# Deep Learning-IV
## (Activation Functions)

DR. JASMEET SINGH,
ASSISTANT PROFESSOR,
CSED, TIET

# Activation Functions

- The activation function is **a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output**.
- Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it.
- The purpose of the activation function is to **introduce non-linearity** into the output of a neuron.

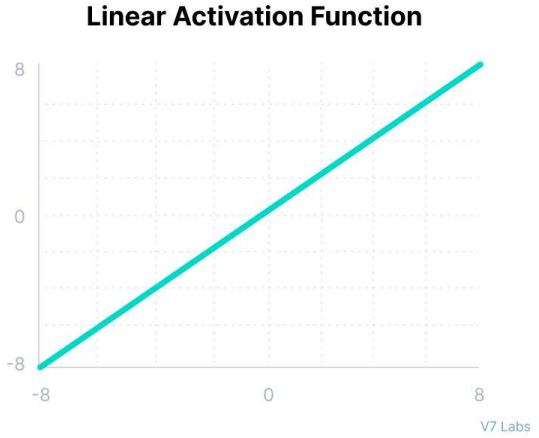
# Types of Activation Functions

Various types of activation functions used in neural networks are:

1. Linear Activation Function
2. Sigmoid Activation Function
3. Tanh (Hyperbolic Tangent) Activation Function
4. Rectified Linear Unit (ReLU) Activation Function
5. Leaky ReLU Activation Function
6. Parametric ReLU Activation Function
7. Softmax Activation Function

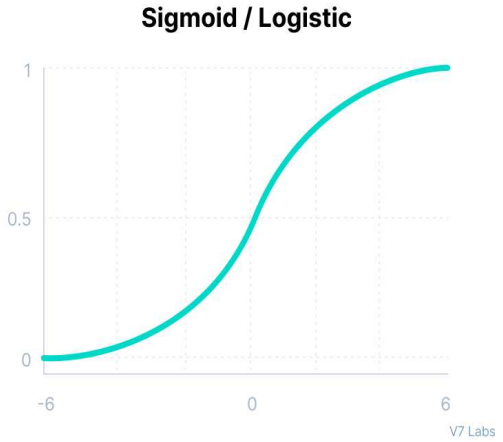
# Linear Activation Function

- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = wx+b**
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.
- **Issues: (Why Linear Activation functions are not used?)**
  - It's not possible to use backpropagation as the derivative of the function is a constant and has no relation to the input x.
  - All layers of the neural network will collapse into one if a linear activation function is used. No matter the number of layers in the neural network, the last layer will still be a linear function of the first layer. So, essentially, a linear activation function turns the neural network into just one layer.



# Sigmoid Activation Function

- It is a function which is plotted as **'S'** shaped graph.
- **Equation :** $A = \frac{1}{1+e^{-z}}$
- **Value Range :** (0,1)
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.
- Derivative: $A' = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}.\left(1 - \frac{1}{1+e^{-z}}\right) = A(1-A)$



# Sigmoid Activation Function (Contd.....)

**Limitations:**

**1. Vanishing Gradient.**

- The derivative of the function is : f'(x) = sigmoid(x)*(1-sigmoid(x)).

Since $\sigma(x) \in (0, 1)$ ,the product $\sigma(x)\big(1-\sigma(x)\big)$is always positive and reaches its maximum at $\sigma(x) = 0.5$.

**Maximum value of derivative:**

$\sigma'(0) = 0.5(1-0.5) = 0.25$

**Therefore, range of derivative is:** $\boxed{(0,\ 0.25]}$

During backpropagation, gradients are multiplied layer by layer. If activation functions like **sigmoid is** used:

Its derivatives is at most $\leq$ **0.25.** Often become very **tiny (close to 0)** for large positive or negative input. Multiplying many small gradients across layers results in an **exponentially shrinking gradient**

Small *number* $\times$ *Small number* $\times \cdots \rightarrow 0$ This problem is referred as vanishing gradient as The network can't update earlier layers because gradients almost disappear by the time they reach them.

