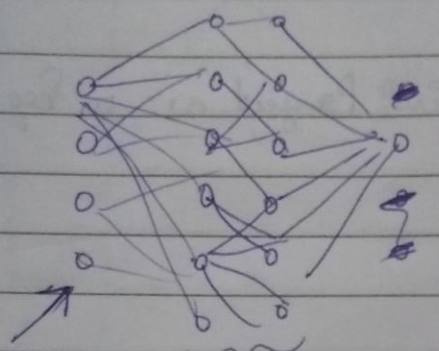
CNN
- convolution → scanning
- Pooling

PAGE No.
DATE

Neural Network



Input layer - accepts input in different forms.

Hidden layer → Performs calculation on these inputs

Output layer → then delivers the outcome of the calculations & extractions

- Each of these layers ~~are connected to~~ contains Neurons that are connected with neurons of previous layers.
- Each Neuron has its own weight.

4n → hidden layer → activat^ functions

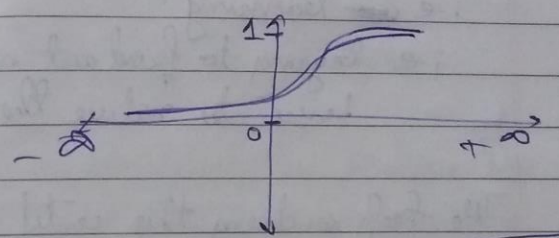
PAGE NO.
DATE

Neural Network

- artificial Neuron → "Perceptron"
- Hidden layer is a layer responsible for learning.
- input layer → receiver layer

Activat^ functions

- more than 50-60 AF.
- Basic → sigmoid (0-1)
- ~~Relu~~ →



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

- Purpose? → data Normalization.
  Normalizat^, standardizat^
  ↙
  changing the scale.
- activat funct basically helps to reduce the data into a smaller range

Normaliz the data → change the scale.
By passing the data to a activat^ funct^, doesn't matter which type of activat^ funct^ we use.

- weights → defines a relat^ bet^ input ~~from~~ taken & input that goes inside the next layer.

~~from~~ we start for new with random weights.

Tensorflow Playground

PAGE No.
DATE

- loss → difference bet' y & $\hat{y}$ $(y-\hat{y})$ → error.
- There are different types of loss functions

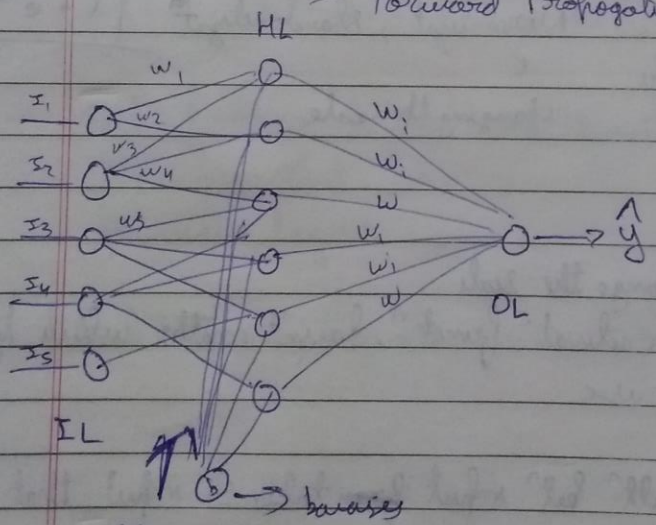
- We pass the loss/cost to the optimizer

- In forward propagation we don't learn anything.

- optimizer tries to perform backward propagation
  i-e learning.
  i-e trying to find out new values of wts & biases to reduce the loss.

We keep on doing this until we find minimum loss

→ forward Propagation



all pnc

← backward Propagation

w_i → weights

$z = \sum_{i=1}^{n} (I_i w_i + b)$ this goes into hidden layer

In hidden layer we have activation layer which fills

learning → finding pattern

in Normalizing the data.
eg: sigmoid function $\frac{1}{1+e^{-x}}$ will result in values bet^ 0 & 1

Initially random weights are assigned

From hidden layers it goes to output layer where we use probabilistic Funct^ like sigmoid to give the Probably $\therefore \hat{y}$
loss = $(y - \hat{y})$ we send to loss to optimizers

We have optimizers which are used to reduce the loss function by back propagation and changing of weights

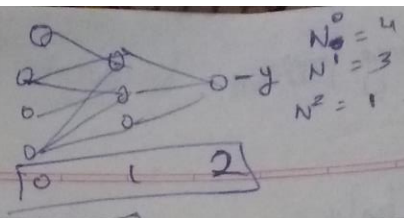
This process is done until we get the lowest loss function.

