Neural Network

1) Neural Network represent & learn highly complex & nonlinear function.

* Neural Network Inspired by Human Brain

Human Brain

1) Human brain contains a number of neurons, of the order of tens of billions, which are highly interconnected.

2) This neurons individually are simple computing units, but together they can perform very complex tasks.

3) Some characteristics of neural human brain incorporated to form Neural Network Following are characteristics.

1) Massive Parallelism
2) Neurons are highly interconnect to each other & solve complex task together
3) Model distributive associtative memory Through weights in synaptic connection

Human - Neuron

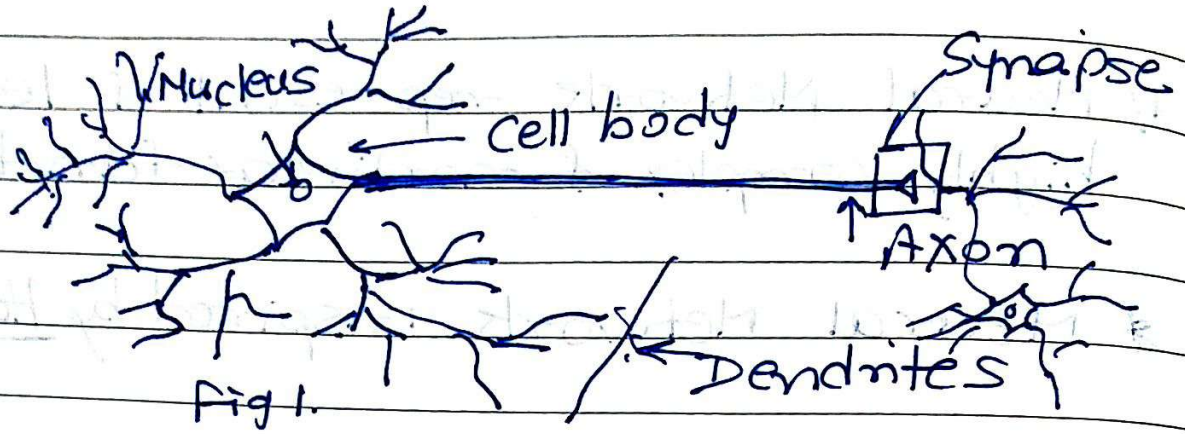


Fig 1.

Neural Unit

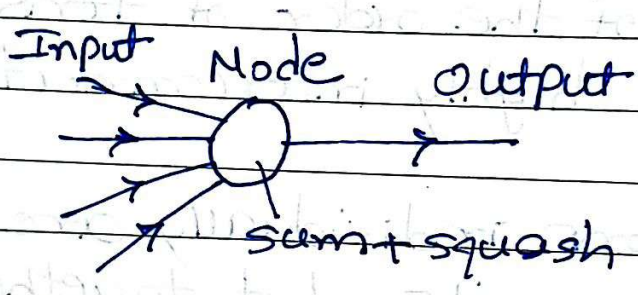


Fig. 2

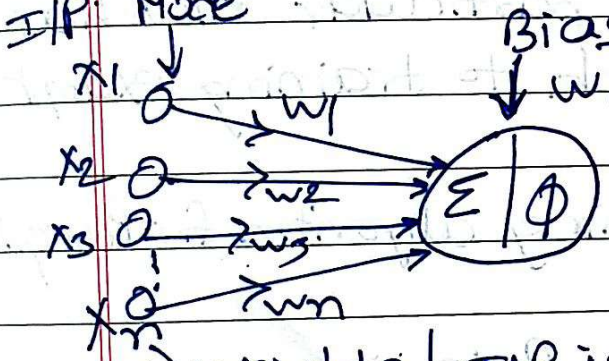
1) As shown in fig 1., there is Nucleus & dendrites through which i/p is expected
2) Through axon is the tail through which the output is given.
3) The fig 1 structure is of the neuron is simulated by a neural network unit as shown in fig 2. which has inputs & then some simple computation are at node & o/p unit
4) for computation use weighted sum of the inputs & then applies a function as squashing function (for ex sigmoid fun^n)

5) Through the dendrites the i/p is accepted & through the axon the output is transmitted through the electric impulses through the synaptic layer to the other neurons

6) So Neural Nlw inspried by neuron human brain.

Single Neural Network

Perceptron Nlw.



1) Weighted I/P is computed along with bias & transfer through transfer function

$$\phi(z) = z$$

1) for example thresholding transfer function.

$$\phi(z) = \sqrt{\quad}$$

2) If weighted i/p & bias summation is grater than threshold O/P is high(1) otherise low(0)

2) Some nonlinear function

1) Sigmoid function 2) Tanh function

$$\therefore y_{in} = \sum_{i=1}^{n} x_i w_i + b$$

$$\therefore y_{out} = f(y_{in})$$

Consider training example

x_1	y_1
x_2	y_2
x_3	y_3
x_{in}	y_n

Training the network means learning these weights $w_0, w_1, w_2 \ldots w_n$ for given training example, so that this particular nlw has a good fetch to training example

So we have a training algorithm for Perceptron

* Perceptron training Rule,

1) Initialise weights $w_0, w_1, w_2 \ldots w_n$ with intial values of the weights & at each iteration we will update the weights.

