NEURAL NETWORKS AND DEEP LEARNING CST 395 CS 5TH SEMESTER HONORS COURSE- Dr Binu V P, 9847390760

CS 5th Semester Honors course for the Computer Science at KTU- Dr Binu V P

Activation Functions

August 19, 2022

Activation Functions

The activation function defines the output of a neuron / node given an input or set of input (output of multiple neurons). It's the mimic of the stimulation of a biological neuron.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.

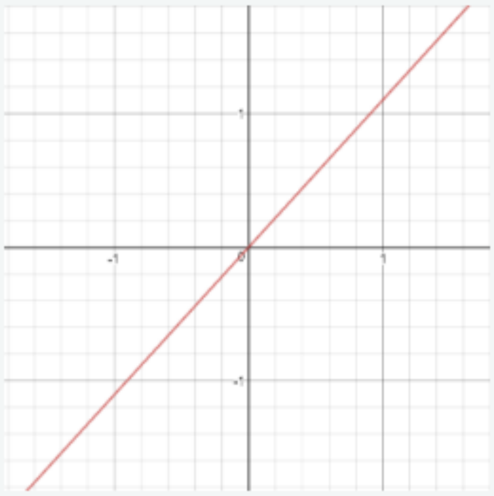
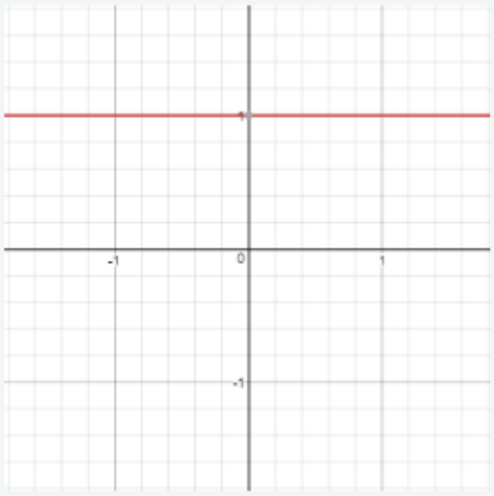
We know, neural network has neurons that work in correspondence of weight, bias and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as back-propagation. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

The Activation Functions can be basically divided into 2 types-

- Linear Activation Function
- Non-linear Activation Function

Linear Activation Functions

- Equation : Linear function has the equation similar to as of a straight line i.e. $y = ax$
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- Range : -inf to +inf
- Uses : Linear activation function is used at just one place i.e. output layer.
- Issues : If we will differentiate linear function to bring non-linearity, result will no more depend on input “x” and function will become constant, it won’t introduce any ground-breaking behavior to our algorithm.

Function	Derivative
$R(z,m) = \{ z * m \}$	$R'(z,m) = \{ m \}$
	
`def linear(z,m): return m*z`	`def linear_prime(z,m): return m`

```python
def linear(z,m):
        return m*z
```

```python
def linear_prime(z,m):
        return m
```

Pros

- It gives a range of activations, so it is not binary activation.
- We can definitely connect a few neurons together and if more than 1 fires, we could take the max (or softmax) and decide based on that.

Cons

- For this function, derivative is a constant. That means, the gradient has no relationship with X.
- It is a constant gradient and the descent is going to be on constant gradient.
- If there is an error in prediction, the changes made by back propagation is constant and not depending on the change in input delta(x) !

For example : Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.

Non-linear Activation Function

The Nonlinear Activation Functions are the most used activation functions.It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.

The main terminologies needed to understand for nonlinear functions are:

