}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 & w_{42}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 & w_{43}^1 \end{bmatrix}_{3 \times 4}$$

$$W^* = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix}_{4 \times 3}$$

$$X = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \end{bmatrix}_{4 \times 1} \quad b = \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}_{3 \times 1}$$

Layer # 1

$$= \omega^T x + b \Rightarrow$$

$$= \begin{bmatrix} \omega_{11}x_{i1} + \omega_{12}x_{i2} + \omega_{13}x_{i3} + \omega_{14}x_{i4} \\ \omega_{21}x_{i1} + \omega_{22}x_{i2} + \omega_{23}x_{i3} + \omega_{24}x_{i4} \\ \omega_{31}x_{i1} + \omega_{32}x_{i2} + \omega_{33}x_{i3} + \omega_{34}x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}$$

$$= \begin{bmatrix} \omega_{11}x_{i1} + \omega_{12}x_{i2} + \omega_{13}x_{i3} + \omega_{14}x_{i4} + b_{11} \\ \omega_{21}x_{i1} + \omega_{22}x_{i2} + \omega_{23}x_{i3} + \omega_{24}x_{i4} + b_{12} \\ \omega_{31}x_{i1} + \omega_{32}x_{i2} + \omega_{33}x_{i3} + \omega_{34}x_{i4} + b_{13} \end{bmatrix}$$

$$= \begin{bmatrix} 0.11 \\ 0.12 \\ 0.13 \end{bmatrix}$$

layer-2 :

$$W = \begin{bmatrix} \omega_{11}^2 & \omega_{12}^2 \\ \omega_{21}^2 & \omega_{22}^2 \\ \omega_{31}^2 & \omega_{32}^2 \end{bmatrix}$$

Now $\sigma(b + W^T x) =$

$$\begin{bmatrix} \omega_{11}^2 & \omega_{21}^2 & \omega_{31}^2 \\ \omega_{12}^2 & \omega_{22}^2 & \omega_{32}^2 \end{bmatrix} \begin{bmatrix} 0.11 \\ 0.12 \\ 0.13 \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}$$

$2 \times 3 \quad \quad 3 \times 1$

$$= \begin{bmatrix} \omega_{11}^2 0.11 + \omega_{21}^2 0.12 + \omega_{31}^2 0.13 + b_{21} \\ \omega_{12}^2 0.11 + \omega_{22}^2 0.12 + \omega_{32}^2 0.13 + b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} 0.21 \\ 0.22 \end{bmatrix}$$