2) for x_1 select w_1 for that o/p y_1, check if o/p y is wrong need to update weight. So that o/p is closer to given o/p.

$$\therefore \text{New } w_i = \text{Old } w_i + \Delta w_i$$

$$\Delta w_i = \eta (y - \hat{y}) x_i$$

η is learning rate, it control how much you change based on error. the weight updation.

By taking the example one by one. It can be shown that this perceptron learning converges if D is linearly seprable.

That means, there is a linear decision surface that separates the positive & negative examples. if such a linear seperation surface exists then by applying this perceptron training rule, the learning algorithm will converge to hypothesis which will separate the positive & negative example.

→ But if data is not linearly seperable then this perceptron learning algorithm will not converge, it will not work.

→ Here use Gradient Descent Algorithm for weight updation.

$$E = \frac{1}{2} \sum (y_d - \hat{y}_d)^2$$

$\phi(z) = z$ Stochastic Gradient

$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ gradient descent give direction for local minim

Date
Page

So to handle nonlinear data we need to go for multilayer network.
multilayer network connected with each other
In order to represent complex functions we want non-linear unit, which is differentiable, so we go for a trasfer function which is differentiable & non-linear.
often use transfer function is

Sigmoid Function (Logistic function)

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

Differential of sigmoid function is

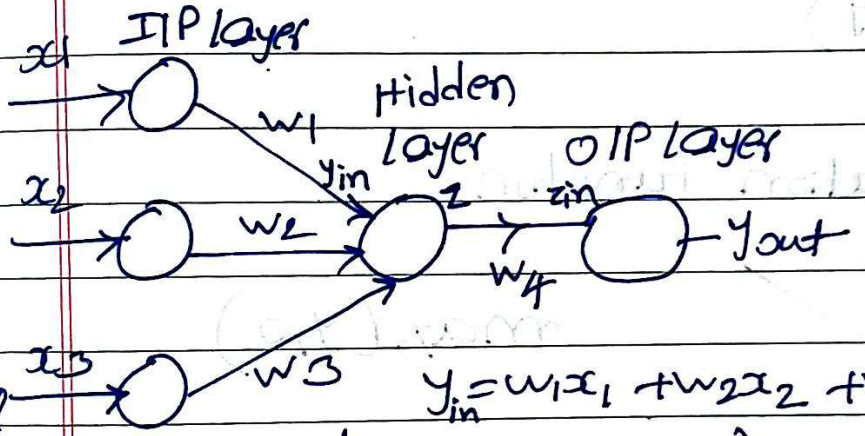
$$\phi'(z) = \phi(z)\,(1-\phi(z))$$

Error is $E = \frac{1}{2}\sum(y_i - \sigma(w \cdot x_i))$

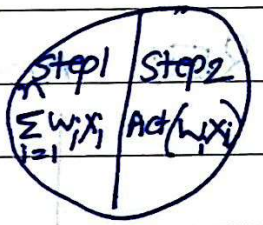
$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\sum_d \frac{\partial E}{\partial y_d} \cdot \frac{\partial \hat{y}_d}{\partial w_i}$$

Basic Airchiture of Neural Network,

How Neural Network works



Hidden Neuron



$y_{in} = w_1x_1 + w_2x_2 + w_3x_3 + \text{Bias}(b)$

forward Propogation $z = Act(y)$

Weight & Activation fun^n playing important role.

For ex.- Activication function Sigmoid $= \frac{1}{1+e^{-y}}$

It transform OIP in bet^n 0 to 1

→ when OIP of activation fun^n is less than

→ 0.5 means OIP is zero means neuron is not activited

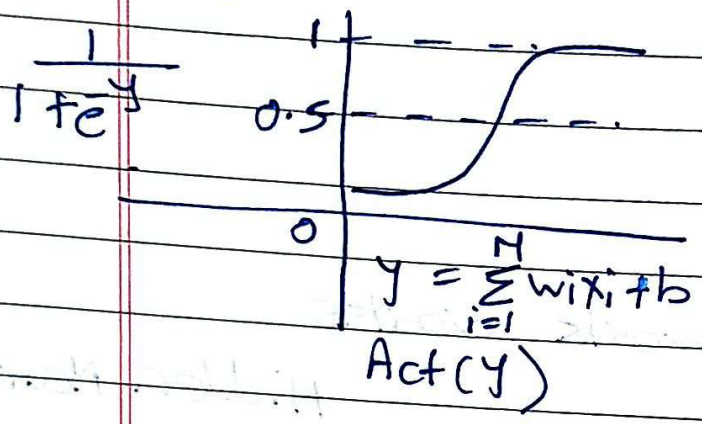
If it is grater than 0.5 the neuron gets activited.

This is forward $z_{in} = z \times w_4$

Activation function $y_{out} = f(z_{in})$

* Activation functions

1) Sigmoid Activation Function.



2) REL Activation Function.

