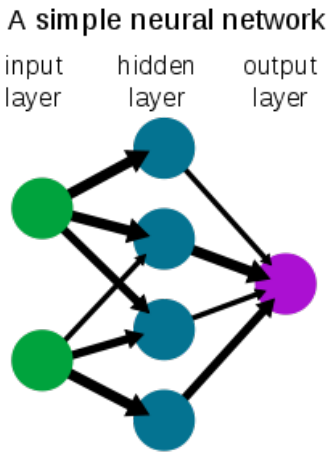
Activation Functions Explained

Step, Sigmoid, Hyperbolic Tangent, Softmax, ReLU, Leaky ReLU Explained

Before diving deep into activation functions, we need to understand how neural networks works.

A neural network is a complex network of artificial neurons that imitates how the brain works. They accept parameters with its associated weights and biases, then we compute the weighted sum of the 'activated' neurons. Our activation function is a decision maker that help to assist which neurons will push forward the values into the next layer.



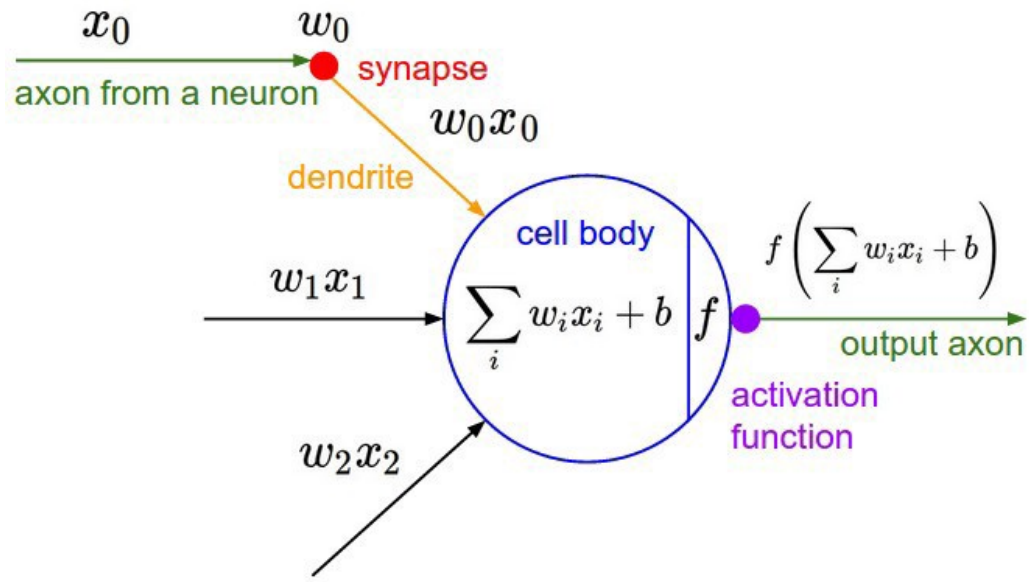
Basic Neural Network, Source

In the forward propagation, our artificial neurons receive inputs from different features. Ideally, each input has its features that is weight and biases which can display dependence that leads to change in the final predicted value. This is a phenomenon referred to as the **Interaction effect**.

A good example would be when we employ regression model on house price prediction dataset. The goal is to predict the price of house based on different features. Some of features might include locality and sq ft area of house. There is some interdependency between area of house and locality i.e. more populated locality thus less area of house and vice versa. There are of course other parameters which we are not considering at the moment. We say there is an interaction effect happening between height and bodyweight.

The activation function takes into account the interaction effects in different parameters and does a transformation that leads to decision of passing value of neuron into the next layer.

Why do we need non-linear function?



Activation function, Source

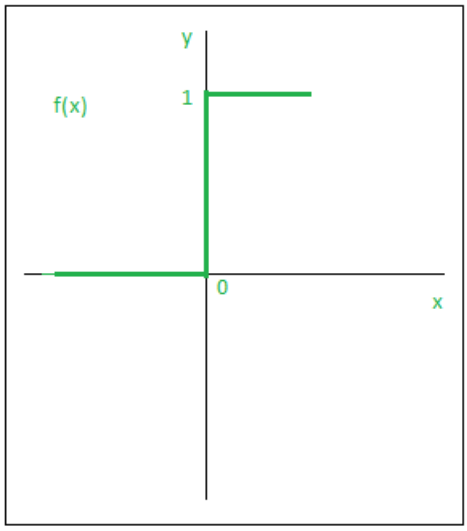
At some point in your deep learning journey you probably must have came across the Universal Approximation Theorem.

