NONLINEAR ACTIVATION FUNCTIONS

Dr. Umarani Jayaraman

Non Linear Functions

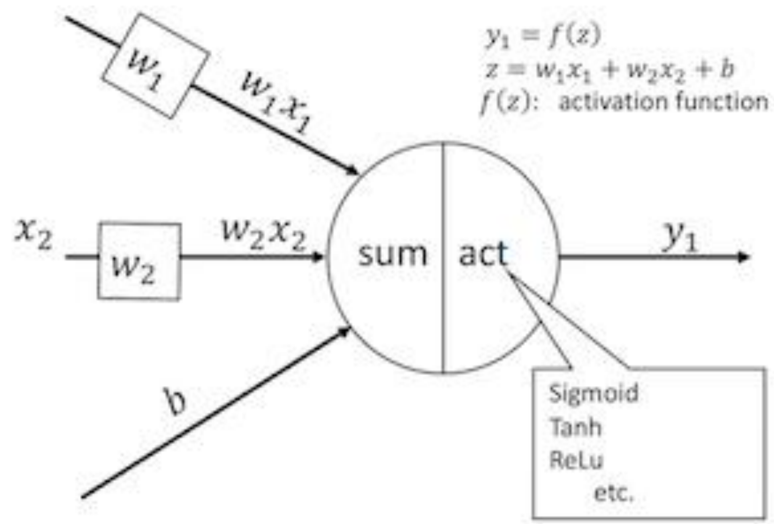
Activation Functions

Why activation functions?

- Activation functions allow for non-linearity in the fundamentally linear model, which is nothing but a sequence of linear operations.
- Activation functions are used to determine the firing of neurons in a neural network.
- A neural network without an activation function is essentially just a linear model
- The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Why activation function?

- Every neuron has two operations
  - Summation: linear combination of input X with W
  - Non linear Activation function *f*: The purpose of the activation function is to **introduce non-linearity** into the output of a neuron.



Characteristics of activation functions

- An ideal activation function is both nonlinear and differentiable.
- The nonlinear behavior of an activation function allows our neural network to learn nonlinear relationships in the data.
- It should be continuous, differentiable, non-decreasing, and easy to compute.
- Differentiability is important because it allows us to back propagate the model's error when training to optimize the weights.

Step/Threshold Function

- While this is the original activation first developed when neural networks were invented, it is no longer used in neural network architectures because it's incompatible with backpropagation.
- Backpropagation allows us to find the optimal weights for our model using a version of gradient descent;
- Unfortunately, the derivative of a step activation function cannot be used to update the weights (since it is 0).

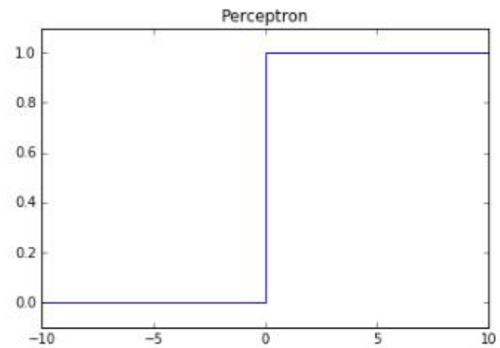
Step/Threshold Function

Function:

$$f(x) = \begin{Bmatrix} 0 & for & x < 0 \\ 1 & for & x \geq 0 \end{Bmatrix}$$

Derivative:

$$f'(x) = \begin{Bmatrix} 0 & for & x \neq 0 \\ ? & for & x = 0 \end{Bmatrix}$$



- **Problems:** not compatible gradient descent via backpropagation since its derivative is zero

Sigmoid (logistic)

- The sigmoid function is commonly used non-linear function
- However, it has fallen out of practice to use this activation function in real-world neural networks due to a problem known as the **vanishing gradient.**

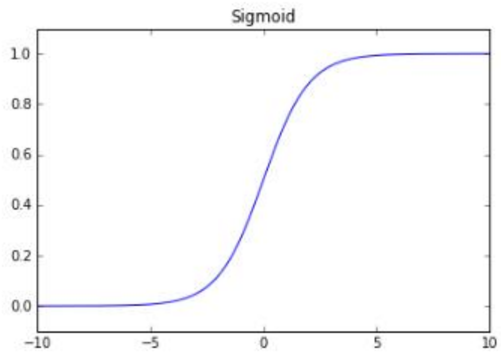
Sigmoid (logistic)

Function:

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

Derivative:

$$f'(x) = f(x)(1 - f(x))$$

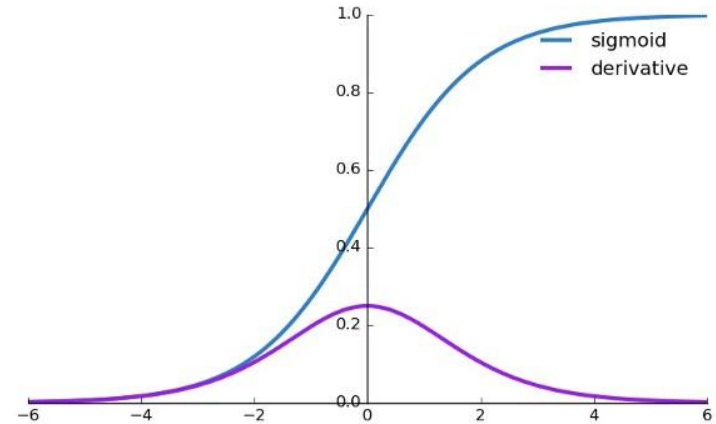


- Problems: vanishing gradient at edges
- Output is not zero centered.