When y is -ve o/p is zero
When y is +ve o/p is w.r.t that

Jeoffrey Hinton

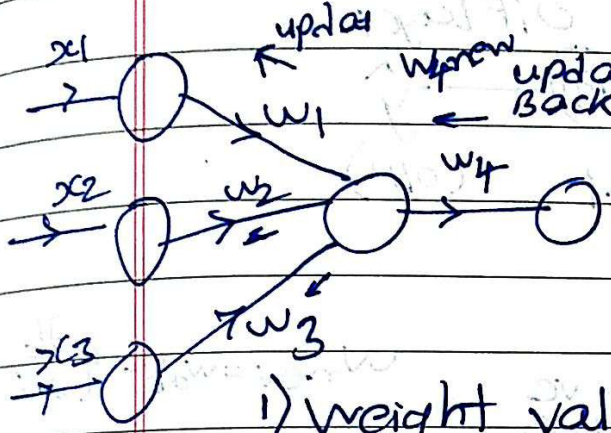
↓

* Train Neural Network with Back Propogation

$$\hat{y} = [x_1 w_1 + x_2 w_2 + x_3 w_3] + b_1$$

$$z = Act(y)$$

x_1	x_2	x_3	y
Play	Study	Sleep	O/P
2H	4H	8H	Pass/Fail



Optimization is used to minimize Loss

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

1) weight value should be adjusted to reduce loss value.
2) To reduce the loss value we need to backpropogate & update the weight value.
3) Update w_4 by backpropogate

$$w_{4_{new}} = w_{4_{old}} - \eta \frac{\partial L}{\partial w_4}$$

find derivative of loss w.r.t. w_4

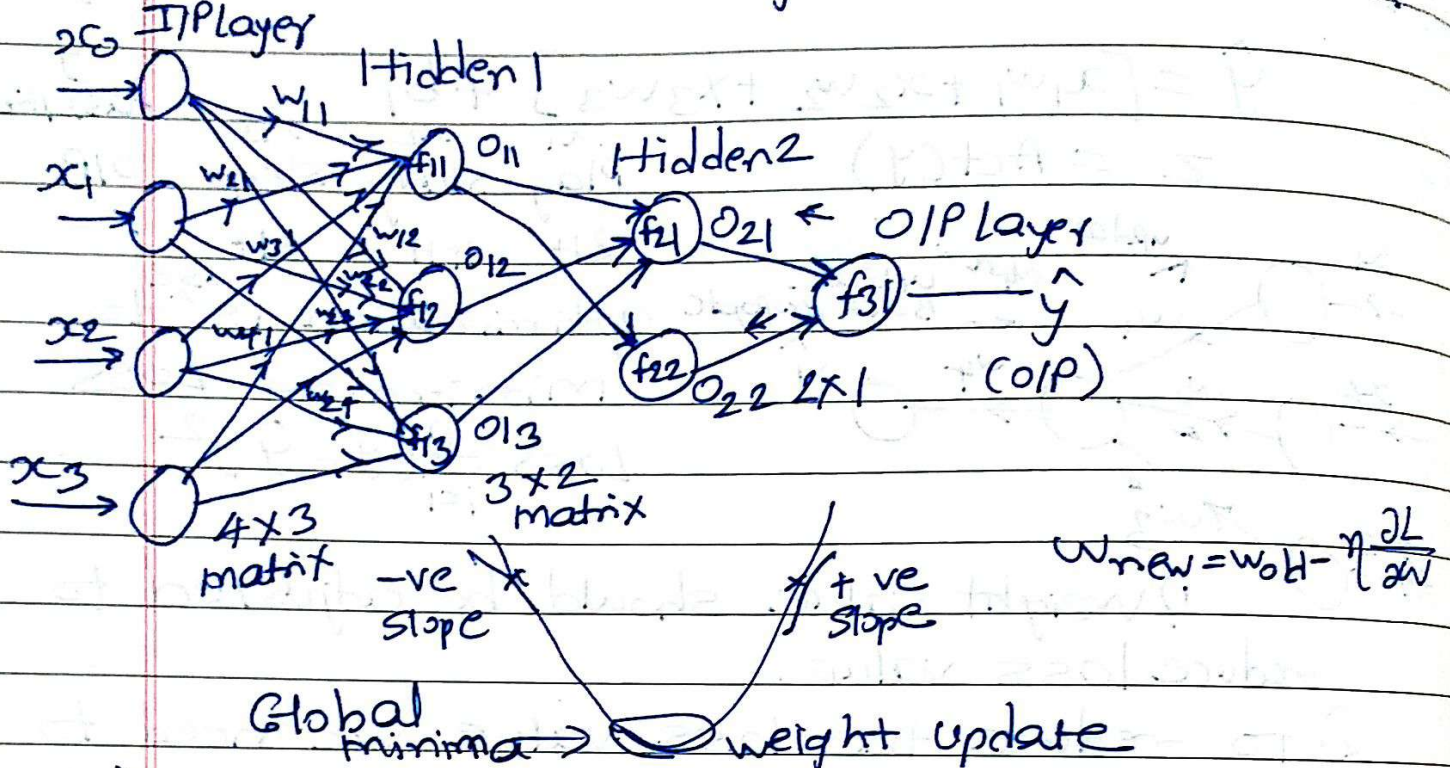
η = Learning Rate - Small value it will help us to achive global minima.