Layer-3

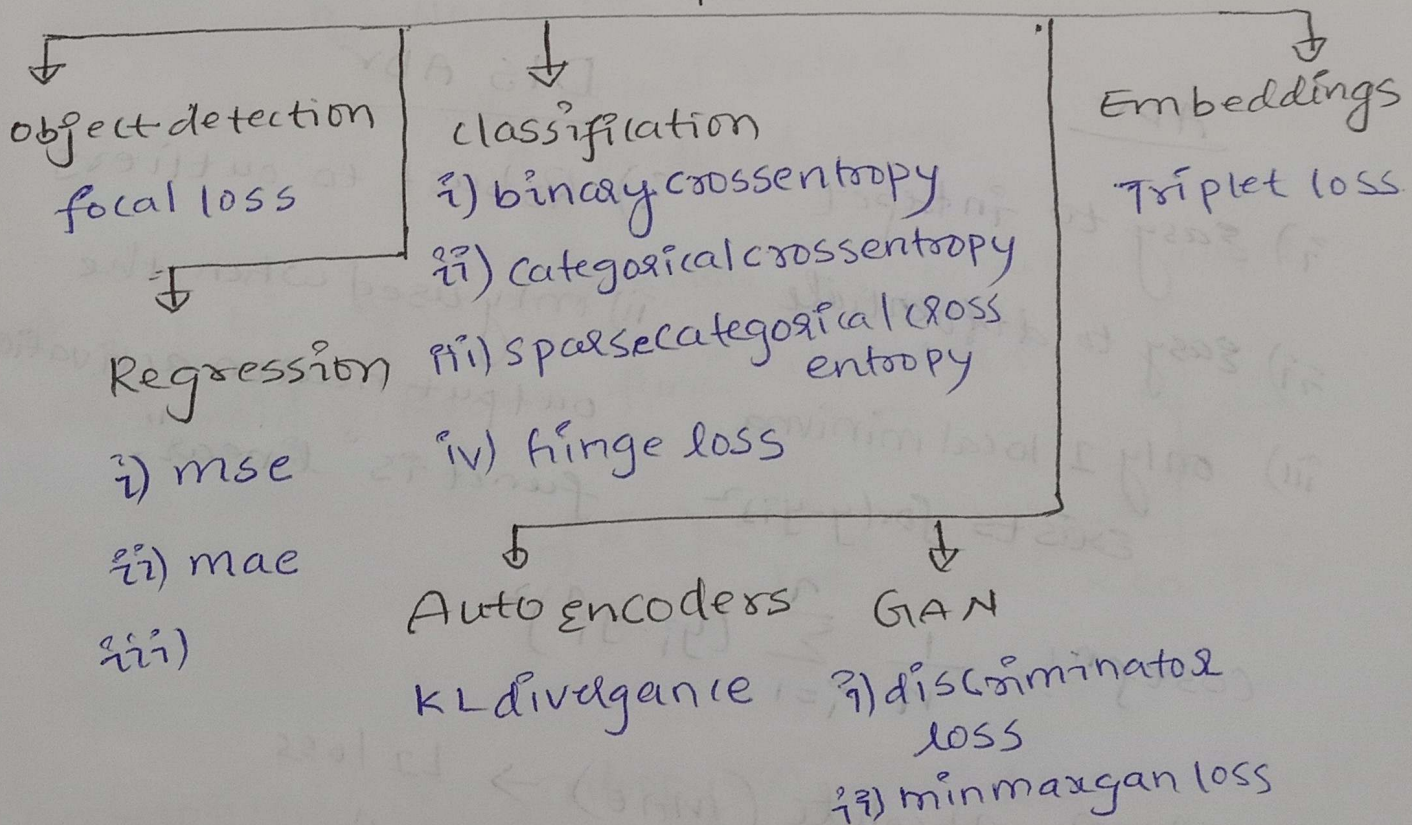
$$W = \begin{bmatrix} w_{11}^3 \\ w_{21}^3 \end{bmatrix} \Rightarrow W^T = \begin{bmatrix} w_{21}^3 & w_{11}^3 \end{bmatrix}$$

$$\sigma(W^T x + b) = \sigma \left[w_{21}^3 \begin{bmatrix} 0.21 \\ 0.22 \end{bmatrix} + b_{31} \right]$$

$$= \sigma \left[w_{21}^3 \cdot 0.21 + w_{11}^3 \cdot 0.22 + b_{31} \right]$$

$$= 0.31$$

Loss functions in DL



Loss Vs Cost function

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

Let say we have 5 students data.

$$L_1 = \text{student 1 loss} = (y_1 - \hat{y}_1)^2$$

$$L_2 = (y_2 - \hat{y}_2)^2$$

$$L_5 = (y_5 - \hat{y}_5)^2$$

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

$$= \frac{1}{5} (\sum L_i)$$

$$= \frac{L_1 + L_2 + L_3 + L_4 + L_5}{5}$$

1. ★ MSE $L = (y_i - \hat{y}_i)^2$ = L_2 error

ADV

- i) Easy to interpret
- ii) Easy to differentiate
- iii) only 1 local minima exists for $(y - \hat{y})^2$

DIS ADV

- i) Robust to outliers
- ii) only used when the output neuron activation function is "linear".