Bias can be introduced anywhere (in hidden layer or OL or IL).
It basically helps in better learning.

$$w_n = w_{n-1} - \eta \Delta E_w$$

$$w_n = w_{n-1} - \eta \left(\frac{dE}{dw}\right)$$

→ new wt, → older wt, → eta → learning rate → const



PAGE No.
DATE

$$w_n = w_{n-1} - \eta\left(\frac{dE}{dw}\right)$$

$$\frac{dc}{dw} = \frac{\partial c}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Chain Rule —

$$\frac{\partial c}{\partial w} = \sum_{i=1}^{N} \frac{\partial c}{\partial z_i} \cdot \frac{\partial z_i}{\partial w}$$

$c \rightarrow$ cost fun^n / loss

$w \rightarrow$ wts

$z = \sum_{i=1}^{N} (x_i w_i + b)$

↑ the input which going inside hidden layer / output fun^n

- $L \rightarrow$ No of layers in Network
- N^n – Dimensionality of the layer $n \in \{0, \ldots L\}$
  - $N^0 \rightarrow$ dimensionality of the input
  - $N^L \rightarrow$ dimensionality of the output
- $W^m \in R^{N \times N^{m-1}}$ – weight matrix for layer $m \in \{1, \ldots L\}$ (hidden layer)
  W_{ij}^m is the wt bet^n the i^{th} unit in layer m & j^{th} unit in layer $m-1$.
- $b^m \in R^{N^m}$ – Bias vector for layer m.
- σ^m – Non linear activation fun^n of the units in layer m, applied elementwise
- $z^m \in R^{N^m}$ – linear min of the inputs to layer m. computed by $z^m = W^m a^{m-1} + b^n$

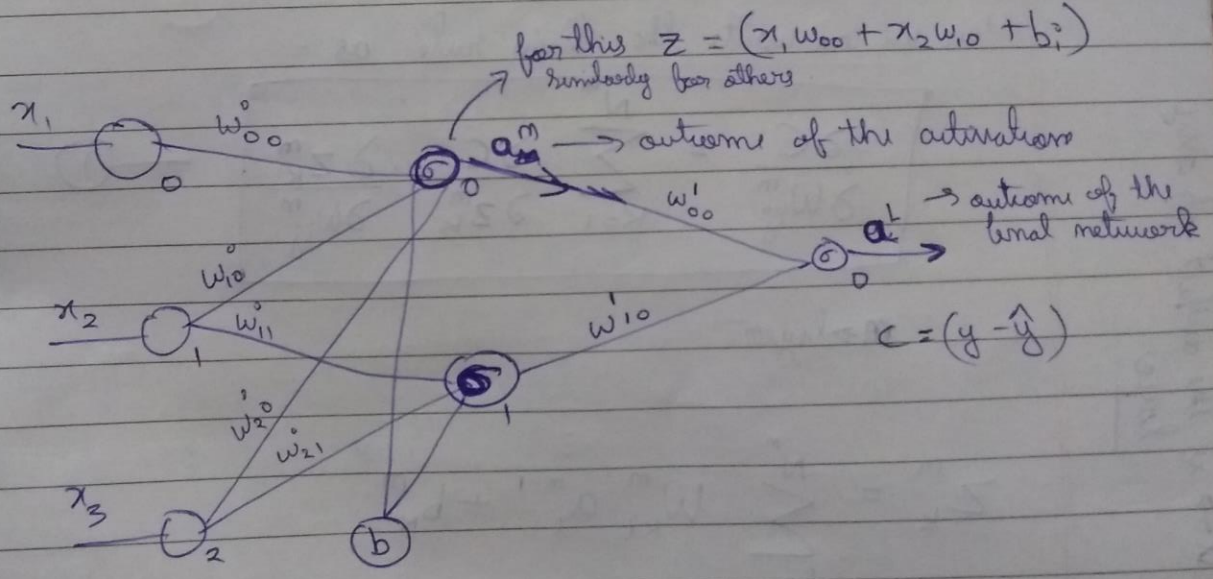
$\sigma^m(h^m)$

↳ Passing some value in the activat° funct°

- $a^m \in R^{N^m}$ → activation of units in layer m - computed by $a^m = \sigma^m(h^m) = \sigma^m(w^m a^{m-1} + b^m)$. a^L is the o/p of the Network. We define a special case a^0 as the input of the network

