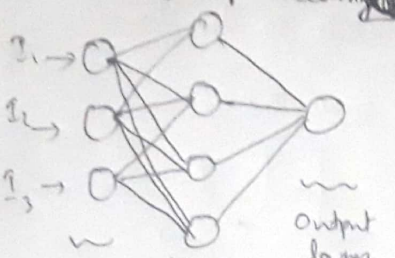
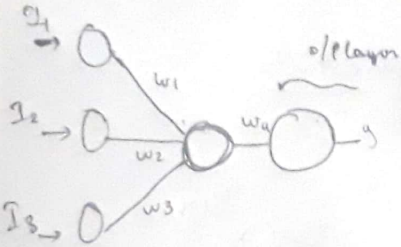
Deep learning





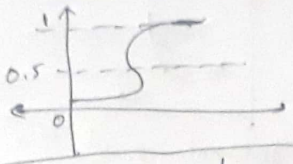
neuron

① $y = w_1 I_1 + w_2 I_2 + w_3 I_3 + bias$

② $z = Act(y)$

Activation function

① Sigmoid function



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

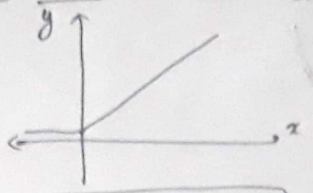
$$z = \sum_{i=1}^{n} w_i x_i + b_i$$

w = weights b = bias
x = input

- if sig. func value < 0.5 → 0
- if sig. func value > 0.5 → 1

0 → not activated
1 → activated.

② Relu → Rectified linear unit



$$Relu = max(z, 0)$$

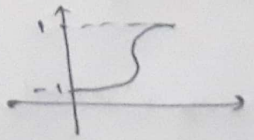
$$z = \sum_{i=1}^{n} w_i x_i + b_i$$

if Relu output is −ve → 0

if relu output is +ve → That corresponding value

- Relu popular than Sigmoid

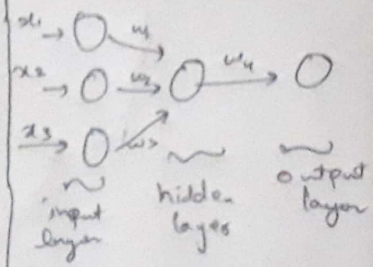
③ Hyperbolic tangent



range = −1 to 1

$$\tanh x = \frac{\sinh x}{\cosh x}$$

↳ Back propagation



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

y → actual
$\hat{y}$ → predicted

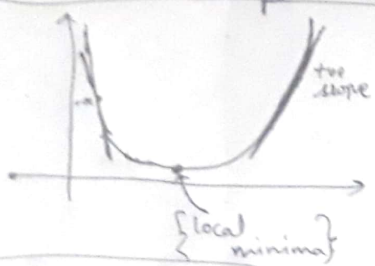
- We try to reduce the loss → optimisers
- We back propagate and change weights acc to optimiser.

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$$

α - learning rate

- Once the weight is modified again we do forward propagation and calculate loss.
- No of times we do this depends on no of epoches.

↳ Gradient Descent :- Optimiser



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

when slope is -ve : ie $\left\{\frac{\partial L}{\partial w} = -ve\right\}$

W_{new} = increases

- then it starts to converge to the local minima.
- How faster it does depends on α

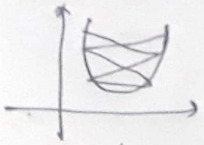
When slope is +ve : ie $\left\{\frac{\partial L}{\partial w} = +ve\right\}$

W_{new} = decreases

- then it starts to converge to local minima
- How faster it does depends on α

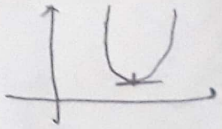
↳ α : Learning Rate : needs to be lesser

Higher α :