# Sigmoid Activation Function (Contd.....)

**Limitations:**

- **2. Output not zero-centered**
  - Sigmoid outputs are in **(0,1)**.
  - Gradients always have the same sign, which causes **zig-zag updates** during optimization.
  - Makes training slower compared to zero-centered activations like **tanh**.
- **3. Saturation at extreme values**
  - For large positive or negative inputs, sigmoid outputs saturate near 1 or 0.
  - Derivative becomes almost zero, causing gradient to vanish.
  - Neurons stop learning when in saturated zone.
- **4. Computationally expensive**
  - Requires computing $e^{-x}$ which is slower than operations used in ReLU.

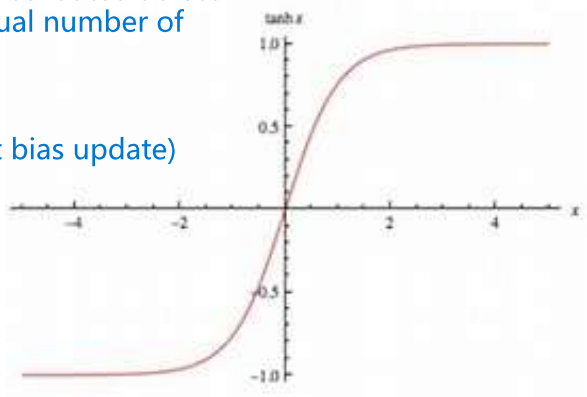
# Tanh (Hyperbolic Tangent) Activation Function

- Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1.
- It's actually mathematically shifted version of the sigmoid function.
- **Equation:** $A = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- **Value Range :-** (-1,1)
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

Since data's mean is zero hence data is evenly distributed across zero i.e equal number of positive values and equal number of negative values
so gradient flows in both direction
convergence occurs faster
weights update becomes balance ( no constant bias update)

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

Mathematical formula of the Tanh function



# Tanh (Hyperbolic Tangent) Activation Function (Contd…)

- **Derivative**: $A' = \frac{(e^x+e^{-x})(e^x+e^{-x})-(e^x-e^{-x})(e^x-e^{-x})}{(e^x+e^{-x})^2} = \frac{4}{(e^x+e^{-x})^2} = \frac{2}{(e^x+e^{-x})} \cdot \frac{2}{(e^x+e^{-x})} = \frac{1}{\cosh x} \frac{1}{\cos}$
$= sech^2 x = (1 - tanh^2 x) = \mathbf{1 - A^2}$

- **Limitation:**
  - It also faces the problem of vanishing gradients similar to the sigmoid activation function.

**Range of derivative:**

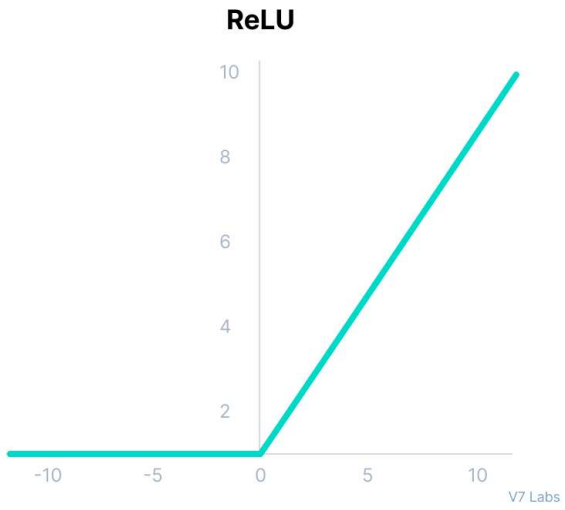
Maximum value = 1 (at $x = 0$). Approaches 0 when | $x$ | becomes large

$$\boxed{0 < tanh'(x) \leq 1}$$

- Although both sigmoid and tanh face vanishing gradient issue, tanh is zero centered, and the gradients are not restricted to move in a certain direction. Therefore, in practice, tanh nonlinearity is always preferred to sigmoid nonlinearity.