- $y \in R^{N^L}$ – Target output of the network

- C → cost / error funct° of the network, which is a funct° of a^L (the network output) & y (treated as const)



L = layer 0 1 2.

N^n $N^0 = 3$ $N^1 = 2$ $N^2 = 1$

How do we change wts?

$$w_m = w_{m-1} - \eta \left(\frac{dE}{dw}\right) \rightarrow \left(\frac{dC}{dw}\right)$$

w_m → new wt, w_{m-1} → old wt, η → const, $\frac{dC}{dw}$ → our aim

PAGE No.
DATE

$\frac{\partial C}{\partial a^L}$ → rate of change of cost function depends on the output of the Network (a^L)

$\frac{\partial a^m}{\partial z^m}$ → rate of change of o/p of the activation funct^n (a^m) depends on the input (z^m) which it takes in that particular layer.

★ Our requirement → To find out rate of change of Error w.r.t weights
or CF/LF

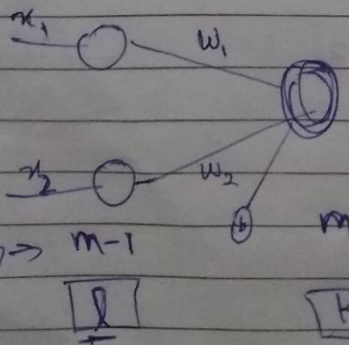
Now we can write the chain rule as:-

$$\frac{\partial C}{\partial w_{ij}^m} = \sum_{K=1}^{N^m} \frac{\partial C}{\partial z_K^m} \cdot \frac{\partial z_K^m}{\partial w_{ij}^m} \quad \text{— (1)}$$

m → layer

a → is the output of any layer

$$z_K^m = \sum_{l=1}^{N^m} w_{Kl}^m a_l^{m-1} + b_K^m$$



→ input of this layer
$= (w_1 x_1 + w_2 x_2 + b)$

w_{Kl}^m → wt betⁿ K & l of m layer
i.e w_1

a_l^{m-1} ⇒ output of l of m-1 layer eg x_1

129

$$\frac{\partial C}{\partial w_{ij}^m} = \sum_{K=1}^{N^m} \frac{\partial C}{\partial z_K^m} \left(\frac{\partial z_K^m}{\partial w_{ij}^m} \right)$$

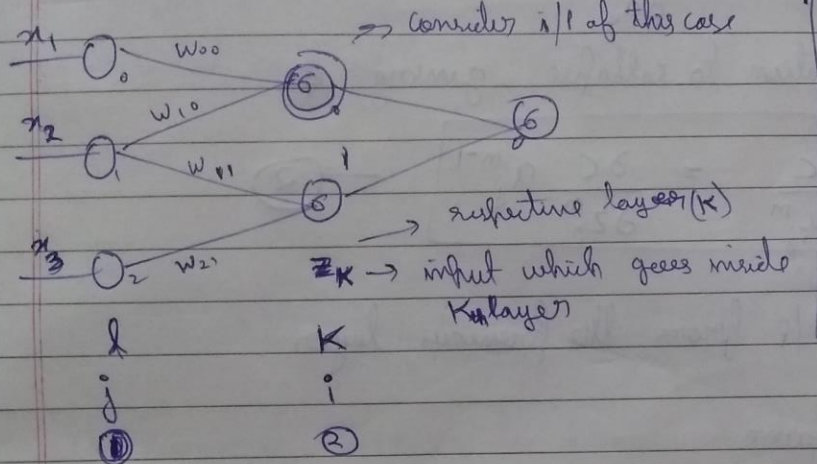
finding

Refering z_K^m in eq ② Finding derivative of z_K^m w.r.t W

$$\frac{\partial z_i^m}{\partial w_{ij}^m} = \frac{\partial}{\partial w_{ij}^m} \left(\sum_{l=1}^{N^m} w_{il}^m a_l^{m-1} + b_i^m \right)$$