According to **universal approximation theorem,** "a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of **R***n*, under mild assumptions on the activation function."

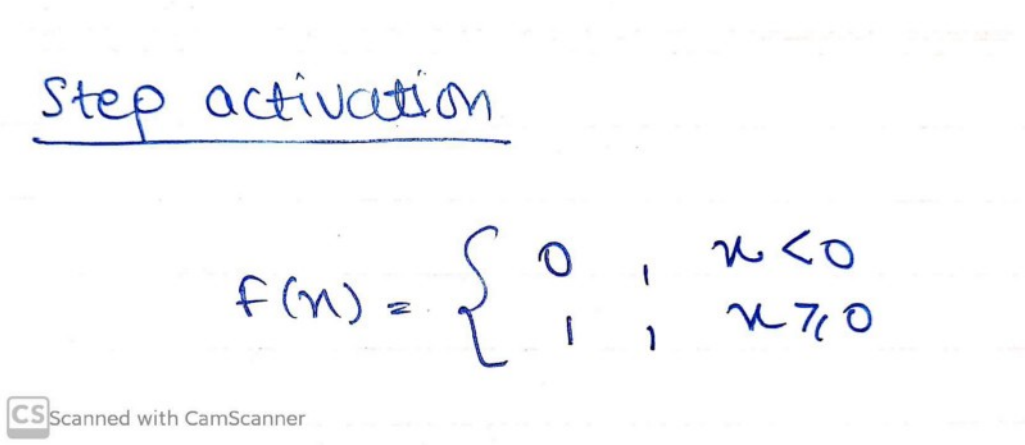
Function with multiple degrees are known as non-linear functions, If there would be no non-linear activation functions then we can only represent a linear relation between input and output and hence UAT won't hold its power. The representation power of deep Neural Network is due to its Non-linear activation functions.

Step activation function

It is one of the most basic type of activation function, it follows the threshold approach. If the input value is following a certain threshold, the neuron is activated and the signals are passed to next layers.



Graphical representation of Binary Step function



This is a great activation function but there are several drawbacks associated with it.

Suppose you're creating a binary classifier, that can only classify in "0" or "1" class, it would work perfect for this type of problem. Now imagine the case if classes are more sort of continuous value, now everything greater than 0 will be classified in class "1"

So we want something to give us intermediate (analog) activation values rather than saying "activated" or not (binary).

Pros

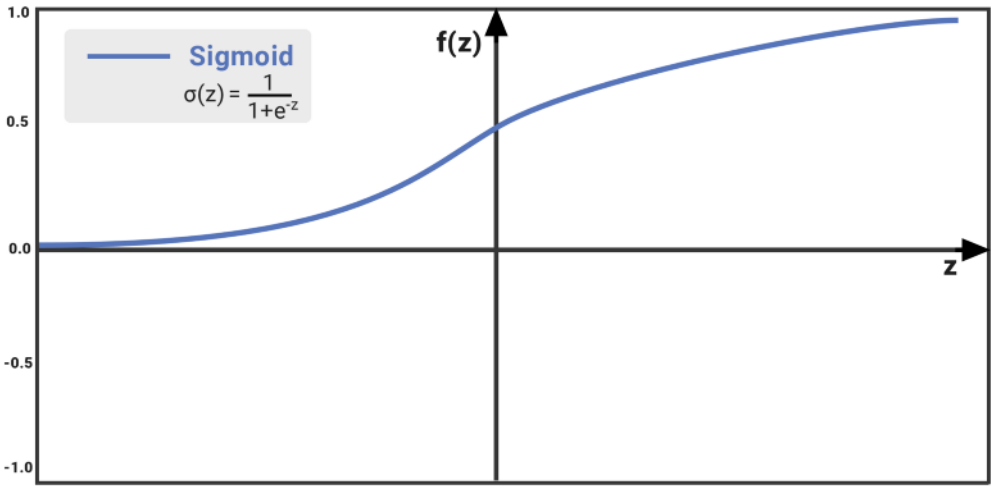
- It gives a range of activations, so it is not binary activation.
- We can definitely connect a few neurons together and if more than 1 fires, we could take the max (or softmax) and decide based on that.

