$ANN$

1) $Architecture\ of\ ANN$
2) $Input\ layer\ -\ Hidden\ layer\ -\ Output\ layer$
3) $Neuron\ =\ Summation\ +\ Activation$
4) $Forward\ prpogation:\ \ Output$
5) $Back\ prpogation:\ \ To\ update\ the\ weights$
6) $Gradient\ descent\ algorithm:\ \ w_{new} = w_{old} - lr * \frac{dj}{dw_{=}w_{old}}$
7) $Gradient\ descent\ implementaion\ using\ python$

1) $How\ to\ update\ weights:\ \ \ Completed$

2) $Activation\ functions:$

$Activation\ Functions$

$We\ \ know\ that\ inside\ Neuron:\ \ Summation\ +\ Activations$

$Summation\ =\ Linear\ combination\ of\ inputs\ =\ y\ =\ b\ +\ w\ *\ x$

$Linear\ combination\ means\ it\ is\ just\ a\ Linear\ regressio\ problems$

$Just\ pass\ the\ inputs$

$It\ will\ not\ able\ to\ identify\ the\ complex\ problems$

$Complex\ problems\ ======\ Non\ lineairity$

$classification\ problems,\ the\ linear\ combination\ of\ inputs\ is\ not\ enough$

$we\ need\ to\ include\ the\ Non\ linearity:\ \ Activation\ functions$

1) $Sigmoid$

2) $Softwax$

3) $tanh$

4) $ReLU: Rectified\ Linear\ Unit$

5) $Leaky\ ReLU$

$A$) $what\ is\ the\ equation$

$B$) $what\ is\ graph$

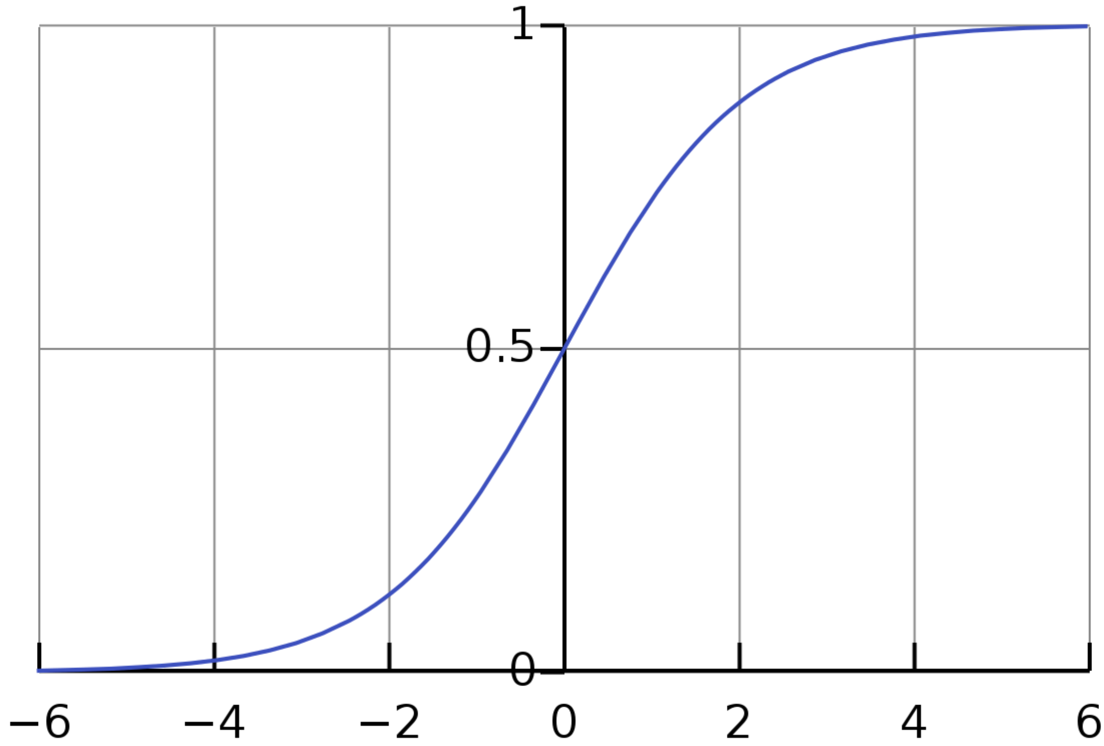
$C$) $what\ is\ the\ range\ of\ the\ equation$