Sigmoid (logistic)

- The sigmoid function has values between 0 to 1
- The output is not Zero-Centered
- Sigmoid saturate and kill gradients.
- We could see at top and bottom level of sigmoid functions the curve changes slowly, if we calculate slope(gradients) it is zero
- Due to this when the x value is small or big the slope is zero

  → then there is no learning



Sigmoid (logistic)

- **When we will use Sigmoid:**
  - If we want output value between 0 to 1 use sigmoid at output layer neuron only
  - For binary classification problem sigmoid is used
  - Otherwise sigmoid is not preferred

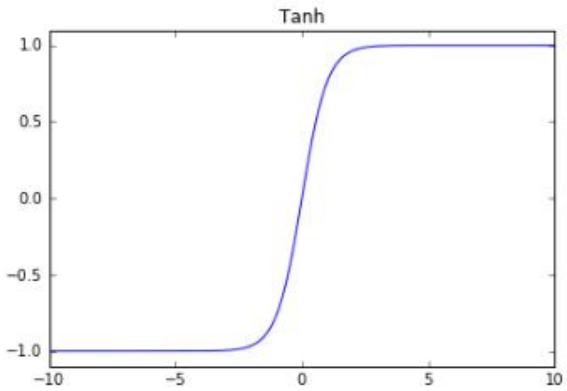
Hyperbolic Tangent

Function:

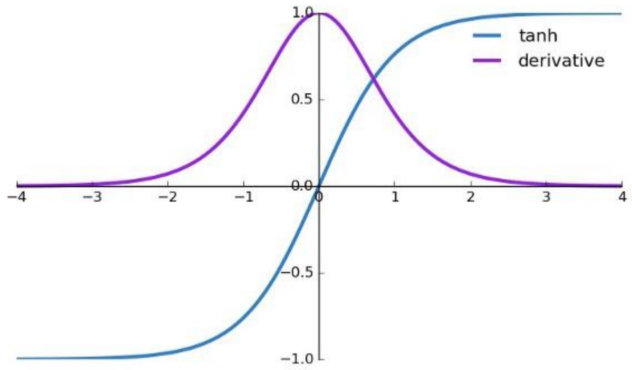
$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Derivative:

$$f'(x) = 1 - f(x)^2$$

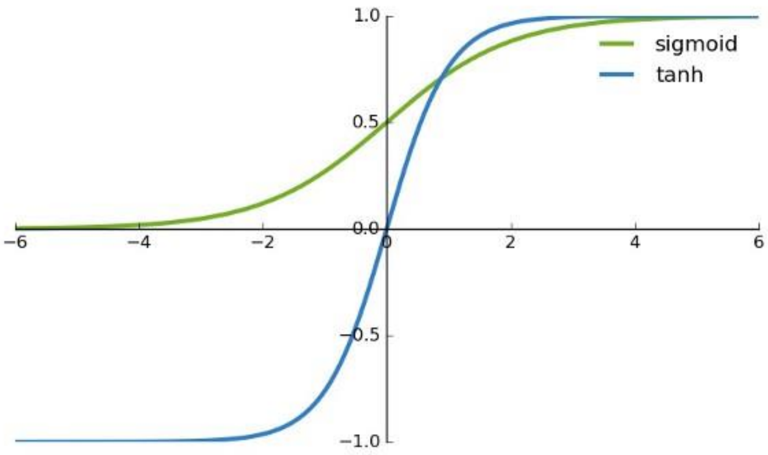


- Problems: vanishing gradient at edges.



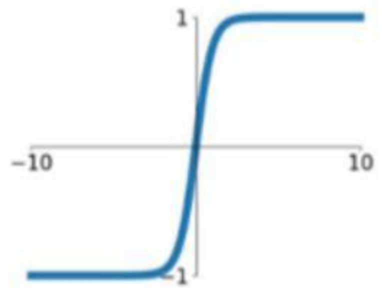
Hyperbolic Tangent

- Now it's output is zero centered because its range in between -1 to 1 i.e -1 < output < 1 .
- Hence optimization is *easier* in this method hence in practice it is always preferred over Sigmoid function .
- **But still it suffers from Vanishing gradient problem.**