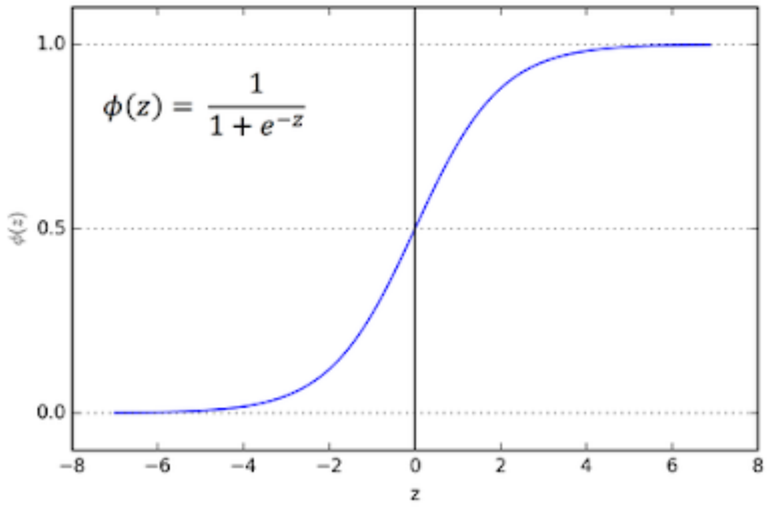
Derivative or Differential: Change in y-axis w.r.t. change in x-axis.It is also known as slope.

Monotonic function: A function which is either entirely non-increasing or non-decreasing.

The Nonlinear Activation Functions are mainly divided on the basis of their range or curves

1. Sigmoid or Logistic Activation Function

The Sigmoid Function curve looks like a S-shape.



The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

The function is differentiable.That means, we can find the slope of the sigmoid curve at any two points.

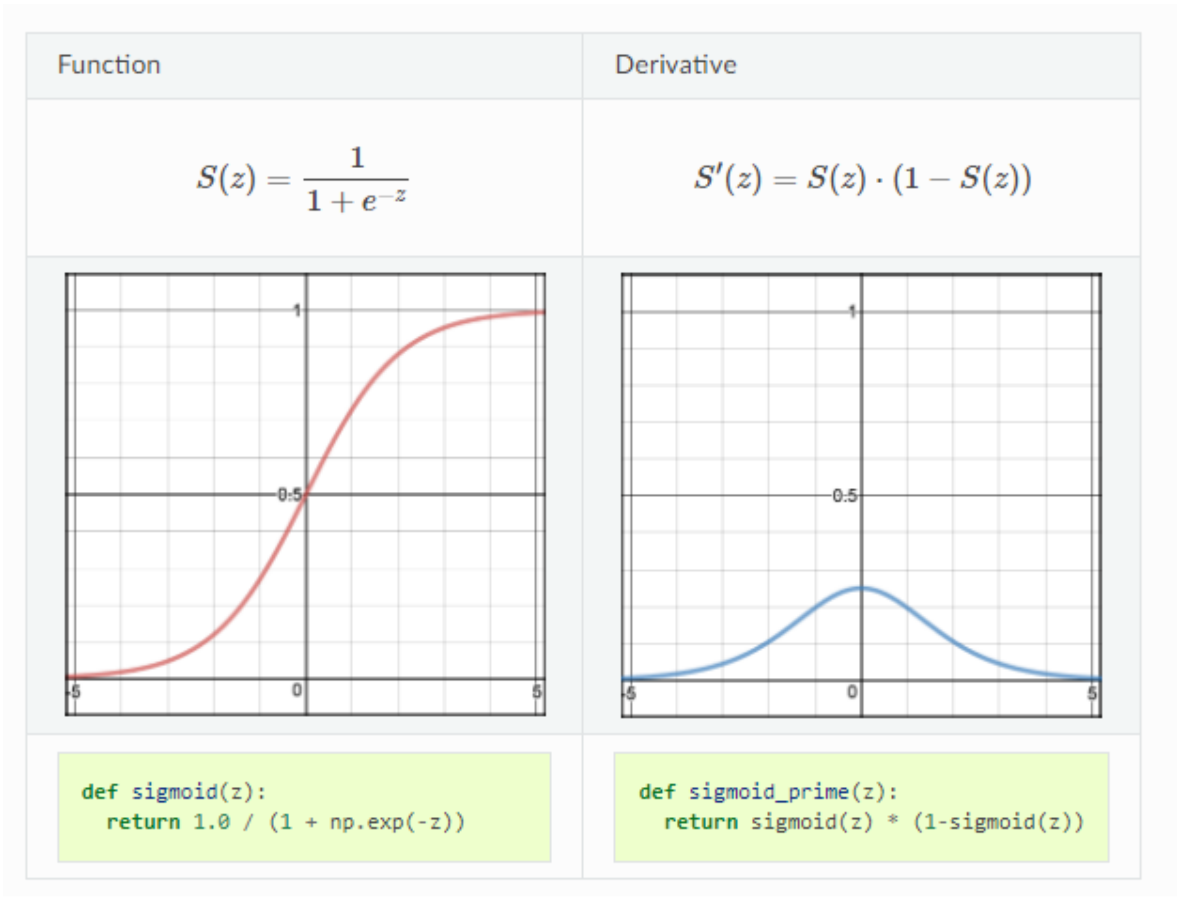
The function is monotonic but function’s derivative is not.