Cons

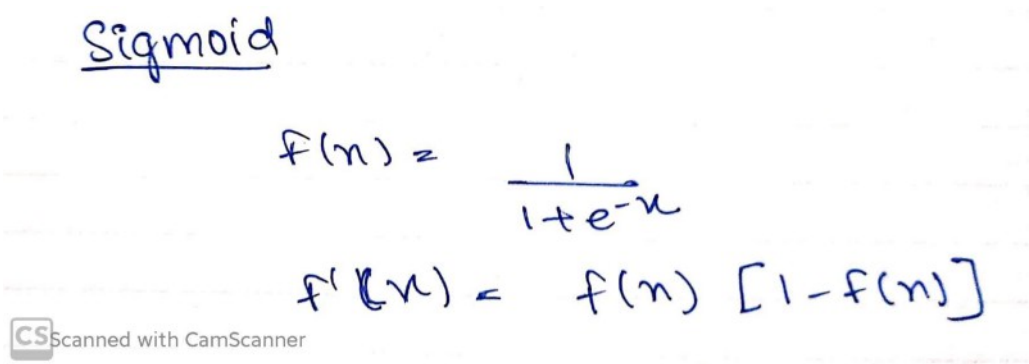
- For this function, derivative is a constant.
- If there is an error in prediction, the changes made by back propagation is constant and not depending on the change in input delta(x).

Sigmoid

This is widely used activation function. This is a smooth function and is continuously differentiable. The biggest advantage, that it has over step function is that it is non-linear.



Graphical representation of sigmoid function



On observation, we see that the value of *f(z)* increases but at a very slow rate. The mathematical reason is that as *z* (on the x-axis) increases, the value of e exponent -z becomes infinitesimally small and the value of *f(z)* become equals to 1 at some point.

In simple words, the function is susceptible to the vanishing gradient problem. Another issue with this function appears when we have multiple hidden layers in our neural network. All the values we are getting through this activation function are positive and sigmoid churns out values of different magnitudes between 0–1 range so it becomes hard to optimize i.e. if any of neuron saturates then weight update won't

happen and our model would fail.

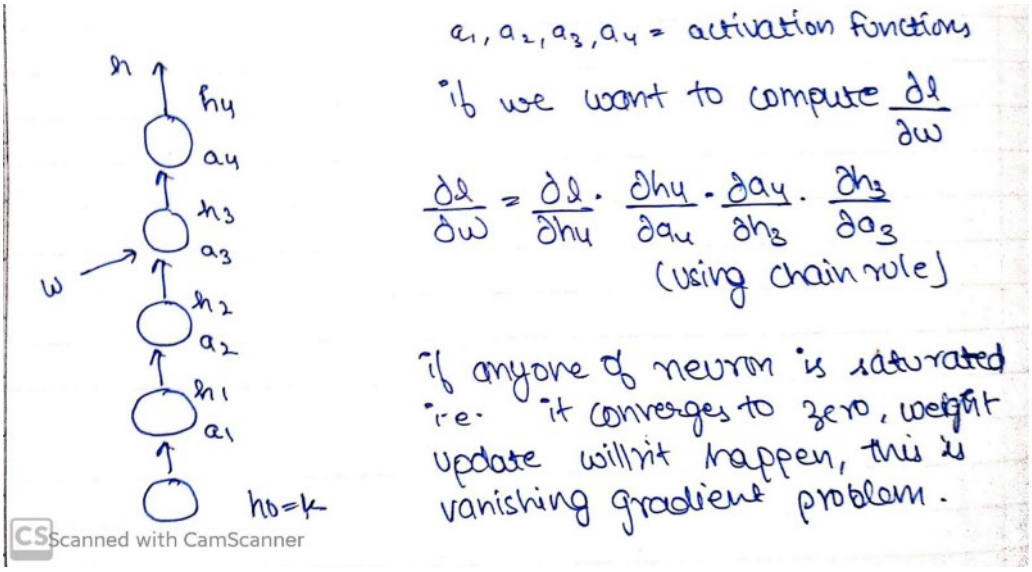
1. While using sigmoid function always initialize the weights to smaller value.
2. It is not zero centered(it is only one sided function i.e. always greater than 1)
3. This function is computationally expensive (because of **e^x)**

Pros

- 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".