→ constant derivative of constant = 0

$$= a_j^{m-1}$$

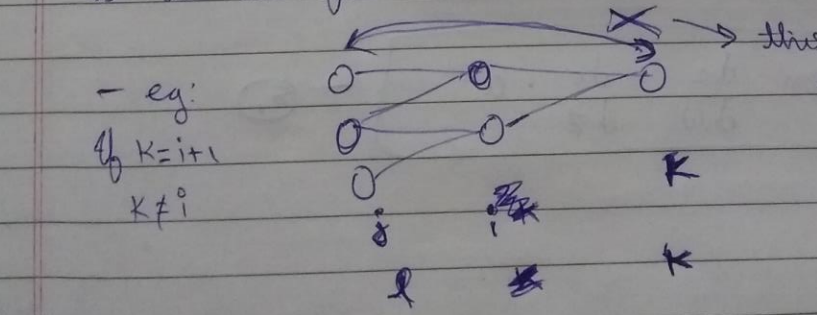


$y = x_1 w_{00}$

$\frac{dy}{dw} = x_1$

$\frac{dy}{dx} = w_{00}$

If $K \neq i$ then there is no relat" but layer ① & ② so we cnt find the derivative ∴ 0.



- eg: If $K = i+1$, $K \neq i$

If $K = i$, the input of layer i will be $x_1 w_{00}$ for a specific neuron and if we derivate it w.r.t W we get x_1 i.e input of previous layer

i.e a_j^{m-1}

$$\frac{\partial z_k^m}{\partial w_{ij}^m} = \begin{cases} 0 & k \neq i \\ a_j^{m-1} & k = i \text{ (same layer)} \end{cases}$$

Now, the fact that $\frac{\partial C}{\partial a_k^m}$ is 0 unless $k = i$ causes the summation to collapse, giving

$$\frac{\partial C}{\partial w_{ij}^m} = \frac{\partial C}{\partial z_i^m} a_j^{m-1} \quad \text{— (2)}$$

$a_j^{m-1} \rightarrow$ I/P from the previous layer.

in vector form

$$\frac{\partial C}{\partial w^m} = \frac{\partial C}{\partial z^m} a^{m-1}$$

in simple form $\frac{dc}{dw} = \frac{dc}{dz} \cdot a^{m-1}$ — (3)

$$b_n = b_{n-1} - \eta \left(\frac{dc}{db}\right)$$

PAGE No.
DATE

* Similarly for the bias variables b^m, we have.

$$\frac{\partial C}{\partial b_i^m} = \sum_{k=1}^{N^m} \frac{\partial C}{\partial z_k^m} \cdot \frac{\partial z_k^m}{\partial b_i^m}$$

finding $\frac{\partial z_k^m}{\partial b_i^m}$

$$\frac{\partial z_i^m}{\partial b_i^m} = \frac{\partial}{\partial b_i^m} \left(\sum_{l=1}^{N^m} w_{il}^m a_l^{m-1} + b_i^m \right)$$

$$= 1$$

$$\frac{\partial z_i^m}{\partial b_i^m} = \begin{cases} 0 & k \neq i \\ 1 & k = i \text{ (same layer)} \end{cases}$$

$z = xw + b$

$\frac{dz}{db} = 0 + 1$

$\boxed{\frac{dz}{db} = 1}$

The summation collapses to give

$$\frac{\partial C}{\partial b_i^m} = \frac{\partial C}{\partial z_i^m} \quad \text{— (3)}$$

in vector form

$$\frac{\partial C}{\partial b^m} = \frac{\partial C}{\partial z^m}$$

in simple form

$$\frac{dc}{db} = \frac{dc}{dz} \cdot 1 \quad \text{— (3.2)}$$

If we are able to find $\frac{\partial C}{\partial z}$ which is same in eqn (2) & eqn (3) we can find out $\frac{dc}{dw}$ & $\frac{dc}{db}$

or values of b_n & w_n

$b_n = b_{n-1} - \eta (dc/db)$

$w_n = w_{n-1} - \eta (dc/dw)$.