*D) where to use*

*Sigmoid:*

*what is the equation:* $\sigma(x) = \frac{1}{1+e^{-x}}$

*what is the Graph:*




*what is the Range:* $0$ *to* $1$ *and it is centered at* $0.5$

$\sigma(-\ inf) = \frac{1}{1+e^{-(-inf)}} = 0$

$\sigma(0) = \frac{1}{1+e^{-(0)}} = 0.5$

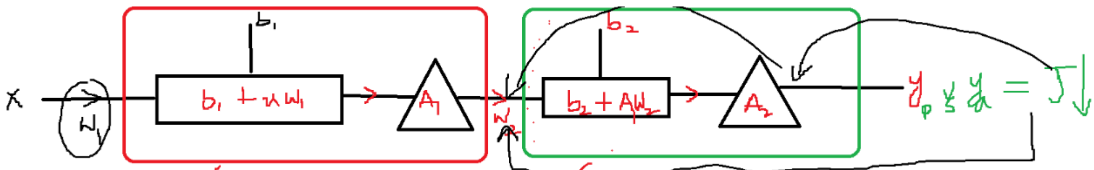
$\sigma(+\ inf) = \frac{1}{1+e^{-(inf)}} = 1$

*where to use:*

*Sigmoid function use at Output layer, for Binary classification problem* *why not at Hidden layer*

*what happens if you use sigmoid at hidden layer*

*Vanish Gradients and Explode gradients*




$A_1 = \sigma(b_1 + xw_1)$ $\quad$ $A_2 = \sigma(b_2 + A_1 W_2)$

$$\frac{dJ}{dW_2} = \frac{dJ}{dA_2} \times \frac{dA_2}{dW_2} \quad \text{(Chain Rule)}$$

$$\frac{dJ}{dW_1} = \frac{dJ}{dA_2} \times \frac{dA_2}{dW_2} \times \frac{dW_2}{dA_1} \times \frac{dA_1}{dW_1}$$

$$\frac{dJ}{dW_2} = \frac{dJ}{dA_2} * \frac{dA_2}{dW_2}$$

$$\frac{dJ}{dW_1} = \frac{dJ}{dA_2} * \frac{dA_2}{dW_2} * \frac{dW_2}{dA_1} * \frac{dA_1}{dW_1}$$

*while updating the weights, we need to calculate Gradients (Slope) of (J) w.r.t weights*

*In this process we also applying a Chain rule, so in that we are finding*

$$the\ slope\ of\ activation\ function\ i.e.\ \frac{dA_2}{dW_2}\ \ and\ \frac{dA_1}{dW_1}$$

$Slope\ of\ activation\ functions$:

## $slope\ of\ sigmoid\ function$

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

$$\frac{d\sigma}{dx} = \sigma^{'}(x) =$$

$$\frac{dy}{dx} = \frac{V\frac{dU}{dx} - U\frac{dV}{dx}}{V^2} \qquad \frac{1}{1+e^{-x}} = \frac{U}{V}\ \ where\ U\ =\ 1;\ \ V\ =\ 1\ +\ e^{-x}$$

$$\frac{\left(1+e^{-x}\right)\frac{d1}{dx} - 1*\frac{d(1+e^{-x})}{dx}}{\left(1+e^{-x}\right)^2} = \frac{0-1*\frac{d(1)}{dx} - \frac{d\left(e^{-x}\right)}{dx}}{\left(1+e^{-x}\right)^2}$$

$$\frac{0-\frac{d\left(e^{-x}\right)}{dx}}{\left(1+e^{-x}\right)^2} = \frac{e^{-x}}{\left(1+e^{-x}\right)^2}$$

$$\frac{d\sigma}{dx} = \sigma^{'}(x) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2} = \frac{e^{-x}}{1+e^{-x}} * \frac{1}{1+e^{-x}}$$