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

2. Mean Absolute (MAE) $\rightarrow$ L_1 loss

$$L = |y_i - \hat{y}_i| \quad \text{C.F} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

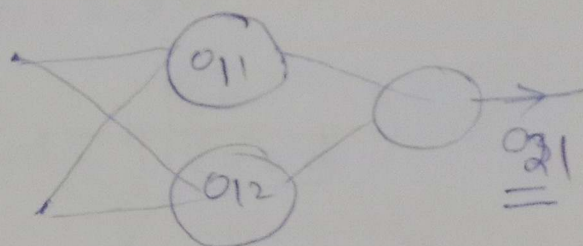
ADV

- 1) Intuitive and easy to understand
- 2) unit of y and Loss f^n are same.
- 3) Robust to outliers

DIS ADV

$|y - \hat{y}_i|$
Not differentiable
when $|y - \hat{y}_i| = 0$.

Binary cross entropy :-



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

$$\text{Cost } f^n = -\frac{1}{n} \left[\sum_{i=1}^n y_i \log \hat{y}_i - (1-y_i) \log (1-\hat{y}_i) \right]$$

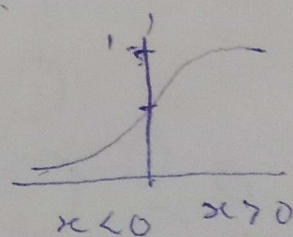
n = No. of points

y = actual values

$\hat{y}$ = predicted values

★ For binary classification in the output layer
Sigmoid activation funcⁿ is preferred.

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



Categorical cross entropy (multi class classification).

$$L = \sum_{i=1}^K (-y_i \log \hat{y}_i)$$

Backpropagation :- (summary)