The logistic sigmoid function can cause a neural network to get stuck at the training time.

- Equation :

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

- Value Range : 0 to 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.



Function	Derivative
$S(z) = \frac{1}{1+e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$

```
def sigmoid(z):
  return 1.0 / (1 + np.exp(-z))
```

```
def sigmoid_prime(z):
  return sigmoid(z) * (1-sigmoid(z))
```

Pros

- It is nonlinear in nature. Combinations of this function are also nonlinear!
- It will give an analog activation unlike step function.
- It has a smooth gradient too.
- It's good for a classifier.
- The output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function. So we have our activations bound in a range. Nice, it won't blow up the activations then.

Cons

- Towards either end of the sigmoid function, the Y values tend to respond very less to changes in X.
- It gives rise to a problem of "vanishing gradients".
- Its output isn't zero centered. It makes the gradient updates go too far in different

directions. 0 < output < 1, and it makes optimization harder.
- Sigmoids saturate and kill gradients.
- The network refuses to learn further or is drastically slow (depending on use case and until gradient /computation gets hit by floating point value limits).
- The softmax function is a more generalized logistic activation function which is used for multiclass classification.

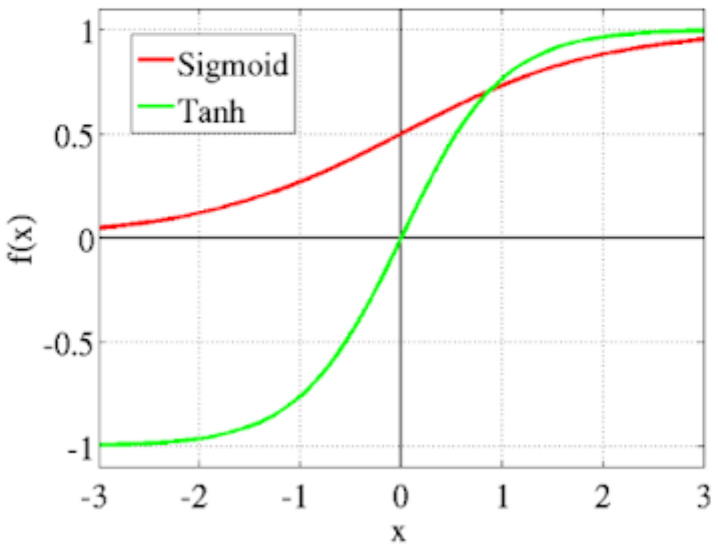
2. Softmax Function :- The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems.
Nature :- non-linear
Uses :- Usually used when trying to handle multiple classes. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
Output:- The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

Mathematically, Softmax is defined as,
$S(y)_i = \frac{exp(y_i)}{\sum_{j=1}^{n} exp(y_j)}$
n- is the number of classes (possible outcomes)
y_i is the input
$exp(y_i)$ is the standard exponential function applied on y_i, the result is a smaller value closer to zero when $y_i < 0$ and large value when $y_i > 0$

3. Tanh or Hyperbolic Tangent Activation Function

tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).Tanh function also knows as Tangent Hyperbolic function. It’s actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
Tanh squashes a real-valued number to the range [-1, 1]. It’s non-linear. But unlike Sigmoid, its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity.