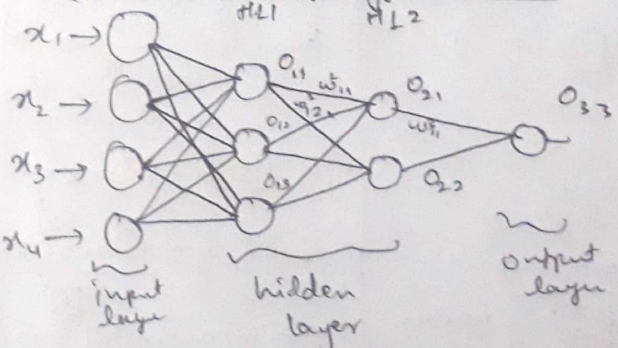
does not reach the local minima

Lower α :



Takes time, but best :)

↳ Chain Rule of differentiation :-



∴ To optimise, we backpropagate & change the weights using optimiser to $W_{new} = W_{old} - \alpha \frac{\partial L}{\partial w}$

Now how do we compute $\frac{\partial L}{\partial w}$? {using chain rule}

$$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial O_{33}} \cdot \frac{\partial O_{33}}{\partial w^3_{11}}$$

→ Chain rule

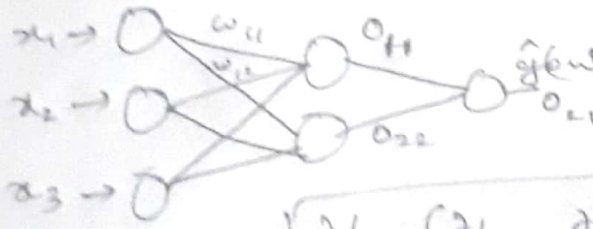
See where the weight is getting affected and change it

Eg: $\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial o_{33}} \cdot \frac{\partial o_{33}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w_{11}^2}$

But that neuron has also ~~connected~~ has another edge, so :-

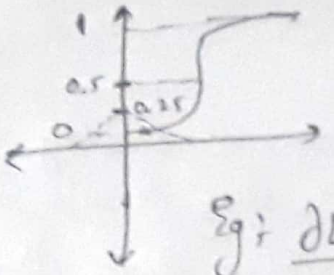
$$\frac{\partial L}{\partial w_{11}^2} = \left[\frac{\partial L}{\partial o_{33}} \cdot \frac{\partial o_{33}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w_{11}^2}\right] + \left[\frac{\partial L}{\partial o_{33}} \cdot \frac{\partial o_{33}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial w_{12}^2}\right]$$

1→ Vanishing gradient problem in Sigmoid



$$\frac{\partial L}{\partial w_1} = \left[\frac{\partial L}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_1}\right] + \left[\frac{\partial L}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial w_{12}}\right]$$

When the activation function is sigmoid the derivative of it ranges from 0 to 0.25



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

Eg: $\frac{\partial L}{\partial w_1^1} = \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_1^1}$

$0.20 \times 0.02 = 0.04$

$w_{1new} = w_{old} - \alpha \frac{\partial L}{\partial w}$ ($\alpha = 1$)

$= 2.5 - (0.04)$

$[w_{1new} = 2.46]$

As the layers increase, the ~~the~~ term $\alpha \frac{\partial L}{\partial w}$ becomes smaller ~~the~~ so that the effect of it is very very small becoming $w_{new} = w_{old}$ where the term is $-\alpha \frac{\partial L}{\partial w}$ is affecting very very small & so we call it as Vanishing gradient.

- So ~~the~~ Vanishing gradient problem is due to sigmoid

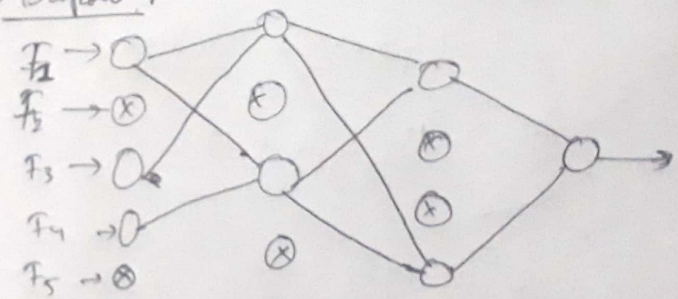
Exploding gradient problem:-

- In exploding gradient problem, generally the weight is given higher value due to which $\eta \frac{\partial L}{\partial w_{11}}$ has a bit higher value than the W_{new} & W_{old} will vary a lot.
- So that the gradient descent does not converge to local minima.