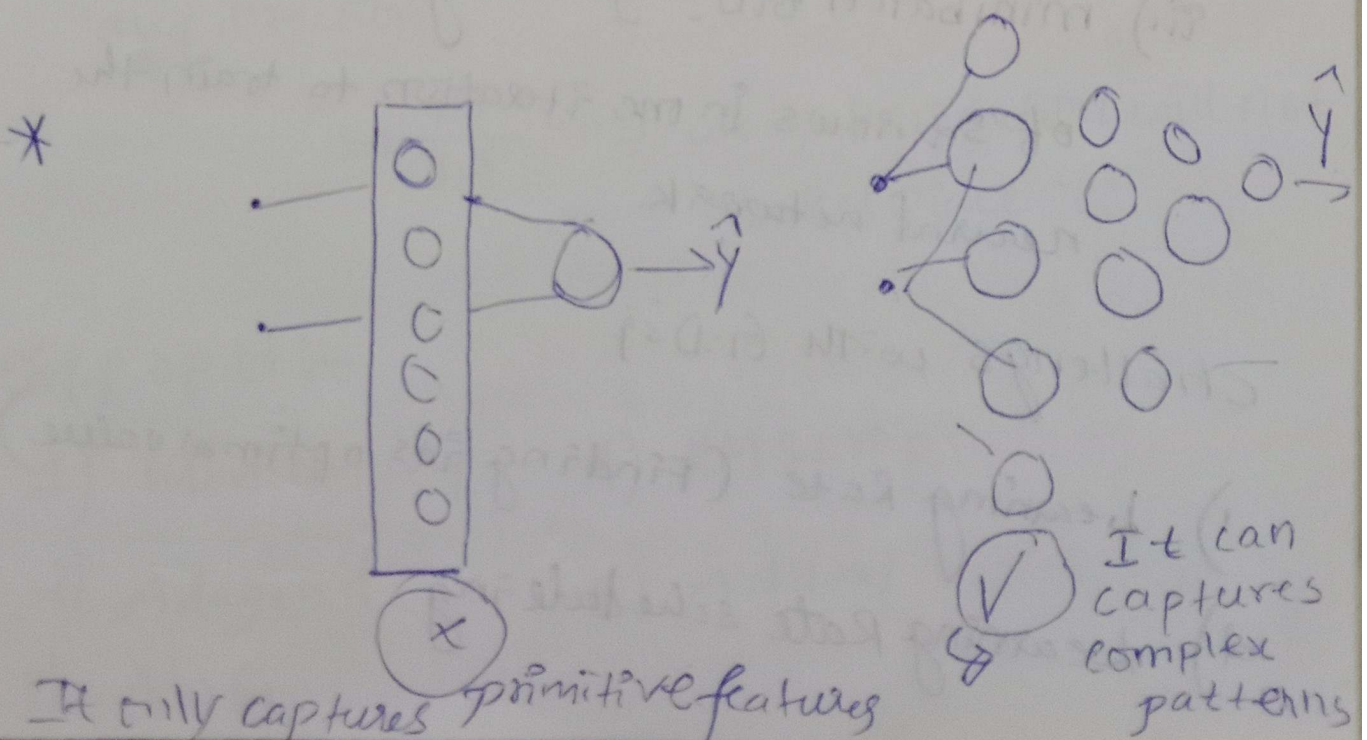
Steps:-

- 1) Initialize the weights and biases
- 2) For Forward propagation
- 3) Using the gradient descent updation
- of the weight for each row.

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

Defⁿ: Finding the gradient of loss function w.r.t to weights and biases; so that we get appropriate values of w & b w.r.t given data set
(os)

Training a neural architecture is called backpropagation.



* Learning up "Learning Rate" :-

→ When^u use complete dataset to train the neural network. we can use this technique that
∴ During start keep learning rate value small then increase.

OPTIMIZERS :-

The action of making the best or most effective use of a situation.

* We know :-

i) Batch gradient descent :- we use complete dataset for training.

ii) Stochastic GD :-

In one iteration we take 1 row to train.

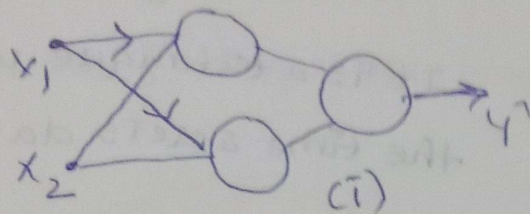
iii) minibatch GD :- generally we take size of 32 rows in one iteration to train the neural network.

Challenges with GD :-

- 1) Learning Rate (Finding its optimal value)
- 2) Learning Rate scheduling

3) we are restricted to use same learning rate to update different weights or biases.

4) $\hat{y}$ or the $L = (\gamma - \hat{y})^2$
or generally always



Loss function depends on $\hat{y}$.

* The the above architecture $\hat{y}$ depends on
- 6 weights and 3 biases. means if we
draw the graph b/w $L \times \hat{y}$. It will

be 10D.

* In 10D there will so many local minimums
so it will be difficult to obtain global
minium.

5) Saddle point (or) inflection point :-

In particular region the slope doesn't change in
any direction.

$\Rightarrow \frac{\partial L}{\partial w} = 0 \Rightarrow$ weight updation will not
takes place

Diff opti:-

1) momentum

4) RMSprop

2) Adagrad

5) Adam

3) NAG

prerequisite | prerequisite:- concept:-

* Exponentially weighted moving Average:-

It is a technique used to find hidden patterns in the time series data.

use cases:-

i) Time series

ii) signal processing

iii) Deep learning optimizers.

iv) Financial forecasting

It's formula formulated based on 2 condⁿ:-

i) The weightage of datapoint at time t >
of datapoint at $t-1$ time

ii) Any point overtime weightage decreases with time.

$$V_t = \beta V_{t-1} + (1-\beta) \theta_t$$

β = constant (0~1)

V_t = exponential weight avg at t

θ_t = data at time t

V_{t-1} = EWA at $t-1$, time.