4) Same way update w_1, w_2, w_3 also

$$w_{1new} = w_{1old} - \eta \frac{\partial L}{\partial w_1}$$

5) Once we get updated value forward propogation happen to find pridicate $\hat{y}$.
6) This will repeat the process ephocs literation untill loss gets reducesed.

How to train multilayer Neural Network



1) I/P matrix of size 4×3
2) $f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{31}$ Neuron function which perform 2 steps
   1) I/P multiply with weight & Summation
   2) Apply this Summation to activation function
3) After all layer execution we get predicated O/P $\hat{y}$ by which we find out Loss function
4) main aim is to reduce loss function to achive actual output.
5) To reduce loss function we have to use optimizer
6) Here we use Gradient Descent
   Stoactistic Gradient Descent

why derivative to adjust the slop.

to update the weight

* Gradient Descent Optimizer
use to update (optimize) weight value to reduce loss function
This will done by backpropogation.

* Using backpropogation we need to update weight.

* If loss function is not reduced means there is problem with learning rate & derivative value & it will not converges properly.
So need to adjust properly learning rate

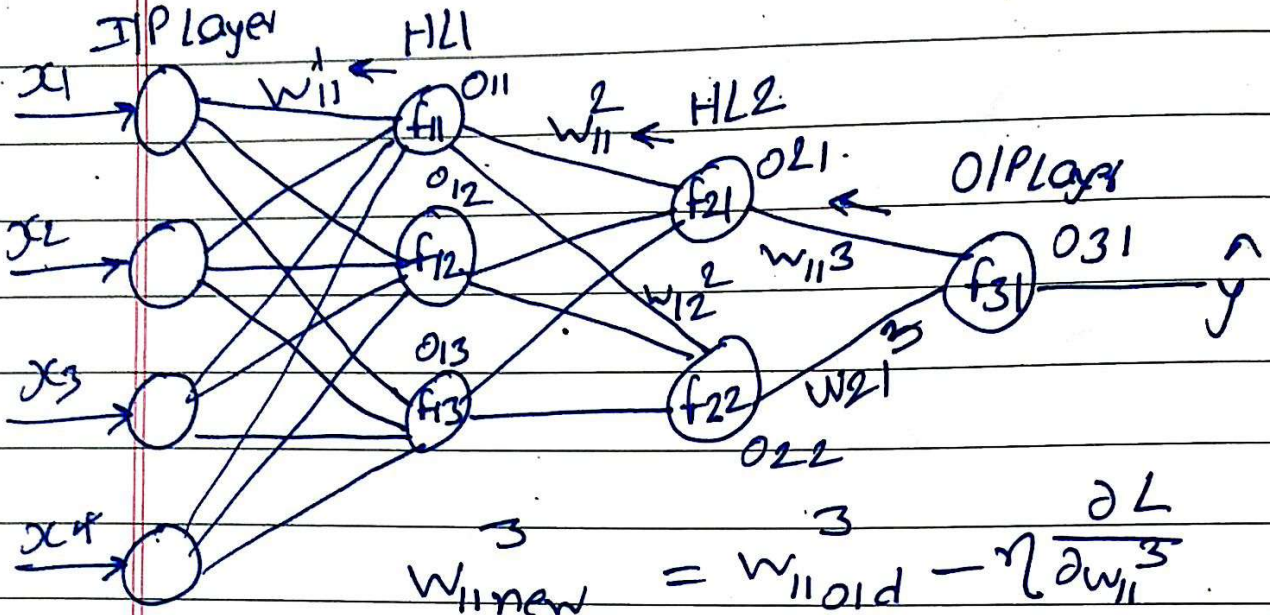
Minimize Loss
Loss $= \sum (y - \hat{y})^2$
Require Optimizer

$$y = [x_1 w_1 + x_2 w_2 + x_3 w_3] + b1$$
$$z = Act(y)$$

$$w_{3new} = w_{4old} - \eta \frac{\partial L}{\partial w_4} \qquad w_{3new} = w_{3old} - \eta \frac{\partial L}{\partial w_3}$$

η = Learning Rate

* Chain Rule In Backpropogation



$$W_{11new}^{3} = W_{11old}^{3} - \eta \frac{\partial L}{\partial w_{11}^{3}}$$

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

Reduce Loss we need optimizer (forex: Gradient Descent, Stoactistic Gradient) through backpropogation we update each & every weight of hidden layer

Now through backpropogation we need to update weight from back.