↳ Drop out & Regularization:-

- When more layers are there in Neural Network, overfitting tends to occur But we will never face underfitting.
- ~~Ways to solve Regularization~~: L_1, L_2, Dropout
- Ways to solve Overfitting: Regularization L_1, L_2, Dropout

↳ Dropout:-



Dropout ratio (P): $0 \leq P \leq 1$ → You assign P value by doing hyper parameter tuning

- During Traintime depending on the P value few neurons gets dropped out randomly (deactive) only the active neurons does the operation after 1 process, again randomly other neurons gets activated and performs training.
- So that only few neurons participate in train

So the neurons are trained during training time.

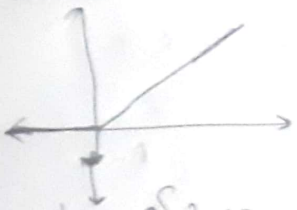
Test time: During the test time, all the neurons gets activated and connected.

Note for all the weights will be multiplied by the dropout ration in each & every layer

Weights = W x P .

Relu & Leaky Relu's

Relu:



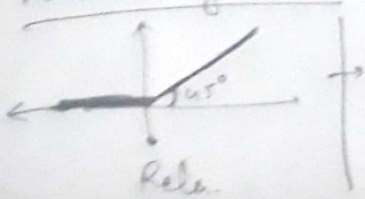
Relu = max(0, z)

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

Relu

when { z < 0 → 0 ; z > 0 → z }

Derivative of Relu:

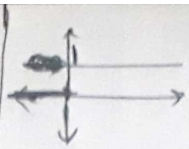


Derivative → slope

$m = \tan\theta$

$\theta = 45° \rightarrow m = 1$

$\theta = 0 \rightarrow m = 0$



deri = { 1 if z > 0 ; 0 if z < 0 }

Derivative of Relu

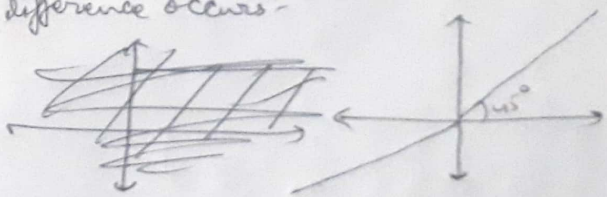
When derivate becomes 0 in z < 0 then:

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

then then $\frac{\partial L}{\partial w} = 0$ then $\eta \frac{\partial L}{\partial w} = 0$ therefore

dead neuron: $W_{new} = W_{old}$ so there is no change.

To overcome it we do Leaky relu ie to when z < 0 we multiply z with a small quantity (a). so therefore a small difference occurs.



Therefore { z if z > 0 ; 0.01(z) if z < 0 }

↳ Weight Initializations:-

key points

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

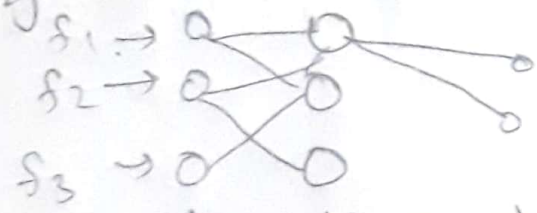
Techniques used: ①: Uniform Distribution

a , b

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

The weights gets assigned uniformly from the range [a, b]

Eg:-



$$\left[\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right] \& \left[\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{2}}\right]$$

②:- Xavier / Gorat Distribution:-

Works well with Sigmoid

Xavier Normal:-

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

$$\sigma = \sqrt{\frac{2}{(fan_{in} + fan_{out})}}$$

Xavier uniform:-

$$W_{ij} \sim U\left[\frac{-\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}\right]$$