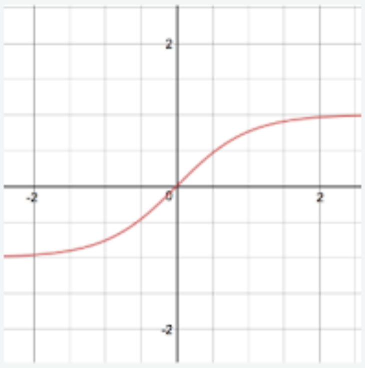
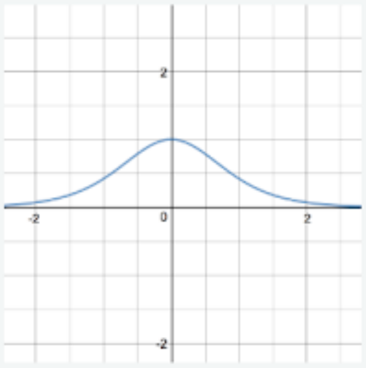
The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

The function is differentiable.

The function is monotonic while its derivative is not monotonic.

The tanh function is mainly used classification between two classes.

Both tanh and logistic sigmoid activation functions are used in feed-forward nets.
Equation :-
$f(x) = tanh(x) = \frac{2}{(1+e^{-2x})} - 1$ OR $tanh(x) = 2 * sigmoid(2x) - 1$
Value Range : -1 to +1
Nature :- non-linear
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.

Function	Derivative
$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$tanh'(z) = 1 - tanh(z)^2$
	

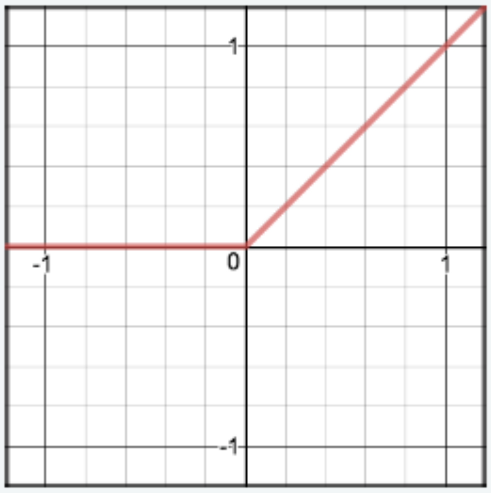
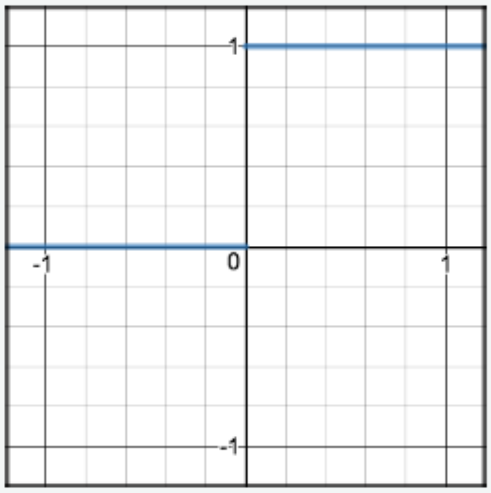
```
def tanh(z):
        return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))
```

```
def tanh_prime(z):
        return 1 - np.power(tanh(z), 2)
```

Pros
The gradient is stronger for tanh than sigmoid (derivatives are steeper).
Cons
Tanh also has the vanishing gradient problem.

4. ReLU (Rectified Linear Unit) Activation Function

A recent invention which stands for **Rectified Linear Units.** The formula is deceptively simple: $max(0, z)$ Despite its name and appearance, it's not linear and provides the same benefits as Sigmoid (i.e. the ability to learn nonlinear functions), but with better performance. it is used in almost all the convolutional neural networks or deep learning

Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix}$
	

`def relu(z):`
`  return max(0, z)`
	`def relu_prime(z):`
	`  return 1 if z > 0 else 0`

As you can see, the ReLU is half rectified (from bottom). $R(z)$ is zero when z is less than zero and $R(z)$ is equal to z when z is above or equal to zero.

Range: [0 to infinity)

The function and its derivative both are monotonic.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

Equation :- $A(x) = max(0, x)$. It gives an output x if x is positive and 0 otherwise.
Value Range :- $[0, \inf)$
Nature :- non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
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.

In simple words, ReLU learns much faster than sigmoid and Tanh function.