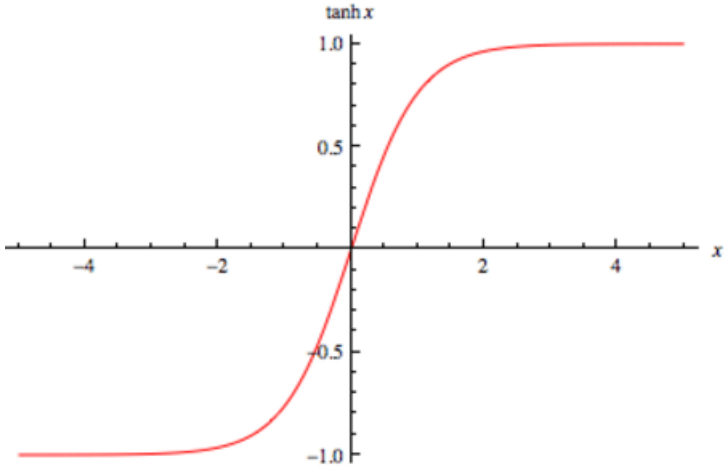


- Its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
- Sigmoids saturate and kill gradients.
- The network refuses to learn further or is drastically slow (depending on use case and until gradient /computation gets hit by floating point value limits).

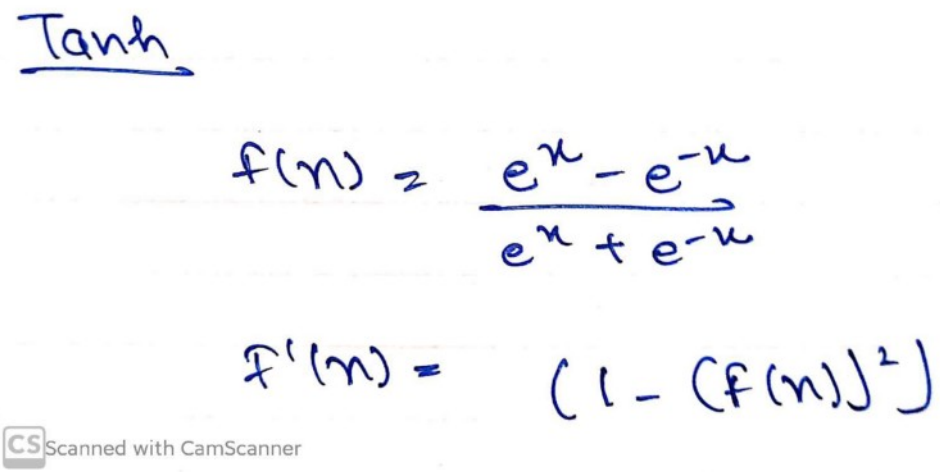
Tanh(widely used in RNN)

This function is advanced version of sigmoid function, in sigmoid we saw our values were one sided that is bound between (0,1] however in the case of tanh these are bounded between [-1,1].

The derivative of the Tanh function is steeper as compared to the sigmoid function. Our choice of using Sigmoid or Tanh really depends on the requirement of the gradient for the problem statement.



Graphical representation of Tanh function



Pros

- The gradient is stronger for tanh than sigmoid (derivatives are steeper).

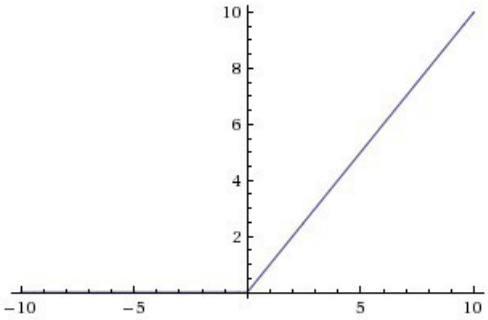
Cons

- Tanh also has the vanishing gradient problem.
- It is also computationally expensive.