1) first find loss function
2) $W_{11new}^{3} = W_{11old}^{3} - \eta \frac{\partial L}{\partial w_{11}^{3}}$ ← slop of weight; η :- Learning Rate.

$$\frac{\partial L}{\partial w_{11}^{3}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^{3}} \quad \text{---Chain Rule}$$

classmate
Date
Page

3) for $w_{21}^{3}{}_{new} = w_{21}^{3}{}_{old} - \eta \frac{\partial L}{\partial w_{21}^{3}}$

$$\frac{\partial L}{\partial w_{21}^{3}} = \frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_{21}^{3}}$$

4) $w_{11}^{2}{}_{new} = w_{11}^{2}{}_{old} - \eta \frac{\partial L}{\partial w_{11}^{2}}$

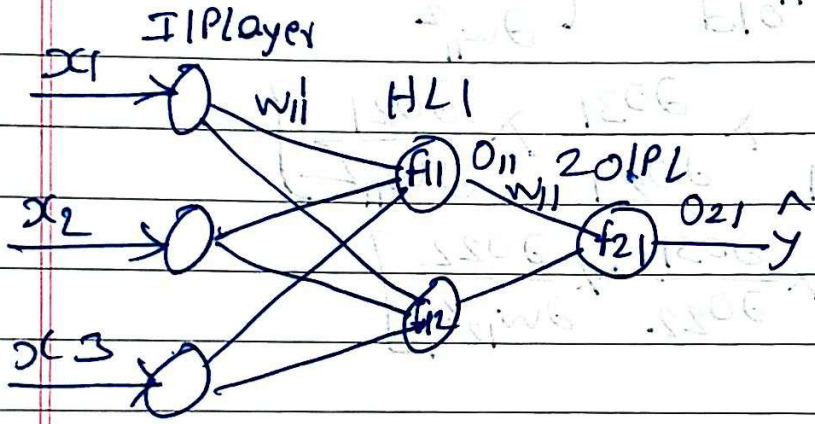
$$\frac{\partial L}{\partial w_{11}^{2}} = \left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{11}^{2}}\right] + \left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial w_{12}^{2}}\right]$$

This derivative means weights are updated unless & until we reach global minima.

1 Epoch consider both forward & backward propagation.

* Vanishing Gradient Problem

* In 1990 to 2000 research cannot create deep neural nlw because of vanishing gradient because at that time Relu is not invented.



Weight Update

$$w_{11New} = w_{11old} - \eta \cdot \frac{\partial L}{\partial w_{11old}}$$

1) At Layer 1 in neuron we do a summation of i/p multiply by weight & add bias.

$$f_{11in} = (x_1 w_{11} + x_2 * w_{12} + x_3 w_{13}) + bias.$$

$$\therefore O_{11} = \text{Activation function } (f_{11})$$

Here we use Sigmoid function which gives o/p in between 0 & 1

2) After getting first transformation this output get transfer to second layer & weight get

assign here w_{11}^2 & again through optimizer we get output by reduceing loss by updating weight.

Using chain rule & weight updation

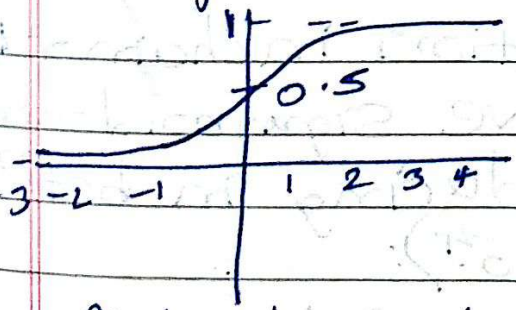
$$w_{11new}^1 = w_{11old}^1 - \eta \frac{\partial L}{\partial w_{11old}^1}$$

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_{11}^1} \leftarrow \text{chain rule}$$

Nor neuron o/p we use sigmoid activation function which transforms o/p in betⁿ 0 to 1

But derivative of sigmoid specifically betⁿ 0 to 0.25

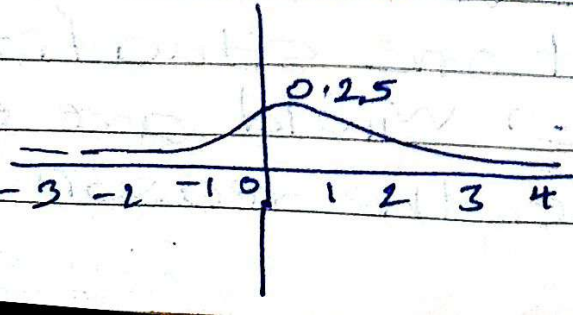
so Sigmoid is



$$z = \Sigma x w + b$$

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

But derivative of sigmoid function will look like this



which is inbetⁿ 0 to 0.25

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

So derivative of sigmoid is in range
$0 \leq \sigma(z) \leq 0.25$

As in out eqn

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_{11}^1}$$

$\frac{\partial o_{11}}{\partial w_{11}^1}$ 0.20 — 0.02 ← value of gradient

Suppose here using sigmoid fun^n which come in bet^n 0 to 0.25 after derivative it goes on reducing supose 0.04 for example $\frac{\partial L}{\partial w_{11}^1} = 0.04$

Original weight $2.5 = w_{11}^{old}$. $\eta = 1$

$$\therefore w_{11}^{new} = w_{11}^{old} - \eta \frac{\partial L}{\partial w_{11}^{old}}$$

$$= 2.5 - 1\,(0.04)$$

$$= 2.46$$

As we go on adding hidden layer & during weight updation in chain rule because of derivative sigmoid function value goes on reducing in bet^n 0 to 0.25 $\rightarrow (10^{-4})$