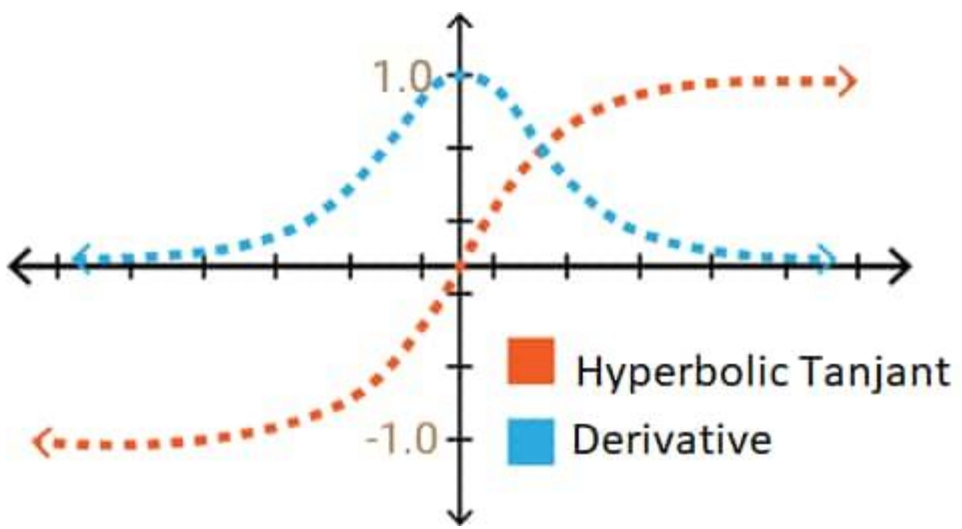


Hyperbolic Tangent

Function	Equation	Range	Derivative
Tanh (Hyperbolic tangent)	$f(x) = \frac{2}{1+e^{-2x}} - 1$	-1,1	$f'(x) = 1 - f(x)^2$



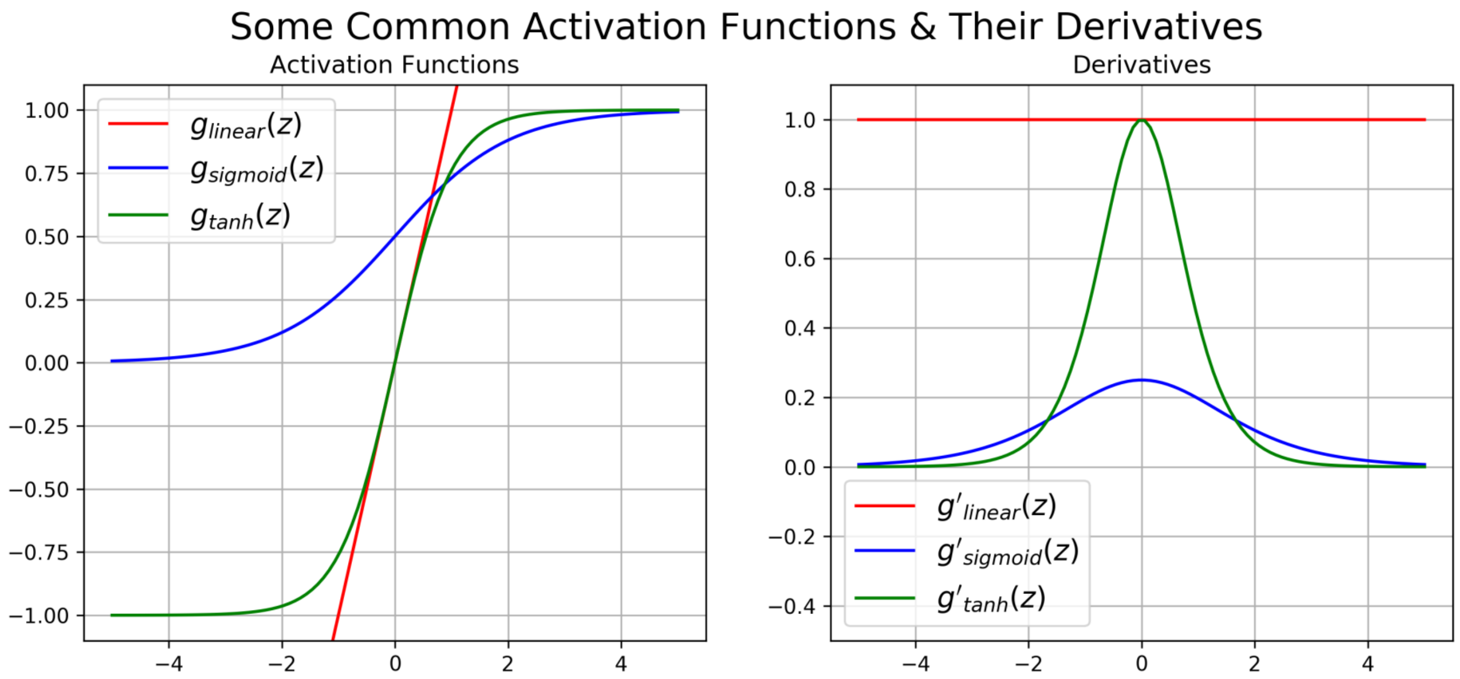
$$tanh(x) \; = \; 2 \, sigmoid(2x) \; - \; 1$$



Hyperbolic Tangent

- **When will use/ what is the use of zero centerted:**
- 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.