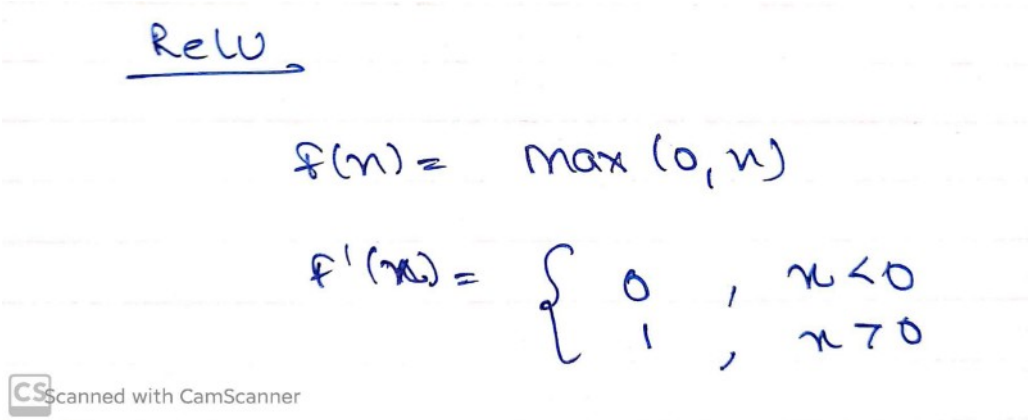
ReLU(widely used in CNN)

It stands for Rectified Linear Unit and unlike it's name it is a non linear activation function.

The ReLu function is as shown above. It gives an output x if x is positive and 0 otherwise.



Graphical representation of ReLu



The combinations of ReLu are also non linear, that means we can stack layers. It is not bound though. The range of ReLu is [0, inf). This means it can pump up the activation.

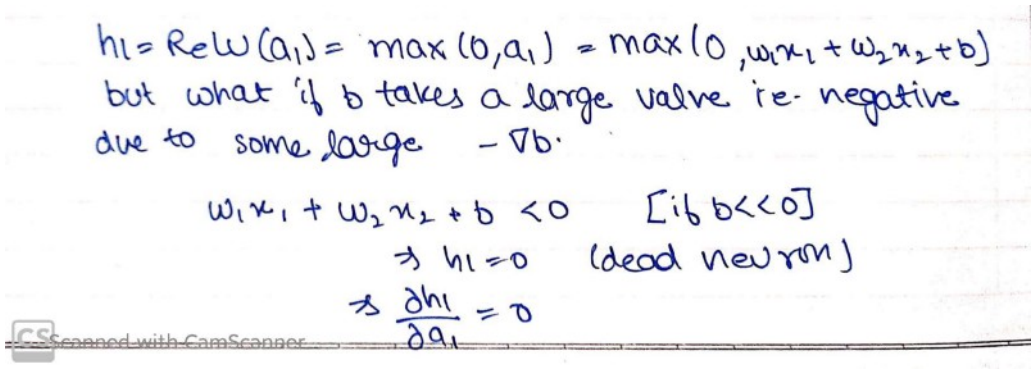
Unlike sigmoid and tanh it doesn't fire all neurons together rather, at a time only a few neurons are activated making the network sparse making it efficient. It is also computationally economic against former.

Pros

- It avoids and rectifies vanishing gradient problem.
- Doesn't saturate in positive region.
- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.

Cons

- 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. Simply saying that ReLu could result in Dead Neurons.
- It suffers from ReLu Dying problem.