* So as the number of layer increase this derivative value is very smaller value and at one situation $w_{11}^{new} = w_{11}^{old}$ are equal & matching ie the reason sigmoid

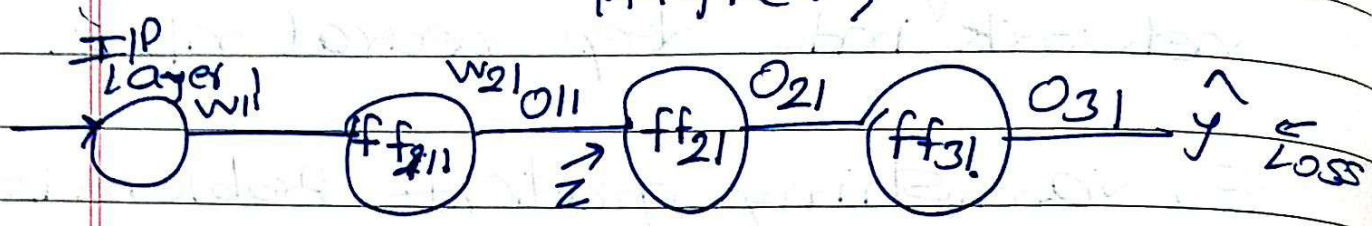
not work properly when we added more layer & convert neural network into deep neural n/w.

So vanishing gradient Problem is due to Sigmoid Activation function as value is in betⁿ 0 to 0.25

Due to this Relu Sigmoid function Comes.

For tanh transfer function also same problem comes.

* Exploding Gradient Problem (Derivative value become Higher)



$$w'_{11new} = w'_{11old} - \eta \frac{\partial L}{\partial w'_{11}}$$

$$\frac{\partial L}{\partial w'_{11}} = \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w'_{11}}$$

$z = w_{21} o_{11} + b_2$

$\therefore o_{21} = \phi(z)$ — ϕ is sigmoid activation function

$\therefore \frac{\partial o_2}{\partial o_{11}} = \frac{\partial \phi(z)}{\partial z} \times \frac{\partial z}{\partial o_{11}}$ — $0 \le \phi(z) \le 0.25$

for ex consider $= 0 \le \phi(z) \le 0.25 \times \frac{\partial (w_{21} o_{11} + b_2)}{\partial o_{11}}$

$= 0 \le \phi(z) \le 0.25 * w_{21}$ assue $w_{21} = 500$

$= 0.25 \times 500 = 125$

So weight Intiallisation is also very important.

If weights are intiallize to very high value we get variance high & it never get convergance to global minima

if $\frac{\partial L}{\partial w'_{11}} = 200 \times 125 \times 100$ which is very high value

due to which w'_{11} new is -ve as $\frac{\partial L}{\partial w'_{11}}$ is high & situation occurs of Exploding Gradient

classmate
Date
Page

* Drop Out & Regularization.

1) When we have many layer in artifical neural network & it is deep network then we have more no. of weights & bias parameter.

2) Due to with ANN tend to becomes overfit the datset problem or a particular use case so we should have to set weight to overcome fitting problem

3) In multilayer neural network we neverface underfitting problem. but overfitting may problem. & facing high variance problem

4) There are two ways to avoide overfitting problem.

1) Regularization by L1 & L2 Norm.

2) Drop Out

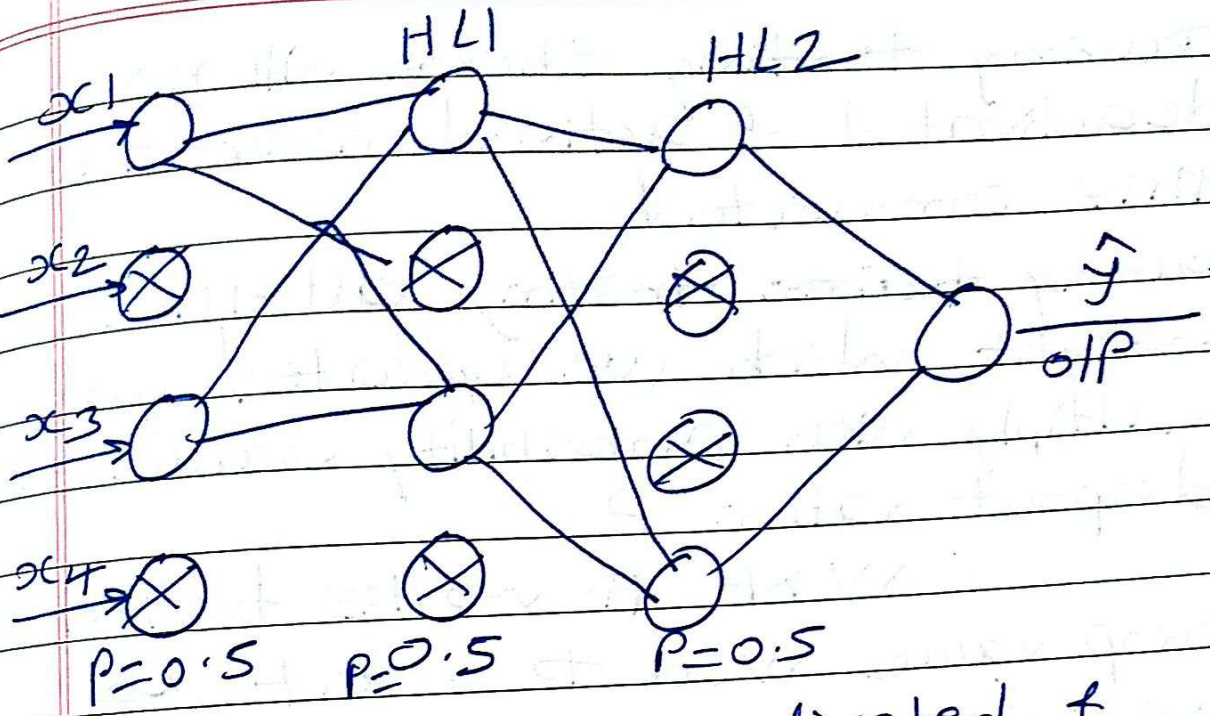
* Drop out comes in 2014. by
1) Nitsh Shrivastav } Thesis.
2) Jeoffery Hiton }