* Generally $\beta = 0.9$ we take.

Momentum optimization:-

- The why?

* Generally the graph b/w Loss f' & parameters will be non-convex

- It solves 3 problems:

i) high curvature

ii) saddle point

iii) local minima's

The-What:-

$$w_{t+1} = w_t - v_t$$

decay factor

$$v_t = \beta v_{t-1} + \eta \frac{\partial L}{\partial w_t}$$

$$\beta = 0.9 \text{ (generally)}$$

previous velocities

Adv!: convergence will be faster compared to SGD or GD.

Dis Adv!: In the process of convergence there will be more oscillations. Other optimizers

are there which will converge faster (in non-convex datasets).

★ In concave graph of loss function

& parameters. Then momentum SGD is very useful. As we can operate values of β .

Nesterov Accelerated Gradient:- (NAG);

Steps

$$w_{t+1} = w_t - v_t$$

$$v_t = \beta v_{t-1} + \eta \nabla w_t$$

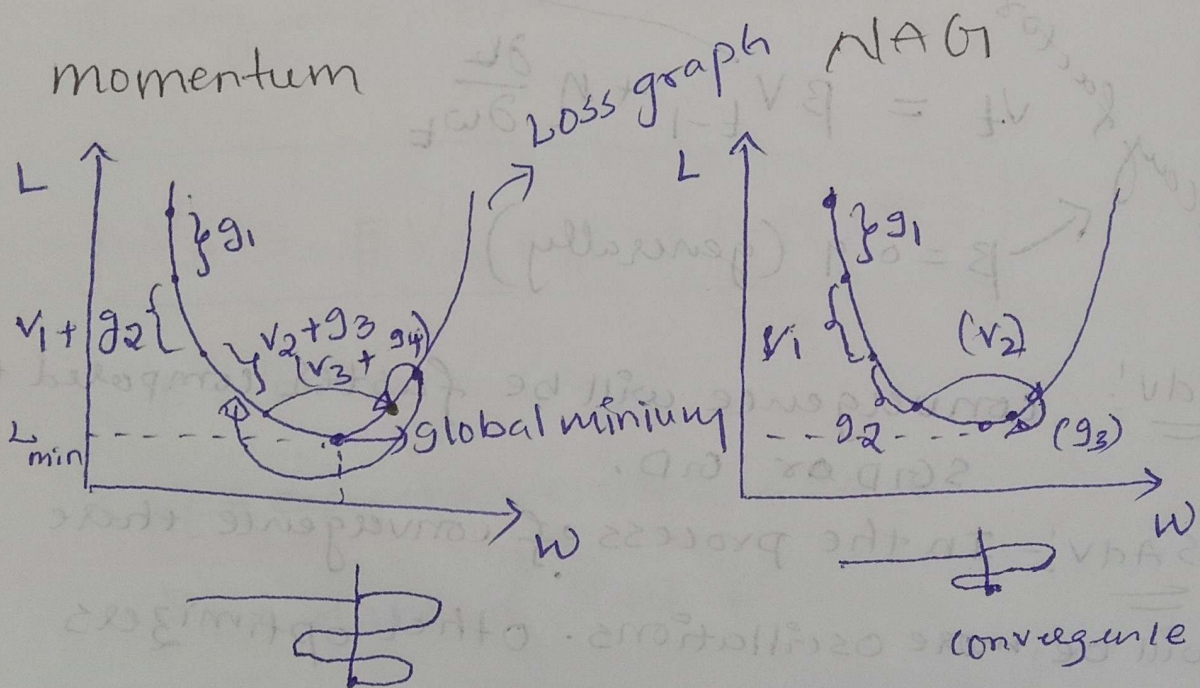
Step-1

* In NAG 1st we int update weight with only momentum as:-

$$w_{t+1} = w_t - \beta v_{t-1}$$

Step-2:- Now weights are updated because of gradient

$$w_{t+1} = w_t - \eta \nabla w_t$$

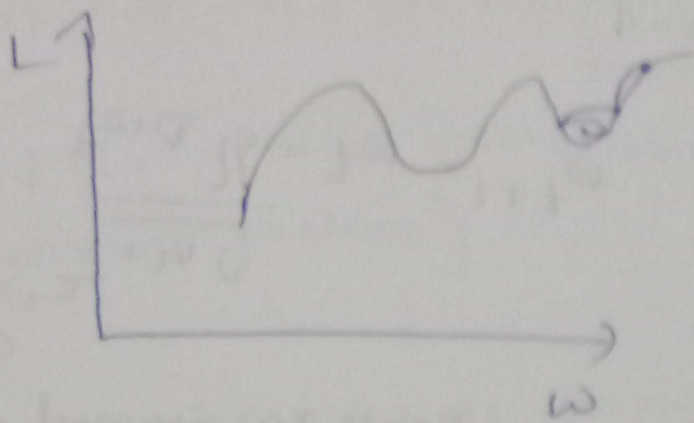


Here $v_3 + g_4$ occurs
simultaneously

Here (v_2) after

g_3 weight
update takes
place.

Dis Adv:-



* As we are decreasing the velocity as we not considering because of this oscillations are decreasing.

* As velocity is in some datasets there is the possibility that it may not overcome the local minimum.

Code:- `tf.keras.optimizers.SGD(`

`learning_rate=0.01, momentum=0.0, nesterov=False,`
`name='SGD')`

✓
SGD

✓
Momentum

✓
NAB

Above code is
valid for
SGD

momentum=0.9

momentum=
nesterov=True

nesterov=False

AdaGrad

$$\text{for: } w_{t+1} = w_t - \frac{\eta \nabla w_t}{\sqrt{v_t + \epsilon}}$$

ϵ small no. so that denominator $\neq 0$

v_t = past gradients ka squared sum

$$= v_{t-1} + (\nabla w_t)^2$$