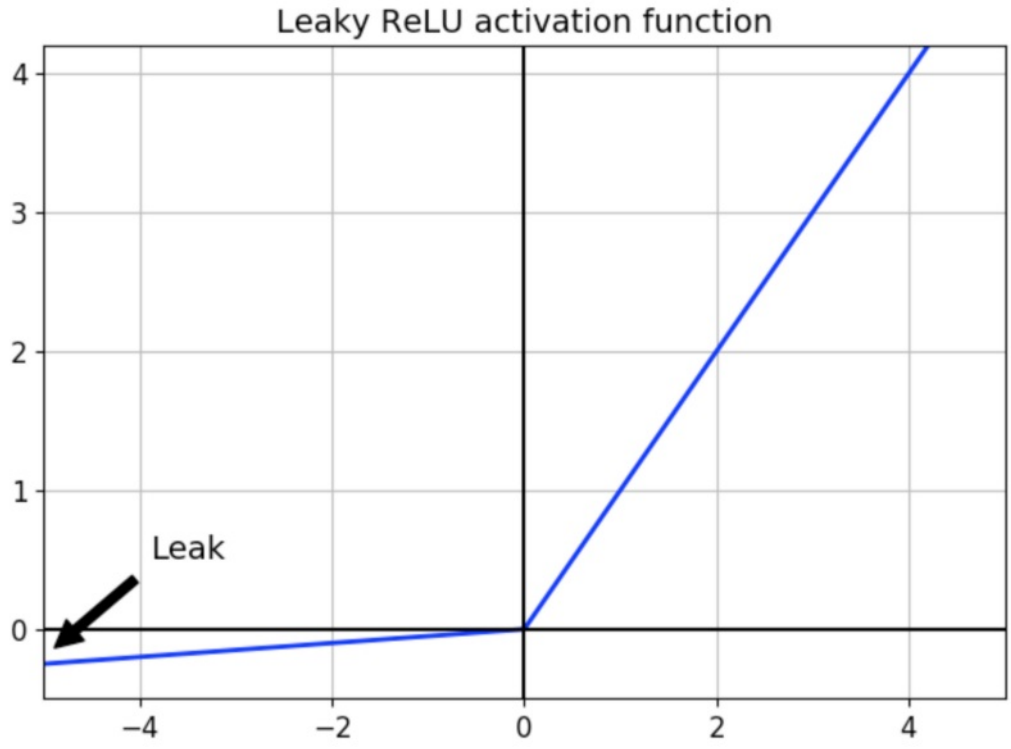
- The range of ReLu is [0, inf). This means it can blow up the activation.

As a final note on ReLU, there is a way to counter the dying ReLU problem by using a modified version of the function called the 'Leaky' ReLU.

Leaky ReLU

It is improved version of the ReLU function. Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x.

This leaky value is given as a value of 0.01 if given a different value near zero, the name of the function changes randomly as Leaky ReLU. The definition range of the leaky-ReLU continues to be minus infinity. It can be defined as:



Graphical representation of Leaky ReLu

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

$$f'(x) = \begin{cases} 0.01, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Pros

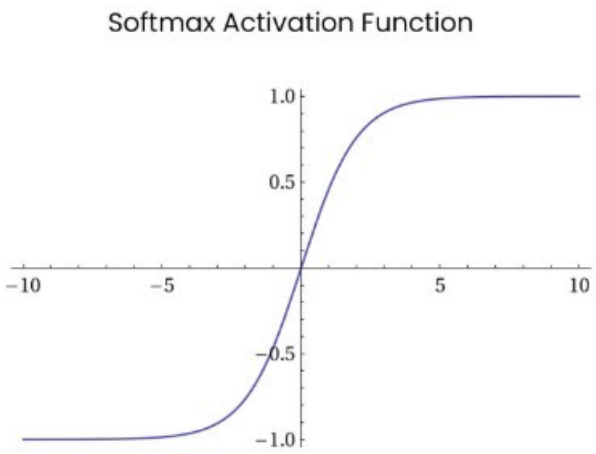
- Leaky ReLUs are one attempt to fix the “dying ReLU” problem by having a small negative slope (of 0.01, or so).
- Easy to compute.
- Close to zero centered output.

Cons

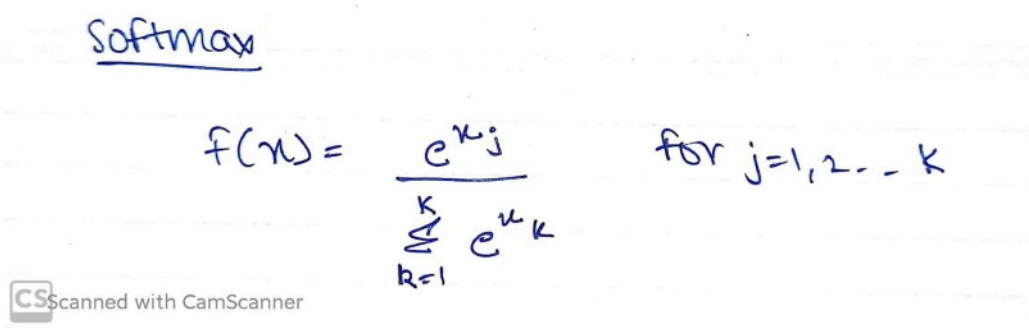
- As it possess linearity, it can’t be used for the complex Classification. It lags behind the Sigmoid and Tanh for some of the use cases.
- Leaky ReLU does not provide consistent predictions for negative input values.