for ex- Random forest Tree
i) In decision tree when we try to reach in depth over fitting occurs
ii) So in random forest we use subset of features & form multiple tree to avoide overfitting & improve accurg

* Similarly in DropOut
To implement we select Drop Out ratio → P which is $0 \leq P \leq 1$

1) We select subset of feature from i/p layer, similarly subset of feature from hidden layer
2) So we are not selecting all feature we are selecting subset of feature along with i/p layer to hidden layer

So as shown in fig Supose P i.e Dropout Ration $P = 0.5$ & we select subset of feature.



∴ 2 Neuron are activated & 2 Neuron are deactivated

During forward & backward prpogation based on P value it will randomly deactived some neuron

In backprpogtion the neuron which are activated for that weight gets updated. Again in forward prpogation randomly some neuron activted & deactivated.

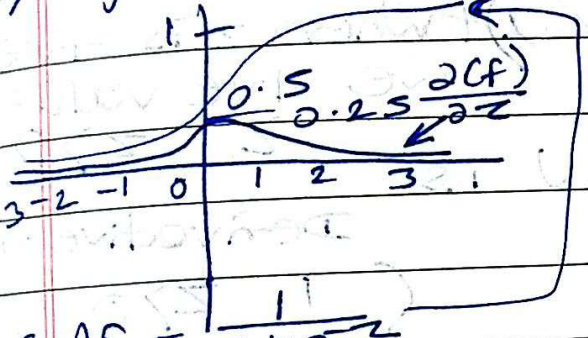
So to avoide overfitting we are deactivating & activating neuron or i/p feature by drop out ratio in training.

* During testing there will no deactivated & actived neuron every this connected
* During ~~train~~ testing all the weight which we updated gets multiply with probability value or dropout ration p
  ∴ W x p it works for test data.
* Drop value need to select as hyperparameter optimization

* Rectified Linear Unit / Leaky Relu

* Summary

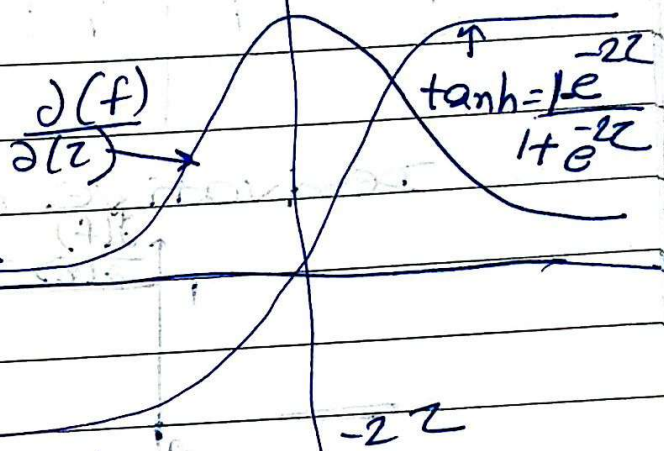
1) Sigmoid AF



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

$$0 \leq \frac{\partial(f)}{\partial z} \leq 0.25$$

2) Threshold Activation



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

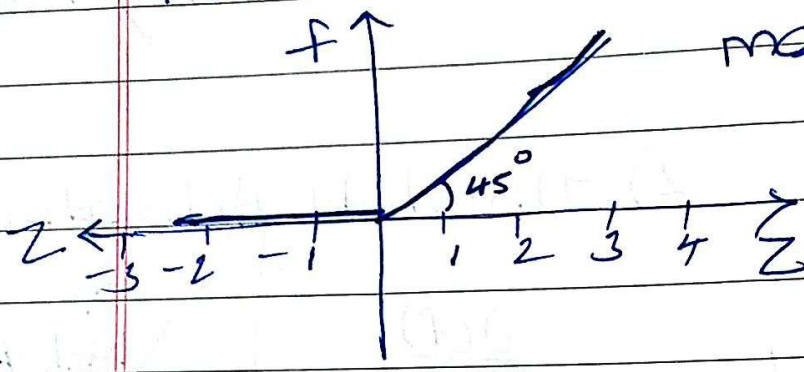
$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial w}$$