# ReLU Activation Function

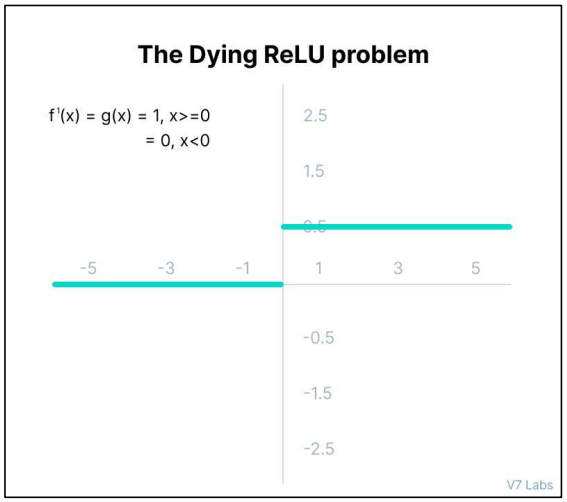
- ReLU stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Uses :-** ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.



# ReLU Activation Function (Contd.…)

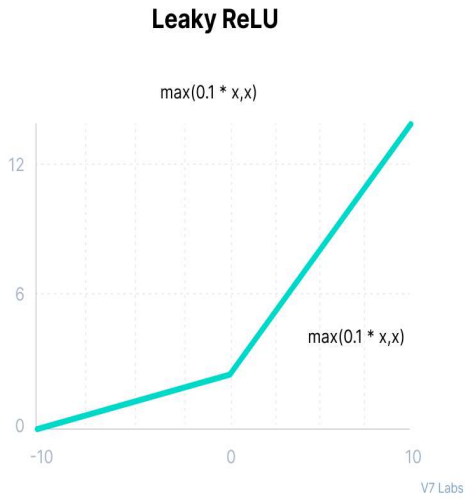
- **Derivative:** $A' = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x < 0 \\ not \; defined \; at \; x = 0 \end{cases}$

- **Limitations**:
  - The Dying ReLU problem: The negative side of the graph makes the gradient value zero. Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated.
  - All the negative input values become zero immediately, which decreases the model's ability to fit or train from the data properly.



# Leaky ReLU Activation Function

- Leaky ReLU is an improved version of ReLU function to solve the Dying ReLU problem as it has a small positive slope in the negative area.
- **Equation**: $A=\begin{cases} x \; if \; x > 0 \\ 0.01x \; if \; x \leq 0 \end{cases}$
- **Uses:** The advantages of Leaky ReLU are same as that of ReLU, in addition to the fact that it does enable backpropagation, even for negative input values.
- **Derivative:** $A' = \begin{cases} 1 \; if \; x > 0 \\ 0.01 \; if \; x < 0 \\ not \; defined \; at \; x = 0 \end{cases}$
- **Limitations**:
  - The predictions may not be consistent for negative input values.
  - The gradient for negative values is a small value that makes the learning of model parameters time-consuming.

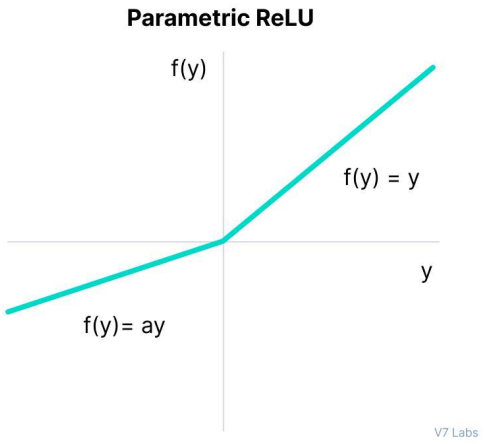



If u see the formula w=w-alpha*graidient u will see if gradient is small then learning becomes small

here negative slope is learnable parameter a and not fixed
leaky and parametric relu both prevent dead neurons but parametric relu also makes sure optimized slope for each neuron