Pros

- It avoids and rectifies vanishing gradient problem.
- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.

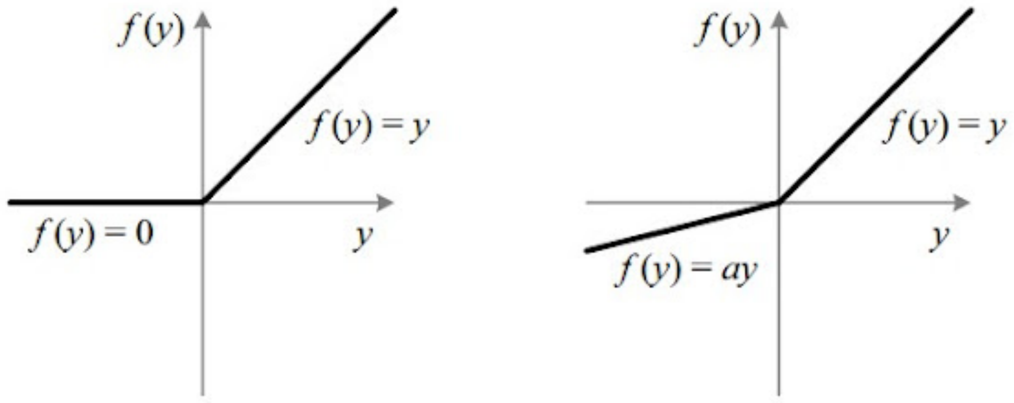
Cons

- One of its limitations is that it should only be used within hidden layers of a neural network model.
- Some gradients can be fragile during training and can die. It can cause a weight update which will makes it never activate on any data point again. In other words, ReLu can result in dead neurons.
- In another words, For activations in the region $(x < 0)$ of ReLu, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes). This is called the dying ReLu problem.

The range of ReLu is [0,∞)[0,∞). This means it can blow up the activation.

5. Leaky ReLU

LeakyRelu is a variant of ReLU. Instead of being 0 when $z < 0$, a leaky ReLU allows a small, non-zero, constant gradient α (Normally, $\alpha = 0.01$).
It is an attempt to solve the dying ReLU problem

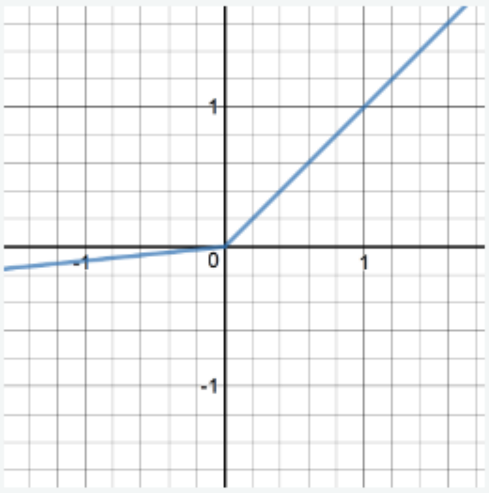
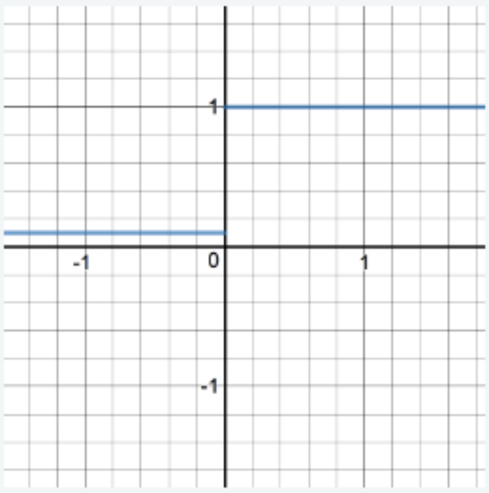


The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.

When a is not 0.01 then it is called Randomized ReLU.

Therefore the range of the Leaky ReLU is (-infinity to infinity).

Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives also monotonic in nature.