Condⁿ for usage:

- i) when the features are not in the same scale
 - ii) features are sparse (means most of values in that column are 0)
- elongated ball problem

Imp: In Adagrad we use different learnings to update different weights or biases.

DIS ADV: we don't use this optimizer

in neural networks. Because as the no. of epochs are increasing v_t goes to big numbers

then $\frac{\eta}{\sqrt{v_t}} \approx \text{negligible}$, so weight updation will not take place and we can't get

* Adagrad is not used for non-convex optimization.

* It is used for convex optimization (like linear regression problems)

RMS Prop: Root mean square propagation

$$\Rightarrow \begin{cases} v_t = \beta v_{t-1} + (1-\beta) (\frac{1}{n} \nabla w_t)^2 \\ w_{t+1} = w_t - \sqrt{v_t} \end{cases} \quad \boxed{\beta = 0.95}$$

* we are more weightage for new gradients rather than previous.

Let $v_0 = 0$

$$v_1 = 0.95 \times 0 + 0.05 (\nabla w_1)^2$$

$$v_2 = 0.05 \times 0.95 \times (\nabla w_1)^2 + 0.05 (\nabla w_2)^2$$

$$v_3 = 0.95 \times 0.95 \times 0.05 (\nabla w_1)^2 + 0.95 \times 0.05 (\nabla w_2)^2$$

value becoming small. } Now it is solving prob of Ada grad

DIS ADV: No disadvantage, it is as powerful as Adam.

Adam:- It combines the core Ideas that we have learned before

i) momentum

ii) Adagrad (learning rate)

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \times m_t$$

where

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) (\nabla w_t)$$

momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla w_t)^2$$

Adagrad

Initial: $v_0 = 0$ & $m_0 = 0$ so there will be

some bias

Bias correction:-

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^{(t)}} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^{(t)}}$$

epoch no. = t

★ most popular algorithm for non-convex optimization.