③:- He init + (Relu)

He uniform:-

$$W_{ij} \approx U\left[\frac{-6}{fan_{in}}, \frac{\sqrt{6}}{fan_{in}}\right]$$

(a , b)

He normal:-

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

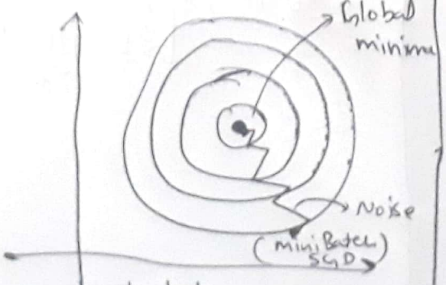
$$\sigma = \sqrt{\frac{2}{fan_{in}}}$$

↳ Stochastic Gradient Descent

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

Now, if :

- $\frac{\partial L}{\partial W_{old}}$ → When we consider all n points → gradient descent
- $\frac{\partial L}{\partial W_{old}}$ → 1 data point → SGD
- $\frac{\partial L}{\partial W_{old}}$ → k data points [k<n] → mini Batch SGD
- $\left[\frac{\partial L}{\partial W_{old}}\right]_{\text{mini Batch SGD}} \approx \left[\frac{\partial L}{\partial W_{old}}\right]_{GD}$



There is a noise produced in mini batch Stochastic GD

To remove it, we use SGD with momentum

↳ Global Maxima & Local Minima

SGD with momentum :-

- SGD from above it takes more time to converge that is due to the noise so we need to remove it.
- Exponential moving average is used to remove the noise

time	t_1	t_2	t_3	t_4	...	t_n
data point	b_1	b_2	b_3	b_4	..	b_n

$V_{t_1} = b_1$ $\quad \{0 \le \gamma \le 1\}$

$V_{t_2} = (\gamma)(V_1) + b_2$

~~$V_{t_2} = (\gamma)(b_1) + b_2$~~

$V_{t_3} = \gamma V_2 + b_3$

$= \gamma [\gamma b_1 + b_2] + b_3$

$= \gamma^2 b_1 + \gamma b_2 + b_3$

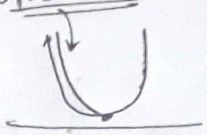
$V_{t_3} = \gamma^2 b_1 + \gamma b_2 + b_3$

when $\gamma = 0.5$

$V_{t_3} = 0.25 b_1 + 0.5 b_2 + b_3$

more importance is given to b_3

Updated SGD



So,

So formula becomes :-

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

$$W_{old} - \left[\gamma V_{t-1} + \eta \frac{\partial L}{\partial W_{old}} \right]$$

$$V_{t-1} = 1 * \left[\frac{\partial L}{\partial W_{old}}\right]_t + \gamma \left[\frac{\partial L}{\partial W_{old}}\right]_{t-1} + \gamma^2 \left[\frac{\partial L}{\partial W_{old}}\right]_{t-2}$$

↳ Adagrad Optimizer:-

- In GD, SGD, Mini batch SGD we used to have the η (learning rate) fixed to each and every neuron.
- But in Adagrad optimizer, we have η varying learning rate at each ~~epoch~~ iterations.

$$W_t = W_{t-1} - \eta'_t \left(\frac{\partial L}{\partial W_{t-1}}\right)$$

$t \rightarrow$ current, $t-1 \rightarrow$ previous.

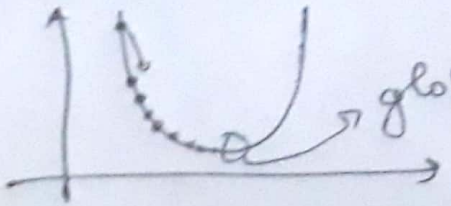
$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

$\eta \rightarrow$ initial LR

$\varepsilon \rightarrow$ small tve number

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

Starts with high value then η'_t decreases so W_t also varies.



Disadvantage with Adagrad:-

- the α_t parameter gets increased a lot after some time.

↳ Adadelta & Rmsprop:-