Activation functions- sigmoid, tanh and linear and its derivatives



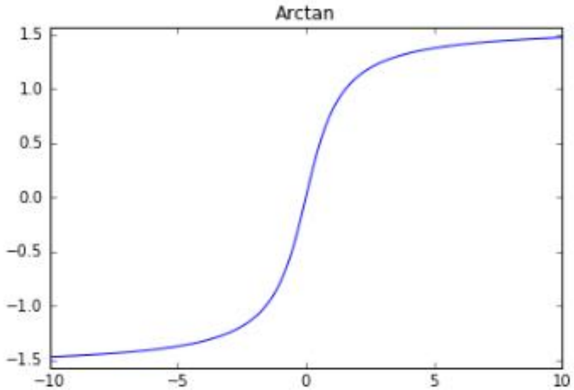
Inverse Tangent

Function:

$$f(x) = \tan^{-1}(x)$$

Derivative:

$$f'(x) = \frac{1}{x^2 + 1}$$



- Output is zero centered.

ReLU (Rectified Linear Unit)

Function:

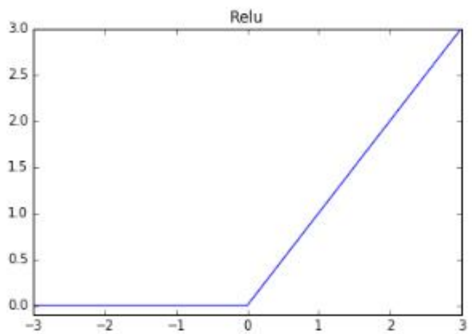
$$f(x) = \begin{Bmatrix} 0 & for & x < 0 \\ x & for & x \geq 0 \end{Bmatrix}$$

or (another way to write the ReLU function is...)

$$f(x) = \max(x, 0)$$

Derivative:

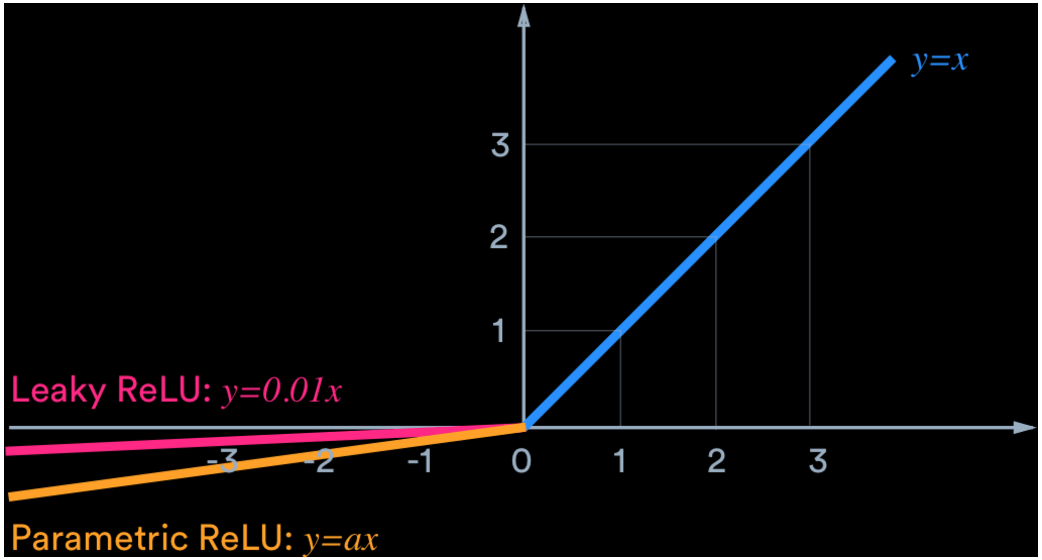
$$f'(x) = \begin{Bmatrix} 0 & for & x < 0 \\ 1 & for & x \geq 0 \end{Bmatrix}$$



- This is one of the most popularly used activation functions since 2017.
- It avoids and rectifies **vanishing gradient** problem . Almost all deep learning Models use **ReLu** nowadays.
- ReLu could result in Dead Neurons

ReLU Variants

- Due to its popularity, a number of variants have been proposed that provide an incremental benefit over standard ReLUs
  - Leaky ReLU, Parametric ReLU
  - Maxout, ELU



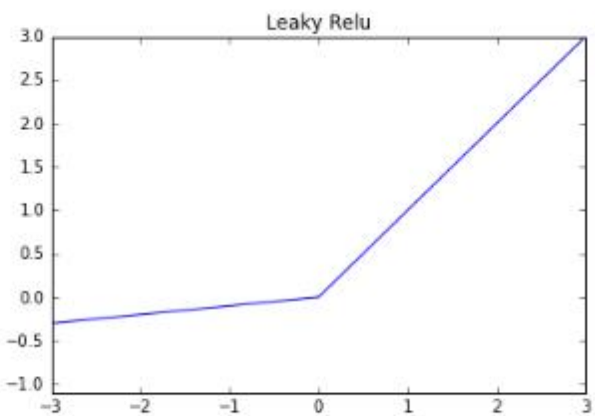
ReLU Variants- Leaky ReLU

Function:

$$f(x) = \begin{cases} 0.1x & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$$