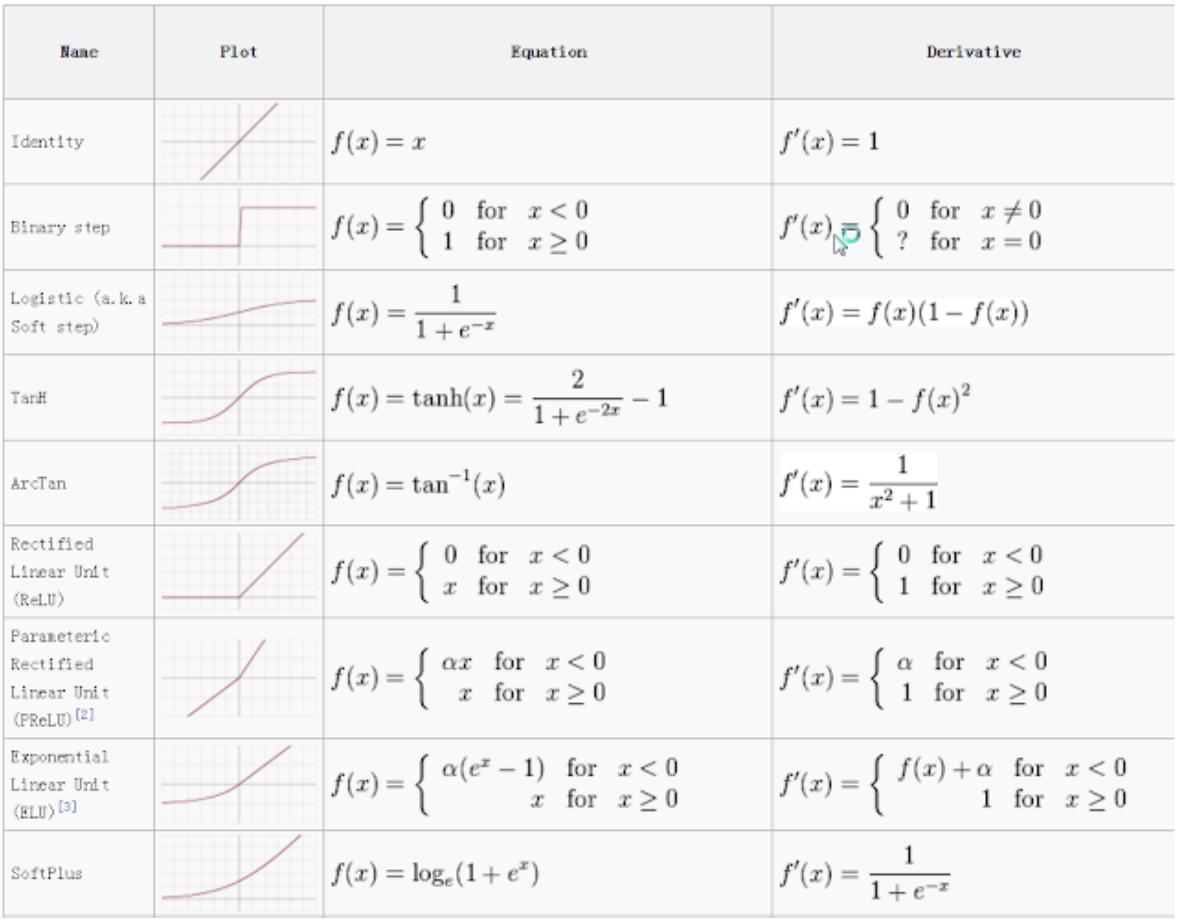
Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha z & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ \alpha & z < 0 \end{Bmatrix}$
	

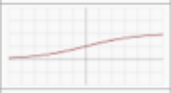
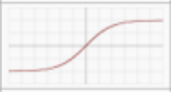
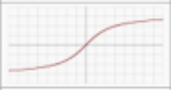
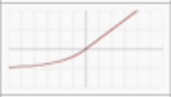
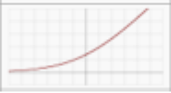
`def leakyrelu(z, alpha):`
`        return max(alpha * z, z)`
	`def leakyrelu_prime(z, alpha):`
	`        return 1 if z > 0 else alpha`