Softmax

Softmax function calculates the probabilities distribution of the event over ’n’ different events. This function will calculate the probabilities of each target class over all possible target inputs. Later the result will be helpful for determining the target class for the given inputs.



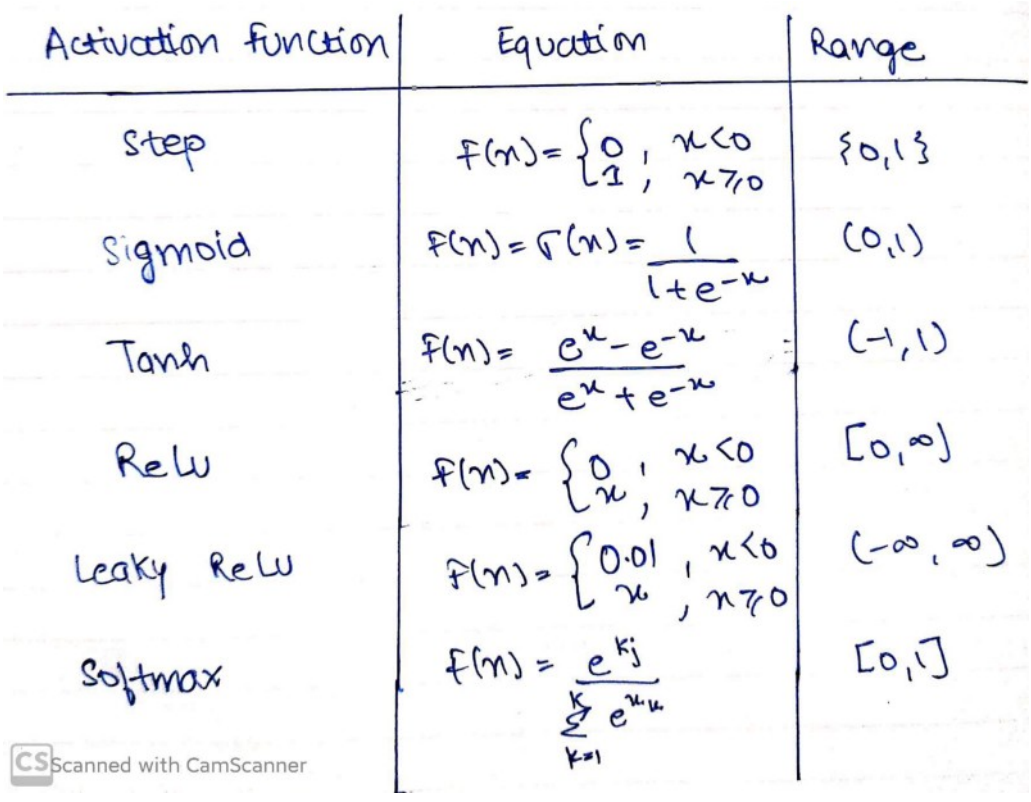
Graphical Representation of Softmax function



While building a network for a multiclass problem, the output layer would have as many neurons as the number of classes in the target. For instance if you have three classes, there would be three neurons in the output layer. Suppose you got the output from the neurons as [1.2 , 0.9 , 0.75].

Applying the softmax function over these values, you will get the following result—[0.42 , 0.31, 0.27]. These represent the probability for the data point belonging to each class.

Which activation function must be used?



Depending upon the properties of the problem we might be able to make a better choice for easy and quicker convergence of the network.

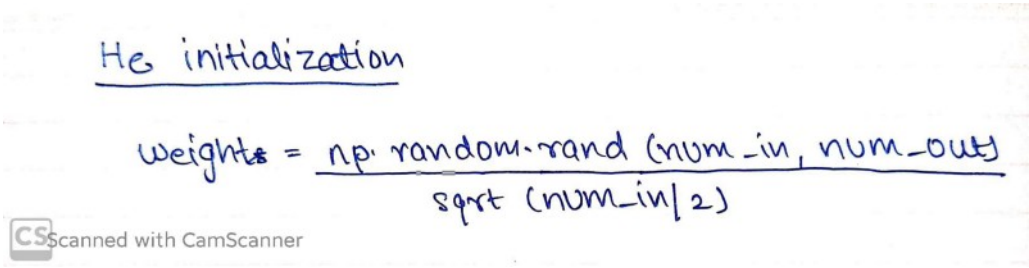
- **Sigmoid functions** and their combinations generally work better in the case of classifiers.
- **Sigmoids and tanh functions** are sometimes avoided due to the vanishing gradient problem.
- **ReLU** function is a general activation function and is used in most cases these days.