Here derivative is less than one during backpropogation & $W_{new} = W_{old}$

Because of this there is very minor change in W_{new} & it will take much time to conveiges to global minima

So this Vanishing Gradient Problem is handle by Relu activation function

RLU (Rectified Linear Unit)

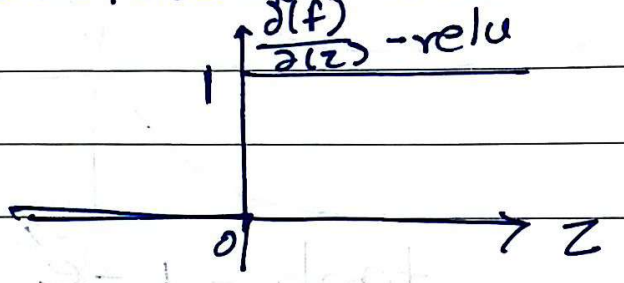


max (0, z)

1) When z is -ve
o/p = 0

2) When z is +ve we +ve value.
z z>0

Derivative of RLU is



Derivative of RLU

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

So in sigmoid allways derivative value is 0 to 0.25 & in tanh it is less than one But in RLU we have 1 & 0

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

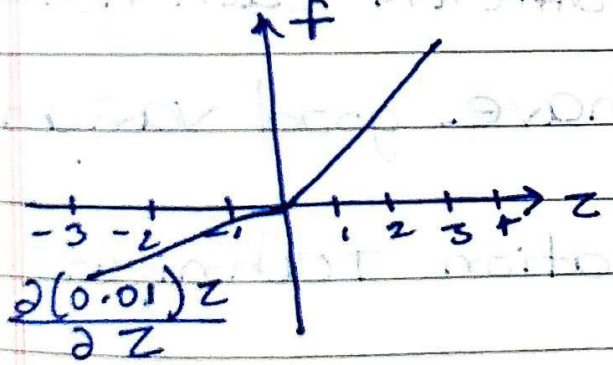
Because of this now $\frac{\partial L}{\partial w}$ is 1 no vanishing gradient proble & our weights gets convergens easily

Supose Derivative is 0, then we get $W_{old} = W_{new}$ as $\frac{\partial L}{\partial w} = 0$
So it will create dead neuron or dead activation function.

means no weight updation process happens so to handle this problem Leaky Relu use.

Leaky Relu

1) In Relu for -ve value of z o/p fun^n is zero.
But in place of zero during -ve value of z we are adding some value.



$$\begin{cases} z & z > 0 \\ 0.01(a) & z < 0 \\ 0.01(z). \end{cases}$$

$\frac{\partial(0.01)z}{\partial z} = 0.01$ So when z is -ve it derivative is 0.01, so it will not be zero & solving the problem of dead neuron.

Weight Initialization

1) Weights should be small (medium small)
2) Weight should not too small due to which vanishing gradient occurs

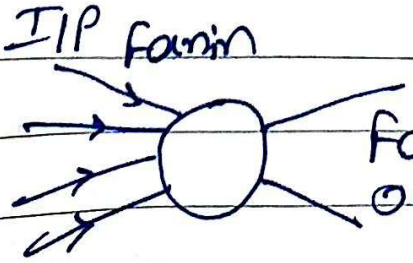
2) Weights should not be same
If weights are same the all neuron learning features in the same way.
So weight should be different each neuron learning new feature.

3) Weights should have good variance

Weight Initialization Techniques

1) Uniform Distribution
weeright taking from uniform distribution in range a (lower) & b (higher)

$$W_{ij} \sim \text{Uniform}\left[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanout}}\right]$$



fanin: No of I/Ps
fanout: - No of O/Ps

2) Xavier / Gorat

i) Xavier Normal or Gorat Normal
weight are taken from normal distribution with mean = 0, σ - Std. deviation or variance

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

$$\sigma = \sqrt{\frac{2}{fan\_in + fan\_out}}$$

ii) Xavier Uniform
Taken from uniform distribution

$$W_{ij} \sim U\left[\underbrace{\frac{-\sqrt{6}}{\sqrt{fan\_in + fan\_out}}}_{a}, \underbrace{\frac{\sqrt{6}}{\sqrt{fan\_in + fan\_out}}}_{b}\right]$$

This will work well with Sigmoid function

3) He init : Use full for Relu (RLU)

i) He Uniform
weight are basically taken from Uniform distribution

$$W_{ij} \sim U\left[\underbrace{-\frac{\sqrt{6}}{\sqrt{fan_{in}}}}_{a}, \underbrace{\frac{\sqrt{6}}{\sqrt{fanout}}}_{b}\right]$$

ii) He Normal : Use normal distribution

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

$$\sigma = \sqrt{\frac{2}{fan_{in}}}$$