Activation Function

**Activation Function:**

Activation functions are an extremely important feature of the artificial neural networks.

They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

The activation function is the non linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.

https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/10/17123344/act.png

## Can we do without an activation function?:

When we do not have the activation function the weights and bias would simply do a linear transformation.

A linear equation is simple to solve but is limited in its capacity to solve complex problems. A neural network without an activation function is essentially just a linear regression model.

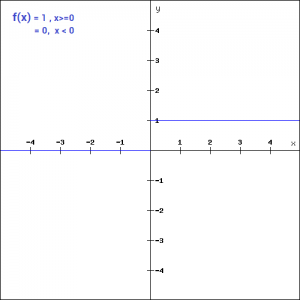
The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

We would want our neural networks to work on complicated tasks like language translations and image classifications. Linear transformations would never be able to perform such tasks.

Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Without the differentiable non linear function, this would not be possible.

### Binary Step Function

It is a threshold based classifier i.e. whether or not the neuron should be activated. If the value Y is above a given threshold value then activate the neuron else leave it deactivated.

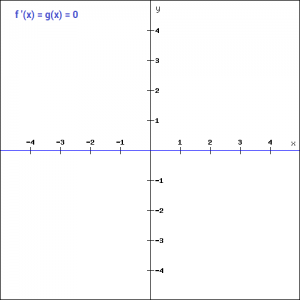


f(x) = 1, x>=0

=0, x<0

**Cons**:Moreover, the gradient of the step function is zero. This makes the step function not so useful since during back-propagation when the gradients of the activation functions are sent for error calculations to improve and optimize the results. The gradient of the step function reduces it all to zero and improvement of the models doesn’t really happen.

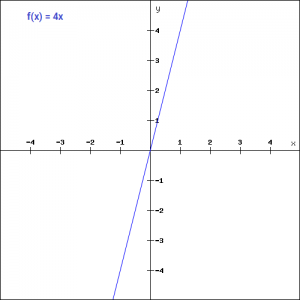
f '(x) = 0, for all x



### Linear Function

We saw the problem with the step function, the gradient being zero, it was impossible to update gradient during the backpropagation. Instead of a simple step function, we can try using a linear function. We can define the function as-

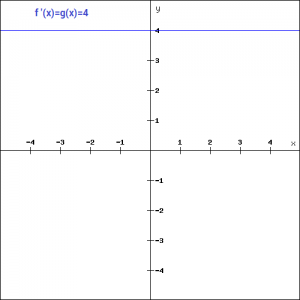
f(x)=ax



We have taken a as 4 in the figure above. Here the activation is proportional to the input. The input x, will be transformed to ax. This can be applied to various neurons and multiple neurons can be activated at the same time. Now, when we have multiple classes, we can choose the one which has the maximum value. But we still have an issue here. Let’s look at the derivative of this function.

**Cons**:

f'(x) = a



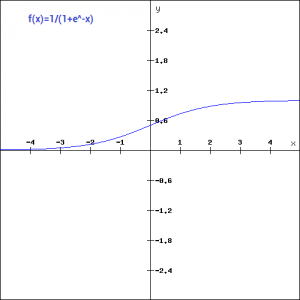
The derivative of a linear function is constant i.e. it does not depend upon the input value x.

This means that every time we do a back propagation, the gradient would be the same. And this is a big problem, we are not really improving the error since the gradient is pretty much the same. And not just that suppose we are trying to perform a complicated task for which we need multiple layers in our network. Now if each layer has a linear transformation, no matter how many layers we have the final output is nothing but a linear transformation of the input. Hence, linear function might be ideal for simple tasks where interpretabilityis highly desired.

### Sigmoid Function

Sigmoid is a widely used activation function. It is of the form-

f(x)=1/(1+e^-x)



This is a smooth function and is continuously differentiable.

The biggest advantage that it has over step and linear function is that it is non-linear.

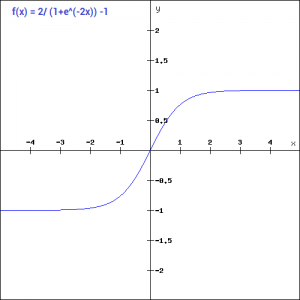
### Tanh Function

The tanh function is very similar to the sigmoid function. It is actually just a scaled version of the sigmoid function.

tanh(x)=2sigmoid(2x)-1

It can be directly written as –

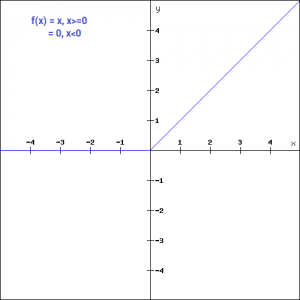
tanh(x)=2/(1+e^(-2x)) -1



### ReLU Function

The ReLU function is the Rectified linear unit. It is the most widely used activation function. It is defined as-

f(x)=max(0,x)

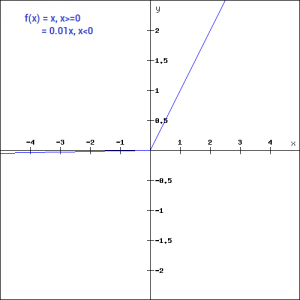


### Leaky ReLU Function

Leaky ReLU function is nothing but an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for x<0, which made the neurons die for activations in that region. Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x. It can be defined as-

f(x)= ax, x<0

= x, x>=0



### Softmax Function

The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems. The sigmoid function as we saw earlier was able to handle just two classes. What shall we do when we are trying to handle multiple classes. Just classifying yes or no for a single class would not help then. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs. This essentially gives the probability of the input being in a particular class. It can be defined as –

https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/10/17014509/softmax.png

Let’s say for example we have the outputs as-

[1.2 , 0.9 , 0.75], When we apply the softmax function we would get [0.42 ,  0.31, 0.27]. So now we can use these as probabilities for the value to be in each class.

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.