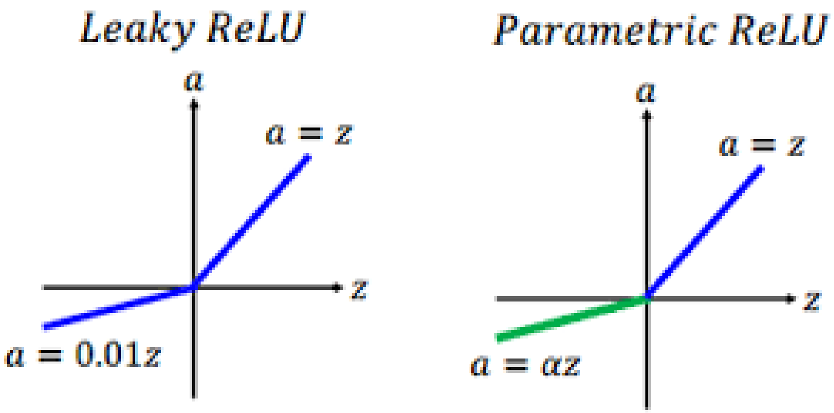
Derivative:

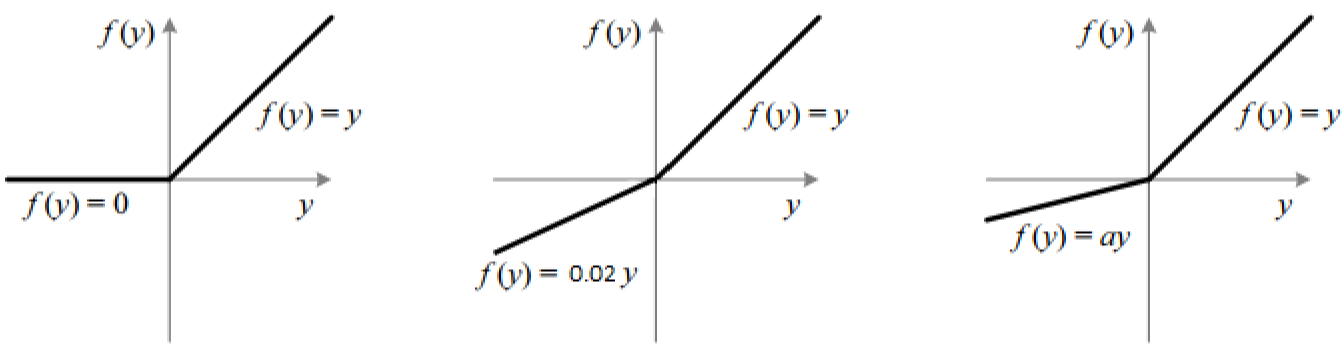
$$f'(x) = \begin{cases} 0.1 & for \quad x < 0 \\ 1 & for \quad x \geq 0 \end{cases}$$



- ***Leaky ReLu*** fix the problem of dead neurons that occurred in ReLU
- It introduces a small slope to keep the updates alive
- But its limitation is that it should only be used within Hidden layers of a Neural Network Model.

ReLU variants- Parametric ReLU





ReLU Variants- Maxout Function

- We then have another variant made form both ReLu and Leaky ReLu called **Maxout** function .

Function:

$$f(\vec{x}) = \max_i x_i$$

Derivative:

$$\frac{\partial f}{\partial x_j} = \begin{Bmatrix} 1 & for & j = \arg\max_i x_i \\ 0 & for & j \neq \arg\max_i x_i \end{Bmatrix}$$

ELU (Exponential Linear Units)

- No dead neurons
- Output is zero centered.