PAGE No.
DATE

* Now, we must compute $\frac{\partial C}{\partial z_k^m}$. For the final layer ($m = L$), this term is straightforward to compute using the chain Rule

$$\boxed{\frac{\partial C}{\partial z_k^L} = \frac{\partial C}{\partial a_k^L} \cdot \frac{\partial a_k^L}{\partial z_k^L}} \quad - (F_1)$$

* $\frac{\partial C}{\partial a_k^L}$ rate of change of C.F w.r.t final outcome of the network a^L

eg:- if $\hat{y}$ is close to y then C.F / loss F^ would be less

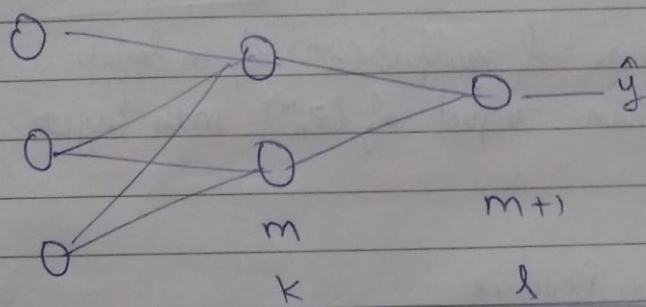
in vector form

$$\boxed{\frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}}$$

for mth layer we can write the eq^

$$\frac{\partial C}{\partial z_k^m} = \frac{\partial C}{\partial a_k^m} \cdot \frac{\partial a_k^m}{\partial z_k^m}$$

$$= \left(\sum_{l=1}^{N^{m+1}} \frac{\partial C}{\partial z_l^{m+1}} \frac{\partial z_l^{m+1}}{\partial a_k^m} \right) \cdot \frac{\partial a_k^m}{\partial z_k^m}$$: applying chain rule



$\frac{\partial C}{\partial z^{m+1}}$ → rate of change of C.F depends upon input (z) in (m+1)th layer.

$\frac{\partial z^{m+1}}{\partial a_k^m}$ → rate of change of z^{m+1} (input in m+1) layer depends upon output (a_k^m) of mth layer.

$$= \left(\sum_{l=1}^{N^{m+1}} \frac{\partial C}{\partial z_l^{m+1}} \frac{\partial}{\partial a_k^m} \left(\sum_{h=1}^{N^m} w_{lh}^{m+1} a_h^m + b_l^{m+1} \right) \right) \frac{\partial a_k^m}{\partial z_k^m}$$ (the b_l^{m+1} term → 0)

$$= \left(\sum_{l=1}^{N^{m+1}} \frac{\partial C}{\partial z_l^{m+1}} w_{lk}^{m+1} \right) \frac{\partial a_k^m}{\partial z_k^m}$$

$$\boxed{\frac{\partial C}{\partial z_k^m} = \left(\sum_{l=1}^{N^{m+1}} w_{kl}^{m+1T} \cdot \frac{\partial C}{\partial z_l^{m+1}} \right) \frac{\partial a_k^m}{\partial z_k^m}} \quad - (4)$$

simple form $\boxed{\frac{dC}{dz} = \left(w \cdot \frac{dC}{dz} \right) \left(\frac{da}{dz} \right)} \quad - (5)_3$

1:46

S_1 - $\frac{dc}{dw^m} = \left(\frac{dc}{dz^m}\right)\left(a^{m-1}\right)$

S_2 - $\frac{dc}{db^m} = \frac{dc}{dz^m} \cdot 1$

S_3 - $\frac{dc}{dz^m} = \left(w^{m+1}\frac{dc}{dz^{m+1}}\right)\left(\frac{da^m}{dz^m}\right)$ - final Eq^n

$\frac{da^m}{z_k^m}$ → rate of change of output (a^m) of m layer w.r.t z_k^m depends upon input of (z_k^m) mth layer.

Backpropogation in Pratice

Nonlinearity	$a^m = \sigma^m(z^m)$	$\partial a^m / \partial z^m$	Notes
sigmoid	$1/(1+e^{-z^m})$	$\sigma^m(z^m)(1-\sigma^m z^m) = a^m(1-a^m$	- squashes any input to the range [0,1]
Tanh	$\frac{e^{z^m}-e^{-z^m}}{e^{z^m}+e^{-z^m}}$	$1-(\sigma^m(z^m))^2 = 1-(a^m)^2$	Equivalent, up to scaling, to the sigmoid function
ReLU	$\max(0, z^m)$	$0, z^m<0$; $1, z^m \geq 0$	Commonly used in NN with many layers.