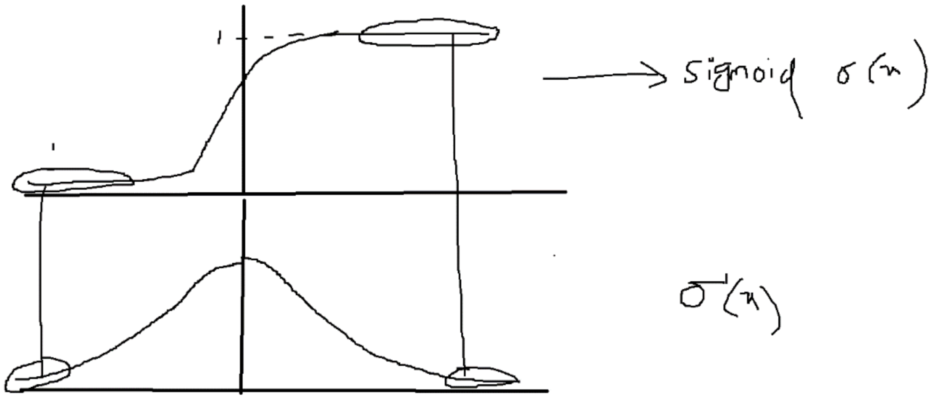
$$\frac{e^{-x}}{1+e^{-x}} = \frac{1+e^{-x}-1}{1+e^{-x}} = \frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} = 1\ -\ \frac{1}{1+e^{-x}}$$

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

$$\frac{d\sigma}{dx} = \sigma^{'}(x) = \left(1\ -\ \frac{1}{1+e^{-x}}\right) * \left(\frac{1}{1+e^{-x}}\right)$$

$$\frac{1}{1+e^{-x}} = sigmoid\ =\ s$$

$$\frac{d\sigma}{dx} = \sigma^{'}(x) = (1\ -\ s) * s$$




*Differentiation of Sigmoid function becomes zero, when w is very large (Explode)*

*Differentiation of sigmoid function becomes zero, when w is very small (Vanish)*