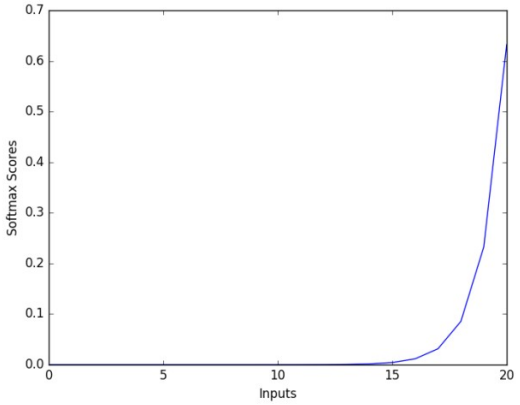
# Parametric ReLU Activation Function

- Parametric ReLU is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis.
- **Equation:** **f**(x) =max(ax, x); Where *"a"* is the slope parameter for negative values.
- **Derivative:** $A' = \begin{cases} 1 \; if \; x > 0 \\ a \; if \; x < 0 \\ not \; defined \; at \; x = 0 \end{cases}$
- **Uses:**
  - The parameterized ReLU function is used when the leaky ReLU function still fails at solving the problem of dead neurons, and the relevant information is not successfully passed to the next layer.
- **Limitations:**
  - This function's limitation is that it may perform differently for different problems depending upon the value of slope parameter ***a.***



# Softmax Activation Function

- The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems.
- **Equation**: $A(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$
- **Uses**: Usually used when trying to handle multiple classes. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.



# Derivative of Softmax Function

$$A(z_i) = s_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} = \frac{e^{z_i}}{D} \; where \; D = \sum_{j=1}^{n} e^{z_j}$$

$$\frac{\partial A(z_i)}{\partial z_k} = \frac{D\frac{\partial e^{z_i}}{\partial z_k} - e^{z_i}\frac{\partial D}{\partial z_k}}{D^2} \text{ ---------(1)}$$

$\frac{\partial e^{z_i}}{\partial z_k} = \begin{cases} e^{z_i} \;\; if \; i = k \\ 0 \; if \; i \neq k \end{cases} = \delta_{ik} e^{z_i}$ where $\delta_{ik}$ is called Kronecker delta and $\delta_{ik} = \begin{cases} 1 \; if \; i = k \\ 0 \; if \; i \neq k \end{cases}$

$$\frac{\partial D}{\partial z_k} = e^{z_i}$$

From 1 $\frac{\partial A(z_i)}{\partial z_k} = \frac{D\delta_{ik} e^{z_i} - e^{z_i} e^{z_k}}{D^2} = \frac{D\delta_{ik}\, Ds_i - Ds_i Ds_k}{D^2} = \delta_{ik} s_i - s_i \, s_k = \begin{cases} s_i(1 - s_i) \; if \; i = k \\ -s_i \, s_k \;\; if \; i \neq k \end{cases}$

This can be represented as Jacobian matrix form as diag(s) - $ss^T$ where s is $[s1,s2,\ldots,sn]^T$

# Derivative of Softmax Function (Contd…)

Let s = [s1,s2,…,sn]$^T$ be the softmax output vector

1. Diagonal Matrix: diag(s):

$$diag(s) = \begin{bmatrix} s_1 & 0 & 0 & \dots & \dots & 0 \\ 0 & s_2 & 0 & \dots & \dots & 0 \\ 0 & 0 & s_3 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \dots & \vdots \\ 0 & 0 & 0 & \dots & \dots & s_n \end{bmatrix}$$