- If we encounter a case of dead neurons in our networks the **leaky ReLU** function is the best choice.
- Always keep in mind that **ReLU** function should only be used in the hidden layers.
- As a rule of thumb, you can begin with using **ReLU** function and then move over to other activation functions in case it doesn't provide with optimum results.

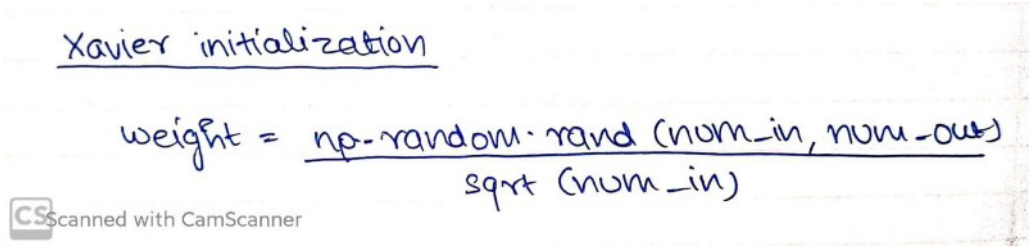
So the activation function is a critical optimization problem that can only be decided on all the requirement of your deep learning model.

Best Practices

1. Never initialize weights from 0 or with same values as then there won't be update, this is known as **symmetry breaking problem**.
2. Always normalize the inputs.
3. Use **ReLU** in hidden layer activation, but be careful with the learning rate and monitor the fraction of dead units.
4. If **ReLU** is giving problems. Try Leaky ReLU, PReLU, Maxout. Do not use sigmoid
5. Never initialize the weights to large value otherwise model will blow up.
6. Weights initialized must be **inversely proportion to input.**
7. The **sigmoid** and **hyperbolic tangent** activation functions cannot be used in networks with many layers due to the vanishing gradient problem.
8. Use **He initialization** for ReLu and Leaky ReLu.



9. Use **Xavier initialization** for tanh, sigmoid.



Conclusion

Hopefully, this article has increased your understanding of optimization techniques.

As always, thank you so much for reading, and please share this article if you found it useful! :)

References:

[1] Comparison of Activation Functions for Deep Neural Networks By Ayyüce Kızrak

[2] Fundamentals of Deep Learning—Activation Functions and When to

Use Them? By Disha Shree Gupta

[3] Understanding Activation Functions in Neural Networks By Avinash Sharma V

[4] 7 Types of Neural Network Activation Functions: How to Choose?

[5] Activation Functions in Neural Networks By Hamza Mahmood

The cover template is designed by me on canva.com, the source is mentioned on every visual, and the un-mentioned visuals are either from my notebook or are designed by me on Paint 3D.

Feel free to connect:

Join me at www.dakshtrehan.com

LinkedIN ~ https://www.linkedin.com/in/dakshtrehan/

Instagram ~ https://www.instagram.com/_daksh_trehan_/

Github ~ https://github.com/dakshtrehan

Check my other articles:-

Detecting COVID-19 using Deep Learning
*A practical approach to help medical practitioners helping us in the battle against COVID-19*towardsdatascience.com
Parameters Optimization Explained
*A brief yet descriptive guide to Gradient Descent, ADAM, ADAGRAD, RMSProp*towardsdatascience.com
Gradient Descent Explained
*A comprehensive guide to Gradient Descent*towardsdatascience.com
Logistic Regression Explained
*Explaining Logistic Regression as easy as it could be.*towardsdatascience.com
Linear Regression Explained
*Explaining Linear Regression as easy as it could be.*medium.com
Relating Machine Learning Techniques to Real-Life.
*Explaining types of ML model as easy as it could be.*levelup.gitconnected.com
Determining perfect fit for your ML model.
Teaching Overfitting vs Underfitting vs Perfect fit in easiest way. medium.com
Serving Data Science to a Rookie
*So, last week my team head asked me to interview some of the possible interns for the team, for the role of data...*medium.com

Follow for further Machine Learning/ Deep Learning blogs.

Cheers.

By Daksh Trehan on May 29, 2020.

Canonical link

Exported from Medium on July 15, 2020.