*Differentiation= slope becomes zero = Gradients becomes zero*

*Gradients are vanished*

$$\frac{dA_2}{dW_2} = 0$$

$$\frac{dJ}{dW_2} = \frac{dJ}{dA_2} * 0 = 0$$

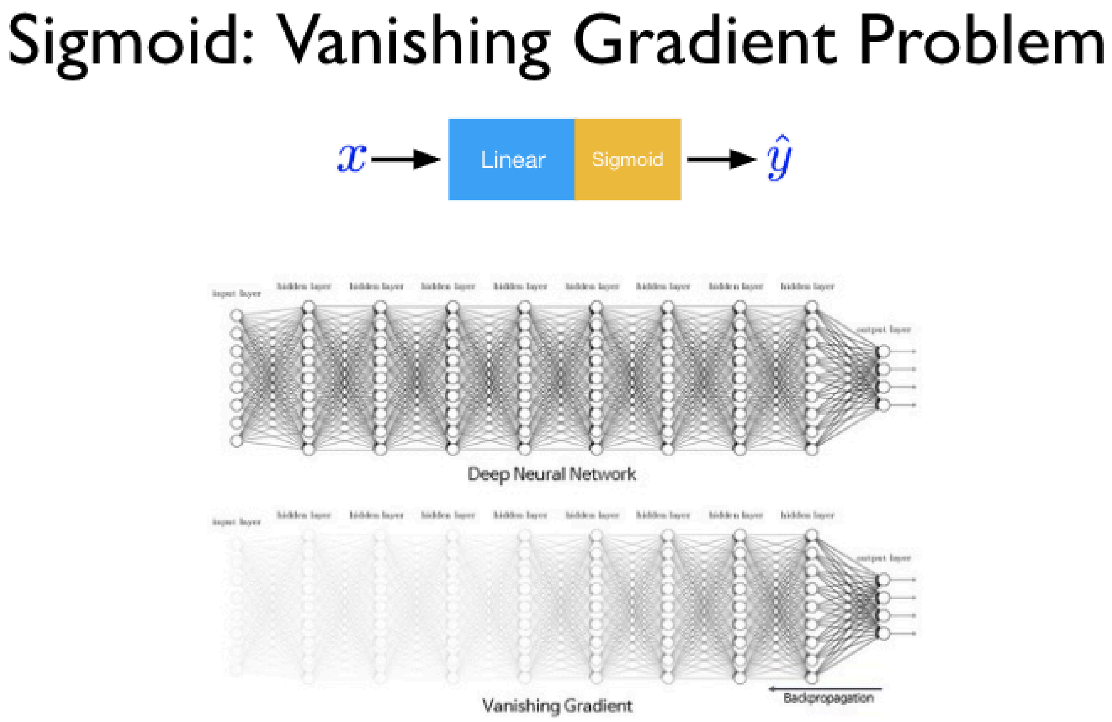
$$W_{2_{new}} = W_{2_{old}} - lr * \frac{dJ}{dW_2}$$

$$W_{2_{new}} = W_{2_{old}} - lr * 0$$

$W_{2_{new}}$ $will\ never\ updated\ ===\ NN\ will\ never\ learn\ any\ patterns$

$It\ wil\ not\ give\ desired\ output$

$\frac{dJ}{dW_1} = \frac{dJ}{dA_2} * 0 * \frac{dW_2}{dA_1} * 0$



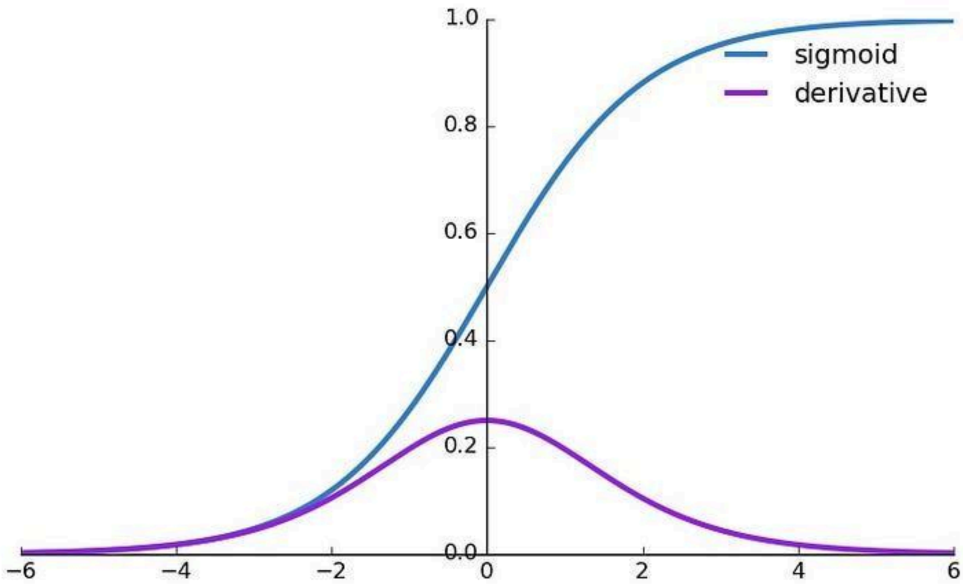

$If\ you\ use\ sigmoid\ function\ at\ hidden\ layer$

$while\ back\ prpogating\ ,\ the\ slope\ of\ sigmoid\ becomes\ zero\ for\ small\ w\ value$