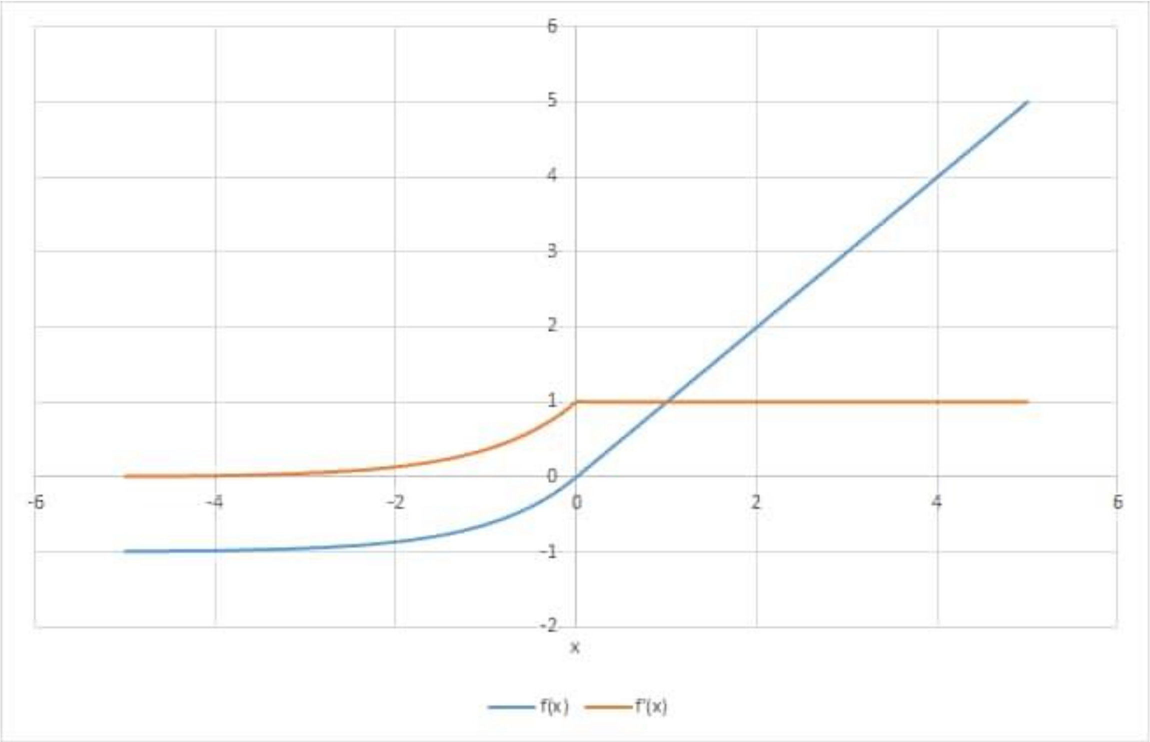
Function:

$$f(\alpha, x) = \lambda \begin{cases} \alpha\left(e^{x} - 1\right) & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$$

Derivative:

$$f'(\alpha, x) = \lambda \begin{cases} f(\alpha, x) + \alpha & for \quad x < 0 \\ 1 & for \quad x \geq 0 \end{cases}$$

ReLU Variants- ELU (Exponential Linear Units)



Identity function – for output layer

- The following activation functions should only be used on the output layer
- Use: Regression

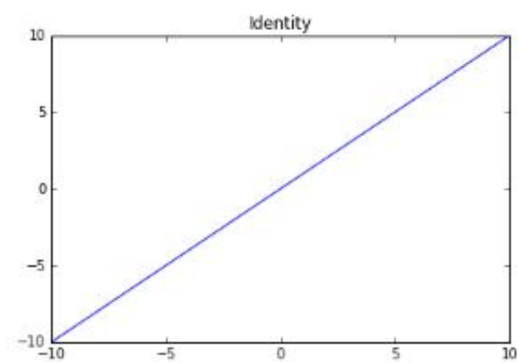
Identity

Function:

$$f(x) = x$$

Derivative:

$$f'(x) = 1$$



Softmax- for output layer

- The softmax function is commonly used as the output activation function for multi-class
- It scales the preceding inputs from a range between 0 and 1 and normalizes the output layer so that the sum of all output neurons is equal to one.
- As a result, we can consider the softmax function as a categorical probability distribution.
- This allows you to communicate a degree of confidence in your class predictions.

Softmax- for output layer

Function:

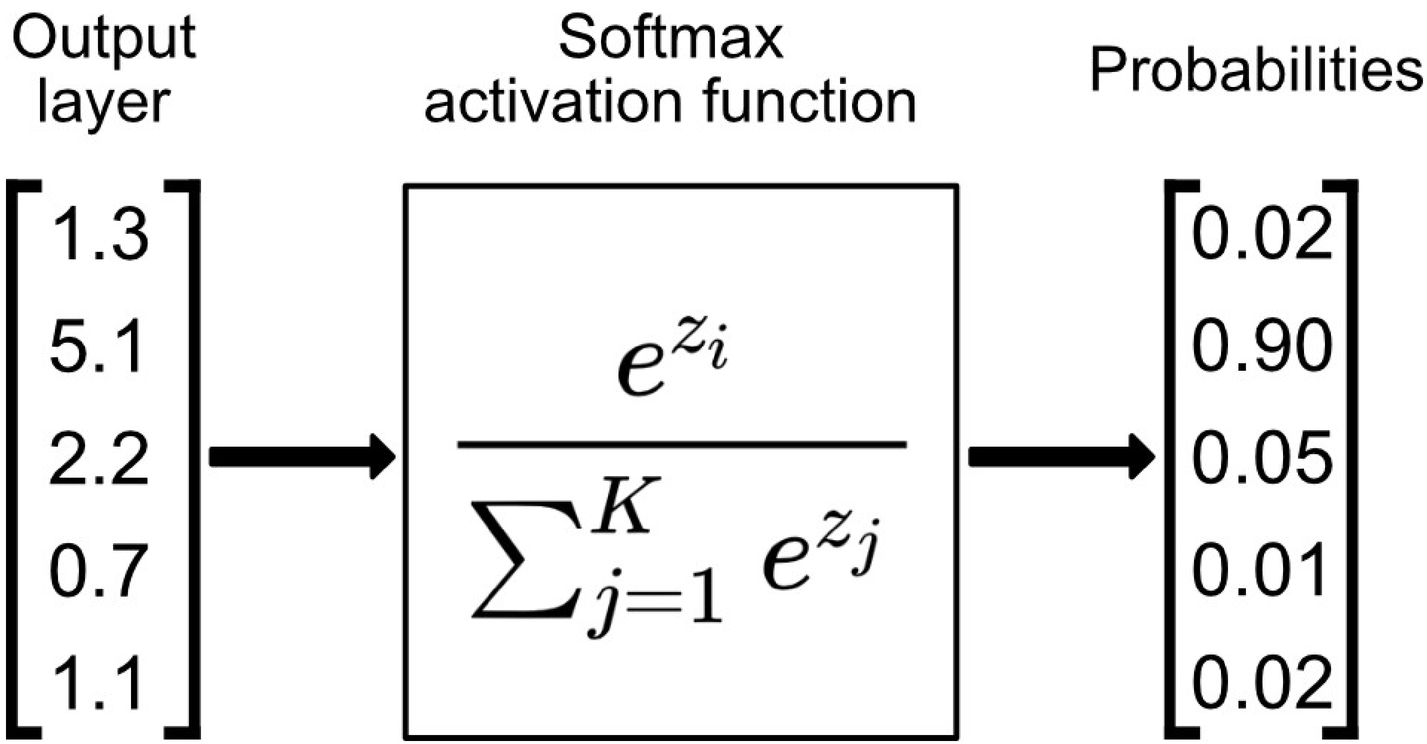
$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad \text{for i} = 1, \ldots, \text{J}$$

Derivative:

$$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})\left(\delta_{ij} - f_i(\vec{x})\right)$$

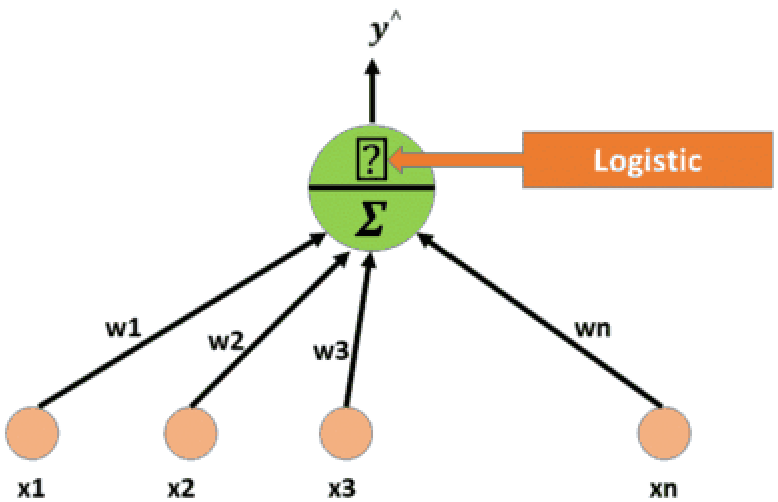
- Note: we use the exponential function to ensure all values in the summation are positive.
- Use: classification.