Now, if we use sigmoid funt^n as our activation layer, the Eq^n will become

$$\frac{dc}{dz^m} = \left(w^{m+1}\frac{dc}{dz^{m+1}}\right) \cdot \left(a^m(1-a^m)\right)$$

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

PAGE No.
DATE

with different sigmoid funct^ our $\frac{\partial a}{\partial z}$ value changes.

Imp
Q. Do we use activat^ funct^ in Backward Propagat^?
-> yes, we do

Cost funct^	C.F	$\frac{\partial C}{\partial a^L}$	Notes
Squared Error	$\frac{1}{2}(y-a^L)^T(y-a^L)$	$y-a^L$	commonly used when the o/p is not constrained to a specific range.
Cross-Entropy	$(y-1)\log(1-a^L)$	$\frac{a^L-y}{a^L(1-a^L)}$	commonly used for binary classificat^ tasks; can yield faster convergence.

* In Pratice, backpropagation proceeds in the following manner for each training sample:

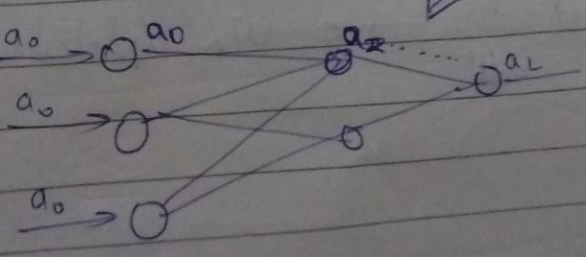
1. Forward Pass : Given the network input a^0, compute a^m, recursively by

$a^1 = \sigma^1(w^1a^0+b^1)$ ... $a^L = \sigma^L(w^La^{L-1}+b^L)$

(bias)

activat^ funct^

many layers in between



2. Backward Pass: Compute → cost func^n

Imp Eq^n — $\frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$ → activation func^n

for the final layer based on the tables in previous page; then recursively compute:

$$\frac{\partial C}{\partial z^m} = \left(w^{m+1T} \frac{\partial C}{\partial z^{m+1}} \right) \cdot \frac{\partial a^m}{\partial z^m} \quad - S_3.$$

for all other layers. Plug these values into

$$\frac{\partial C}{\partial w^m} = \frac{\partial C}{\partial z_i^m} \cdot a^{m-1T} \quad — S_1$$

and

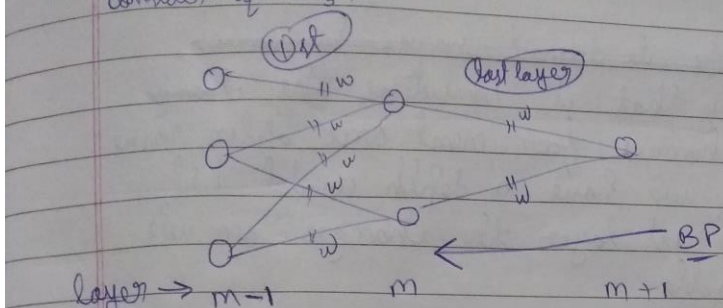
$$\frac{\partial C}{\partial b^m} = \frac{\partial C}{\partial z^m} \quad — S_3$$

to obtain the updates.

By subst Imp Eq^m in S_3 we get $\frac{\partial C}{\partial b^m}$ i.e b_n

and in S_1 we get $\frac{\partial C}{\partial w}$ i.e w_n (updated wt)

Consider Eq^n S_3 :-

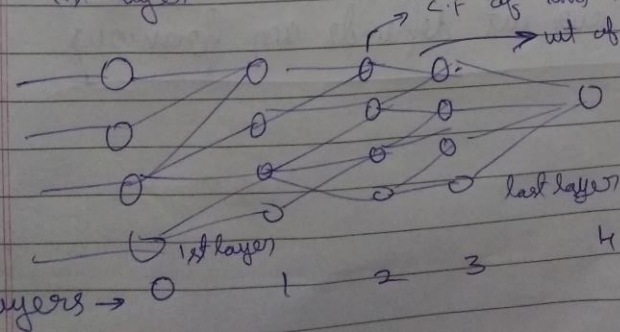


IMP — $\frac{\partial C}{\partial z^m}$ → rate of change of C.F w.r.t input of m^{th} layer (z^m) ~~defn~~ of m^{th} layer.