$then\ slope\ becomes\ zero\ ===>\ the\ slope\ of\ J\ =\ 0$

$then\ slope\ of\ J\ =\ 0\ ===\ \ >\ weight\ updation\ not\ possibile$

$weight\ updation\ affects\ ===\ learning\ the\ patterns\ are\ not\ possibile$




2) $Softmax$

3) $tanh$

$Sigmoid ==== The\ same\ conecpt\ applied\ for$

$softmax\ and$ tanh $tanh$

$all$ 3 $at\ output\ layer\ only$

$equation === graph === range === differentation\ graph$

EWA : Exponential weighted average

Optimization ===== will not

No human being

RMS Prop ==== anse === we are rejecting

x_new = x – alpha * f'(x)

$$v(t) = \beta * v(t-1) + (1-\beta) * \delta(t)$$

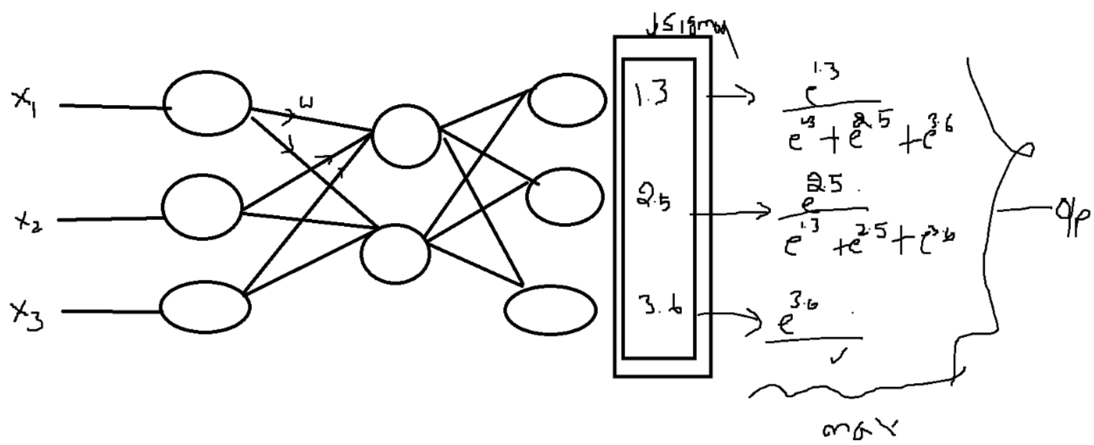
$We\ completed\ Sigmoid\ activation\ function$

2)$Softmax$

$Equation\ =$

$suppose\ there\ 3\ classes\ are\ there\ c_1,\ c_2, c_3$

$in\ the\ output\ layer\ we\ have\ 3\ neurons\ available$



$$p(c_1) = e^{c_1}/(e^{c_1} + e^{c_2} + e^{c_3})$$

$$p(c_2) = e^{c_2}/(e^{c_1} + e^{c_2} + e^{c_3})$$

$$p(c_3) = e^{c_3}/(e^{c_1} + e^{c_2} + e^{c_3})$$

$$output = max\{p(c_1), p\left(c_2\right), p(c_3)\}$$

$$softmax(z) = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

$Graph$: $look\ like\ sigmoid\ only$

$Range$: $0\ to\ 1$

$Problem$: $Vanish\ gradients$

$use\ case$: $at\ output\ layer\ for\ multi\ classification$

3)$Tanh$:

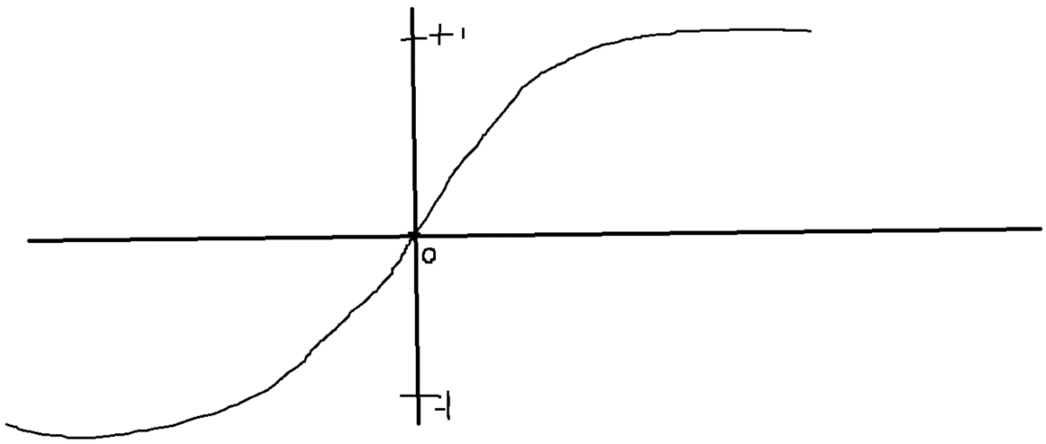
$$Equation\ of\ tanh = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

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

$$\sinh sinh\ (z) = (e^{z} - e^{-z})/k$$

$$\cosh cosh\ (z) = (e^{z} + e^{-z})/k$$

$Range = \quad -1 \; to \; 1$




$Output \;\; either \; 0 \;\; (No) \; or \;\; 1(Yes) === \; sigmoid$

$output \;\; either \;\; -1 \; (No) \; or \;\; 1(Yes) === \; tanh$

tanh $tanh \; is \; zero \; centred \; , \;\; which \; means \; output \; easily \; optimized$

$instead \; of \; sigmoid \; you \; can \; use \; tanh$

$sigmoid$

$softmax$

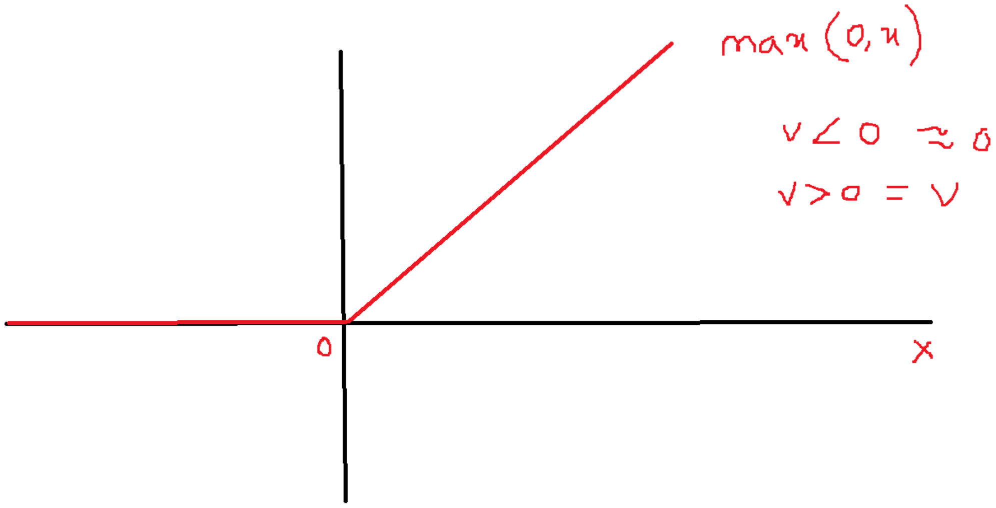
tanh $tanh \;\; all \; are \; at \; output \; layers \;\; and \; all \; are \; having \; draw \; back \; of \; vanish \; gradient$

4) *ReLU( Rectified Linear Unit)*

*it is used at hidden layer, because the gradients of ReLU function never zero*

*if you calculate the slope of ReLU function it never zero*

*It avoid vansih gradients problem*




*psuedo code*:

*if input* $> 0$:

    *input*

*else*:

    *zero*

*suppose input value im passing* $= 50$ *for NN* $==== \quad > 50$

*suppose input value* $=- \; 50$ *for NN* $==== \quad > 0$

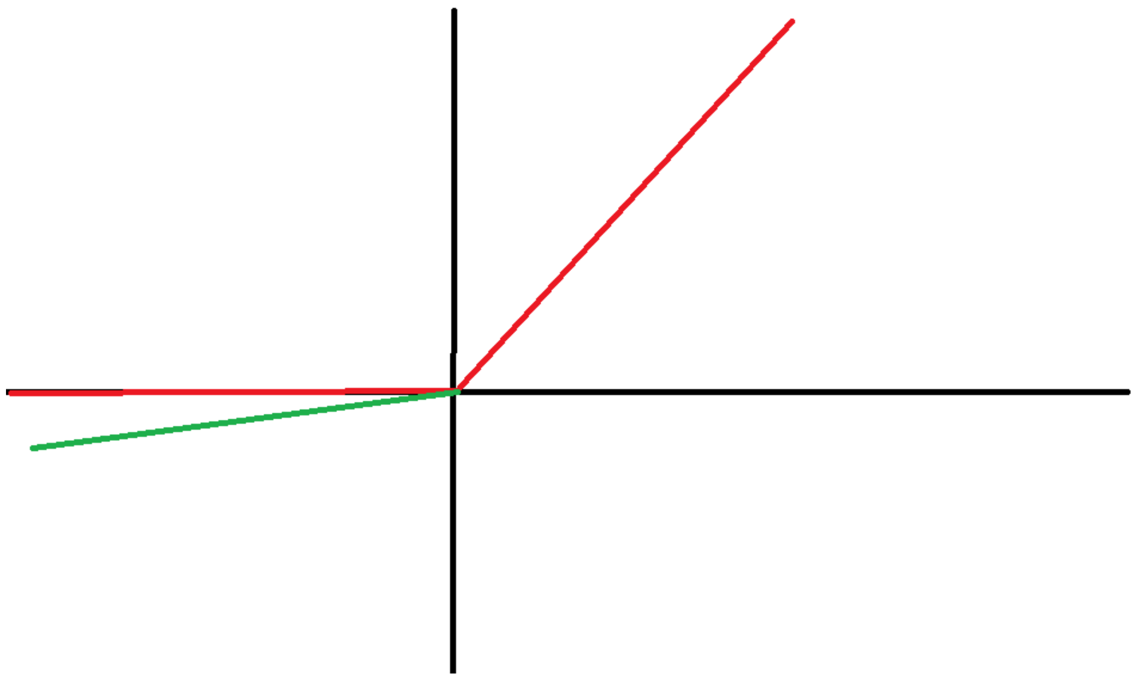
$Use\ case$:  $The\ slope\ never\ zero,\ so\ it\ is\ used\ at\ hidden\ layer$

$Draw\ back$ :  $assume\ that\ all\ the\ input\ values\ are\ negative\ values$

$ReLU\ treat\ as\ negative\ values\ as\ zero$

$we\ multilpy\ with\ zero$ ,  $entire\ NN\ will\ fail$

5) $Leaky\ ReLU$



$(ax, x)$ $a$ $can$ $be$ $anyting$ $a = 0.2$ $or$ $0.5$

$sigmoid = \frac{1}{1+e^{-x}}$ $0$ $to$ $1$ $output$ $layer$ $Bi$ $class$ $Vanish$ $gradients$

$softmax = \frac{e^{x_i}}{\Sigma e_i^{x}}$ $0$ $to$ $1$ $output$ $layer$ $Multi$ $class$ $Vanish$ $gradients$

$tanh = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ $-1$ $to$ $1$ $output$ $layer$ $or$ $Hidden$ $layer$

$bi$ $class$ $Vanish$ $gradients$

$ReLU = (0, x)$ $0$ $to$ $\inf$ $inf$ $hidden$ $layer$ $No$ $Vanish$

$all$ $inputs$ $negtaive$

$LeakyReLU = (ax, x)$