Softmax/ Normalized Exponential Function

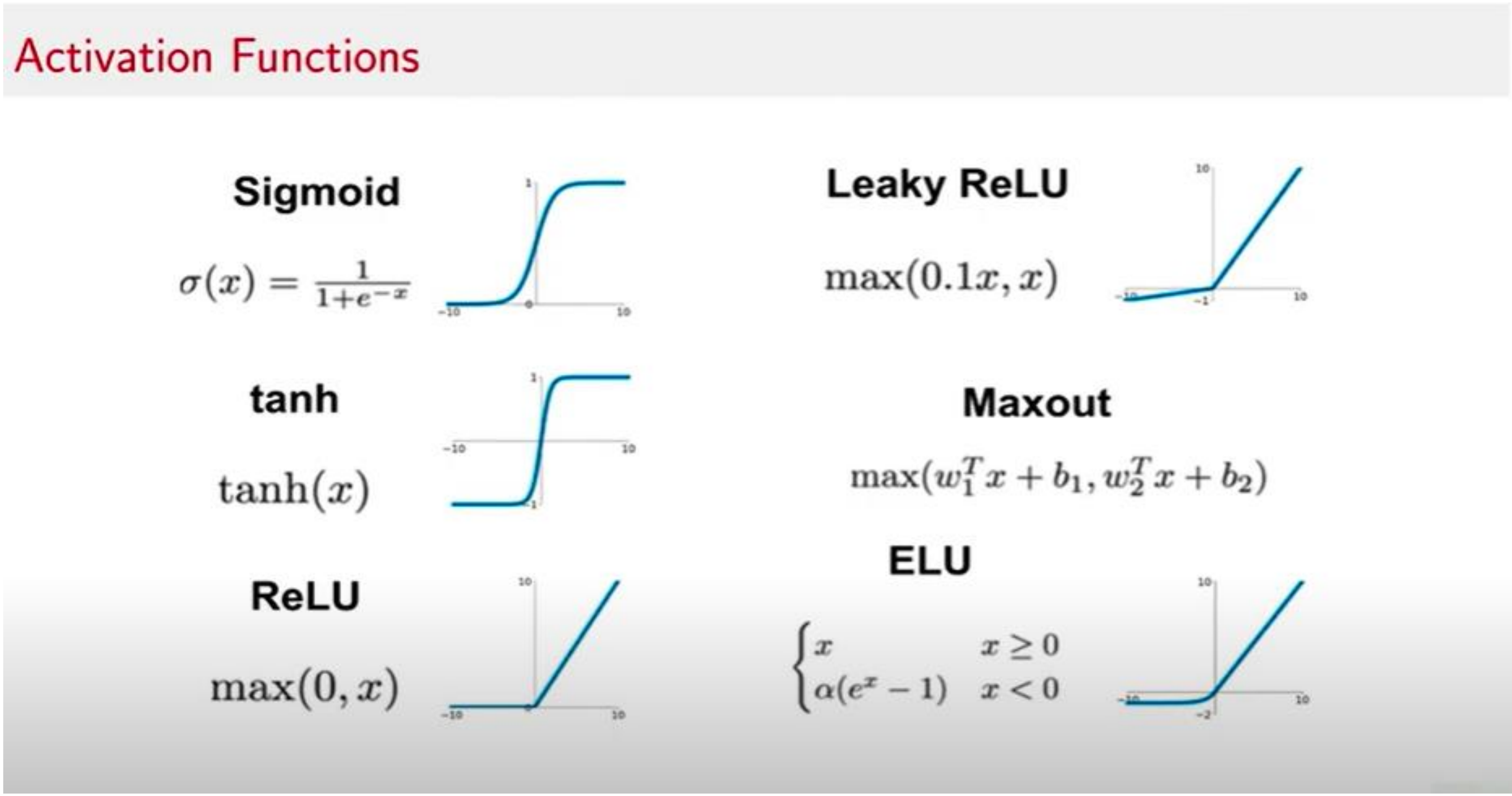


Summary

- Activation Functions



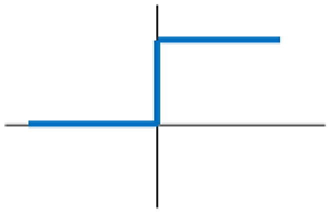
Summary


Activation Functions
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
tanh
$\tanh(x)$
ReLU
$\max(0, x)$
Leaky ReLU
$\max(0.1x, x)$
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

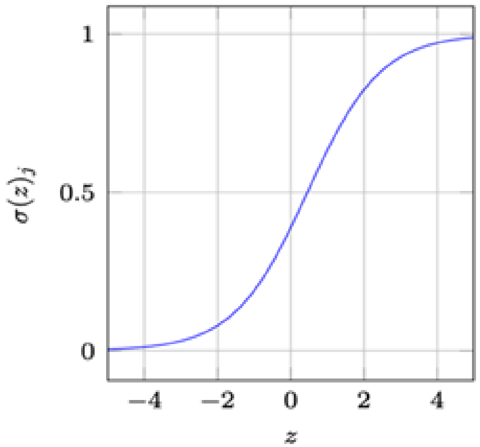
Summary

- Step Function

Function	Equation	Range	Derivative
Binary step	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$	$-\infty, +\infty$	$f'(x) = 0$



- Softmax



(b) Softmax activation function.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Summary

Activation Functions: Which one to choose?

- Use ReLU non-linearity, be careful with learning rates and monitor the fraction of 'dead' units in a network.
- Try Leaky ReLU, ELU, Maxout
- Try tanh, but expect worse performance
- Sigmoid not used much, unless a gating is required
- In general, a good idea for an activation function to be in its linear region for most part of training

Summary

Function Type	*Equation*	*Derivative*
Linear	$f(x) = ax + c$	$f'(x) = a$
Sigmoid	$f(x) = \frac{1}{1+ e^{-x}}$	$f'(x) = f(x)\,(1 - f(x))$
TanH	$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \begin{cases} 0 \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
Parametric ReLU	$f(x) = \begin{cases} \alpha x \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
ELU	$f(x) = \begin{cases} \alpha(e^{x} - 1) \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$

Sources:

- https://www.jeremyjordan.me/neural-networks-activation-functions/
- https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/
- https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
- https://github.com/MlvPrasadOfficial/ineuron.ai/blob/master/IPYNB%20FILES%20DL/Activation%20Functions.ipynb

Sorces-Sigmoid function

- https://deepai.org/machine-learning-glossary-and-terms/sigmoidal-nonlinearity
- https://www.analyticsvidhya.com/blog/2020/12/beginners-take-how-logistic-regression-is-related-to-linear-regression/
- https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Thank you