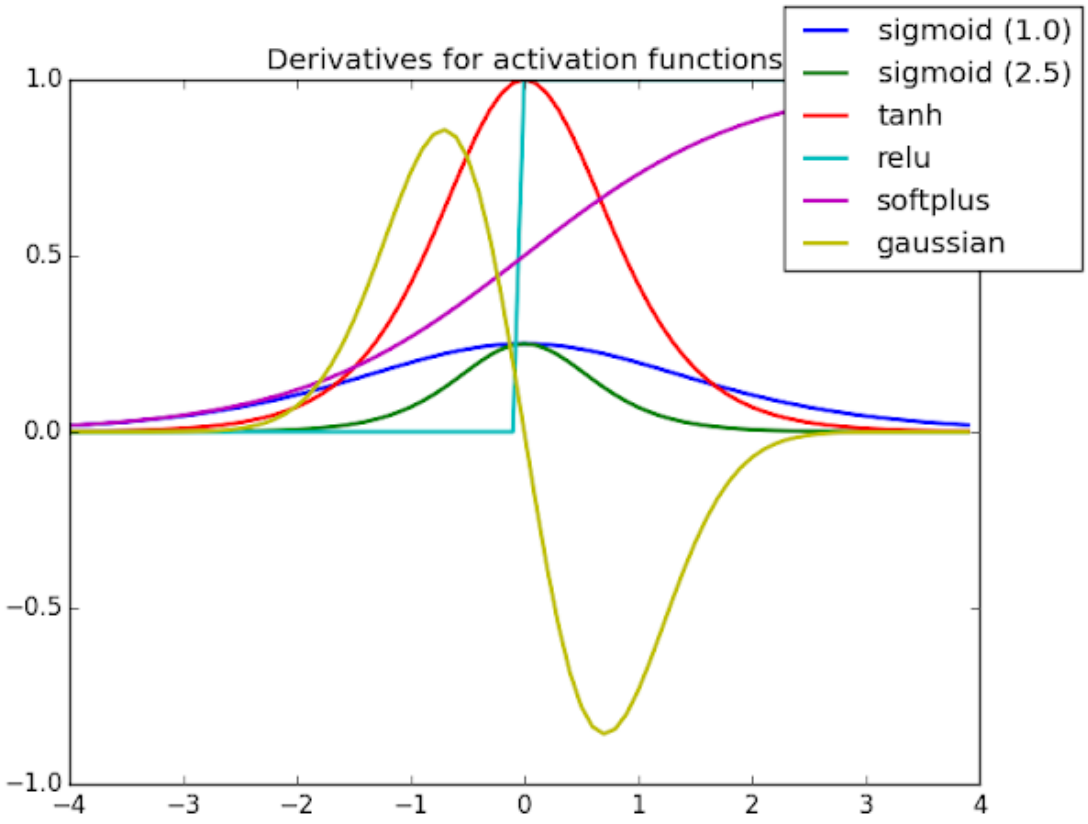
Pros
Leaky ReLUs are one attempt to fix the “dying ReLU” problem by having a small negative slope (of 0.01, or so).

Cons
As it possess linearity, it can’t be used for the complex Classification. It lags behind the Sigmoid and Tanh for some of the use cases.

Summary of Activation Functions



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$



Remarks

The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.
The basic rule of thumb is if you really don’t know what activation function to use, then simply use ReLU as it is a general activation function and is used in most cases these days.
If your output is for binary classification then, sigmoid function is very natural choice for output layer.