~~to~~ depends upon weight of previous layer (w^{m+1})

In Backward Propagation (BP)

Therefore we say that if we have many layers in between & as C.F is dependent on wts of previous layer so if the model is deep, there are very high chances that there will be no change in wts of 1st layer.



2:05

Similarly C.F depends on wts of previous layer in B.P.

Therefore if the wts doesn't change by much, Therefore we say that it considers the change previous wt changes after next one after some time ~~#~~ i.e if we have a depth model till we reach the 1st layer the changes in wts will be negligible.

Therefore we avoid making model with ~~deep~~ many layers.

∴ There will be no learning if wts doesn't change

eg $y = x^2$ $\quad \frac{dy}{dx} = 2x$, ~~6~~

$x=3$

$\frac{d^2y}{dx^2} = 2$ ~~2~~

$\frac{d^3y}{dx^3} = 0$ ~~0~~

similar things happen in case of wts if we keep taking derivative of it.

Only in B.P our wt depends on previous layer

Example :-

Sigmoid Network with cross-entropy loss using gradient descent

→ actual "final"

step 1 $\frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$

↳ L.F.

y → actual value

$= \left(\frac{a^L - y}{a^L(1-a^L)} \right) a^L (1-a^L)$ from tables

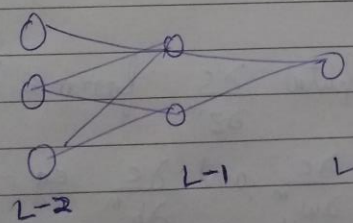
$\boxed{\frac{\partial C}{\partial z^L} = a^L - y}$ → as we have got the value of $\frac{\partial C}{\partial z}$ we can find out (S3)

$\frac{\partial C}{\partial b^L} = (a^L - y)$

$\therefore \boxed{b_n = b_{n-1} = \eta (a_L - y)}$

Bias final value

step 2. From here, we can compute



$\frac{\partial C}{\partial z^{L-1}} = \left(w^{LT} \frac{\partial C}{\partial z^L} \right) \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}}$

finding (S3) for $m = L-1$

$= w^{LT} (a^L - y) \cdot a^{L-1} (1 - a^{L-1})$

$\frac{\partial C}{\partial z^{L-2}} = \left(w^{L-1} \frac{\partial C}{\partial z^{L-1}} \right) \cdot \frac{\partial a^{L-2}}{\partial z^{L-2}}$

$= w^{L-1} \left(w^{LT} (a^L - y) \cdot a^{L-1} (1 - a^{L-1}) \right) \cdot a^{L-2} (1 - a^{L-2})$.

2:20

We can also find now (ε_3)

$$w_n = w_{n-1} - \eta(a^L - y)$$ Updating the weight.

~~for L-1 layer~~ Similarly we can find for all layers.

w_{L-1} = as shown.

Important Note:-

If our cost Function = 0 ∴ in this case. $y - a^L = 0$

then [Imp Eq^n = 0] ∴ $\frac{\partial C}{\partial z^m} = 0$

& $\frac{\partial C}{\partial b^m} = 0$ therefore there will be no updations of bias & weights

↖ and so on, until we have computed $\frac{\partial C}{\partial z^m}$ for $m \in \{1 \dots L\}$

This allows us to compute $\frac{\partial C}{\partial w_{ij}^m}$ and $\frac{\partial C}{\partial b_i^m}$ eg.

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial z^L} a^{L-1}$$

$$= (a^L - y)\, a^{L-1}$$

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial z^{L-1}} a^{L-2}$$

$$= w^{LT}(a^L - y) \cdot a^{L-1}(1 - a^{L-1})\, a^{L-2}$$

So standard gradient descent then updates each parameter as follows:

$$w^m = w^n - \lambda \frac{\partial C}{\partial w^n}$$

$$b^m = b^m - \lambda \frac{\partial C}{\partial b^m}$$

this was our aim i.e how weights & biases get updated

where λ is the learning rate. This process is repeated until some stopping criteria is met.