2. Outer Product ss$^T$ :

$$ss^T = \begin{bmatrix} {s_1}^2 & s_1 s_2 & s_1 s_3 & \dots & \dots & s_1 s_n \\ s_2 s_1 & {s_2}^2 & s_2 s_3 & \dots & \dots & s_2 s_n \\ s_3 s_1 & s_3 s_2 & {s_3}^2 & \dots & \dots & s_3 s_n \\ \vdots & \vdots & \vdots & \dots & \dots & \vdots \\ s_n s_1 & s_n s_2 & s_n s_3 & \dots & \dots & {s_n}^2 \end{bmatrix}$$

# Derivative of Softmax Function (Contd…)

3. Jacobian J: Subtracting 1 and 2, the Jacobian becomes

$$J = diag(s) - ss^T = \begin{bmatrix} s_1(1-s_1) & -s_1s_2 & -s_1s_3 & \dots\dots\dots & \dots\dots & -s_1s_n \\ -s_2s_1 & s_2(1-s_2) & -s_2s_3 & \dots\dots\dots & \dots\dots\dots & -s_2s_n \\ -s_3s_1 & -s_3s_2 & s_3(1-s_3) & \dots\dots\dots & \dots\dots\dots & -s_3s_n \\ \vdots & \vdots & \vdots & \dots\dots & \dots\dots\dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -s_ns_1 & -s_ns_2 & -s_ns_3 & \dots\dots & \dots\dots & s_n(1-sn) \end{bmatrix}$$

# Softmax Function in Back-Propagation

**Loss Function**: The cross-entropy loss for multi-class problem is given by:

$$L = -\sum_{i=1}^{n} y_i \log(s_i)$$

Where $y=[y1,y2,\ldots,yn]^{\top}$ is the one-hot encoded true label vector and si denotes the softmax output

$$\frac{\partial L}{\partial s_i} = -y_i * \frac{1}{s_i}$$

In matrix form, $\frac{\partial L}{\partial} = -Y * \frac{1}{S}$

Where * represents element-wise division.

# Softmax Function in Back-Propagation (contd…)

Using the chain rule,

$$\frac{\partial L}{\partial z} = \frac{\partial s}{\partial z}\frac{\partial L}{\partial s}\ , In\ order\ to\ get\ output\ of\ order\ \#classes, \#examples$$

$$= \begin{bmatrix} s_1(1-s_1) & -s_1s_2 & -s_1s_3 & \dots\dots & \dots\dots & -s_1s_n \\ -s_2s_1 & s_2(1-s_2) & -s_2s_3 & \dots\dots & \dots\dots & -s_2s_n \\ -s_3s_1 & -s_3s_2 & s_3(1-s_3) & \dots\dots & \dots\dots & -s_3s_n \\ \vdots & \vdots & \vdots & \dots\dots & \dots\dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -s_ns_1 & -s_ns_2 & -s_ns_3 & \dots\dots & \dots\dots & s_n(1-sn) \end{bmatrix} . -y * \frac{1}{s}$$

$-y * \frac{1}{s}$ *only one non zero entry will be there as y is one hot vector. Let pthentry be one. So pth column will be multiplied only*

$$= \begin{bmatrix} s_1(1-s_1) & -s_1s_2 & -s_1s_3 & \dots\dots & \dots\dots & -s_1s_n \\ -s_2s_1 & s_2(1-s_2) & -s_2s_3 & \dots\dots & \dots\dots & -s_2s_n \\ -s_3s_1 & -s_3s_2 & s_3(1-s_3) & \dots\dots & \dots\dots & -s_3s_n \\ \vdots & \vdots & \vdots & \dots\dots & \dots\dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -sns_1 & -sns_2 & -sns_3 & \dots\dots & \dots\dots & sn(1-sn) \end{bmatrix} . \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \frac{-1}{s_p} \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_p - 1 \\ \vdots \\ s_n \end{bmatrix}$$

$Therefore, dZ = S - Y\ or\ A(z_i) - Y$ for softmax function