Advantages of ReLU

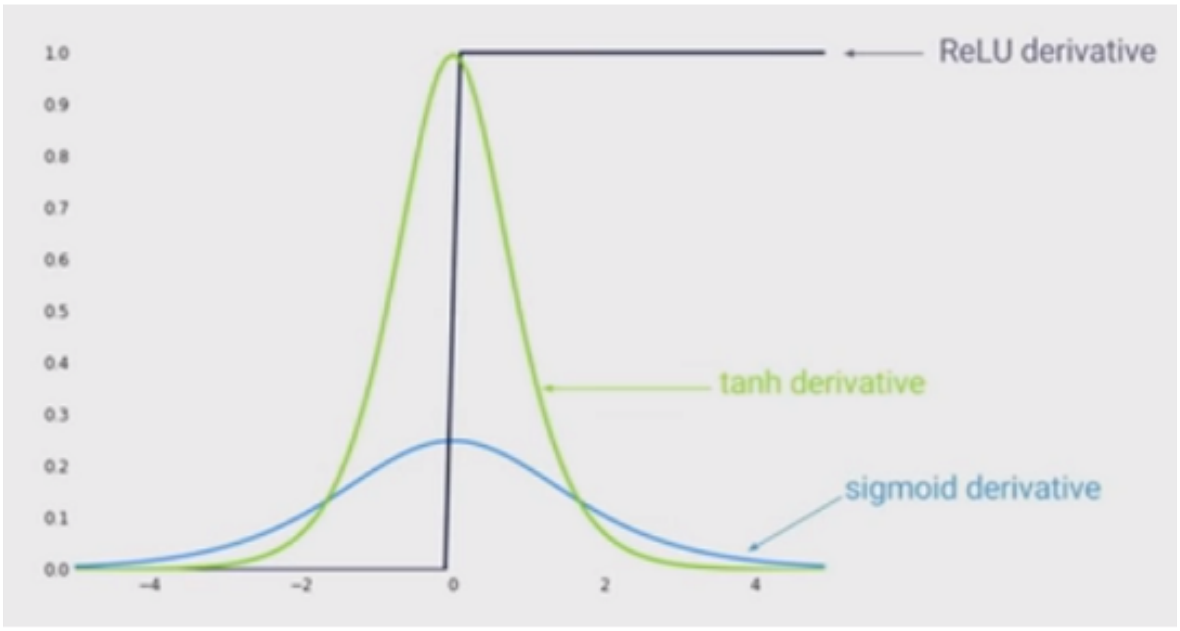
It was ReLU (among other things, admittedly) that facilitated the training of deeper nets. And ever since we have been using ReLU as a default activation function for the hidden layers. So exactly what makes ReLU a better choice over Sigmoid?

Computational Speed

ReLUs are much simpler computationally. The forward and backward passes through ReLU are both just a simple "if" statement.
Sigmoid activation, in comparison, requires computing an exponent.
This advantage is huge when dealing with big networks with many neurons, and can significantly reduce both training and evaluation times.
The training times reduced to half of that of using sigmoid.

Vanishing Gradient

Additionally, sigmoid activations are easier to saturate. There is a comparatively narrow interval of inputs for which the Sigmoid's derivative is sufficiently nonzero. In other words, once a sigmoid reaches either the left or right plateau, it is almost meaningless to make a backward pass through it, since the derivative is very close to 0.
On the other hand, ReLU only saturates when the input is less than 0. And even this saturation can be eliminated by using leaky ReLUs. For very deep networks, saturation hampers learning, and so ReLU provides a nice workaround.



Convergence Speed

With a standard Sigmoid activation, the gradient of the Sigmoid is typically some fraction between 0 and 1. If you have many layers, they multiply, and might give an overall gradient that is exponentially small, so each step of gradient descent will make only a tiny change to the weights, leading to slow convergence (the vanishing gradient problem).

In contrast, with ReLu activation, the gradient of the ReLu is either 0 or 1. That means that often, after many layers, the gradient will include the product of a bunch of 1's and the overall gradient won't be too small or too large.

ReLU provides Better performance and faster convergence.

October 03, 2022

About Me Syllabus Question Paper Dec 2022 Module 1 (Basics of Machine Learning) Overview of Machine Learning Machine Learning Algorithm Linear Regression Capacity, Overfitting and Underfitting Regularization Hyperparameters and Validatio …

READ MORE

Machine Learning Algorithm

October 01, 2022

A machine learning algorithm is an algorithm that is able to learn from data. But what do we mean by learning? Mitchell (1997) provides the definition "A computer program is said to learn from experience E with respect to some class of tasks T and …

READ MORE

Syllabus CST 395 Neural Network and Deep Learning

October 01, 2022

Syllabus Module - 1 (Basics of Machine Learning) Machine Learning basics - Learning algorithms - Supervised, Unsupervised, Reinforcement, overfitting, Underfitting, Hyper parameters and Validation sets, Estimators -Bias and Variance. Challenge …

READ MORE