Neural Network Activation Functions

Sam Haley

Activation Functions

- Very important feature of neural networks
- Determine if the neuron should be activated/fired based upon the input
  - Output = Activation(Σ(weight × input) + bias)
- Activation functions perform the non-linear transformation to the input allowing approximation of complex functions
  - Neural networks without activation functions are simply linear regression models
- Activation functions covered in this brief:
  - Binary Step function
  - Linear Function
  - Sigmoid
  - Tanh
  - ReLU
  - Leaky ReLU
  - Softmax

Overview of Neural Network Training

In order to understand how activation functions impact neural networks it is important to have a general understanding of how neural networks are trained

In general when you train neural networks you have two layers of abstraction:

- Gradient Computation – Back-propagation algorithm
- Optimization (Use the gradients to update the weights) –Stochastic Gradient Descent (SGD) algorithm (many others also exist)

Back-propagation computes the impact of each weight on the cost function (error) as a partial derivative $\frac{\partial E}{\partial w}$ (gradient):

- $\frac{\partial E}{\partial w}$ is computed for each weight in the network
- $\frac{\partial E}{\partial w}$ is decomposed using the chain rule $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial a}\frac{\partial a}{\partial w}$. For any one weight:
    - $\frac{\partial E}{\partial o}$ = derivative of error w.r.t. the node output (in the case of a euclidean distance cost function $\frac{\partial E}{\partial o}$ = output – truth)
    - $\frac{\partial o}{\partial a}$ = derivative of the node output w.r.t. node input (node input is the sum of all preceding layer output multiplied by their weights)
        - The activation function is what takes the input and creates the output
        - $\frac{\partial o}{\partial a}$ = the derivative of the activation function (i.e. $\frac{\partial o}{\partial a}$ = f(a)*(1-f(a)) when f is the sigmoid function)
    - $\frac{\partial a}{\partial w}$ = derivative of node input w.r.t. weight ($\frac{\partial a}{\partial w}$ = preceding layer outputs)

SGD then uses the gradients computed using back-propagation to update the weights

- A ratio of the gradient is subtracted from the weight (this ratio is the learning rate)

Key Takeaway: Due to the impact of the activation function gradients (derivatives) to the back propagation algorithm it is important to understand what the gradients of activation functions look like and how they behave in the presence of different inputs

Binary Step Function

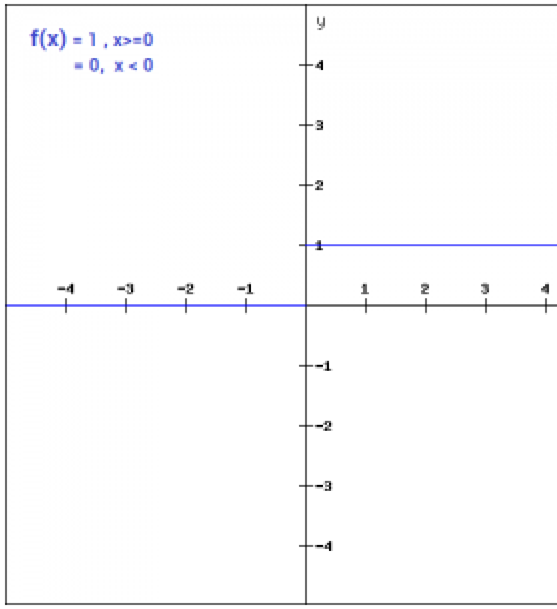
f(x) = 1, x>=0

Can be used when creating a binary classifier

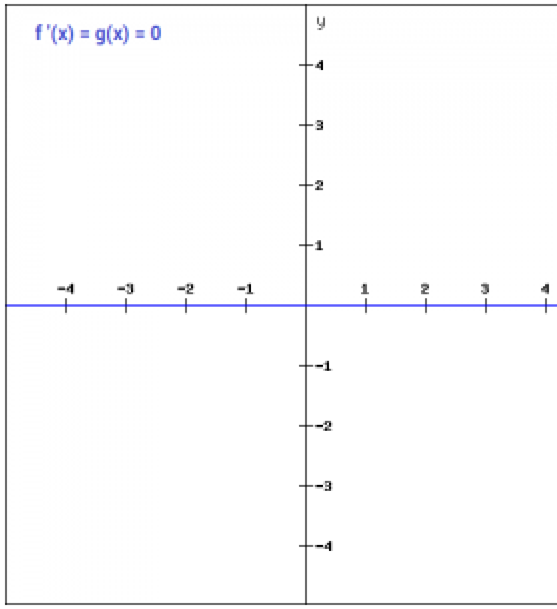
Disadvantages:

- Cannot be used when classifying into multiple classes
- The gradient is zero:
  - Gradients of activation functions are used in back-propagation to optimize the model
  - The zero gradient reduces everything to zero and model improvement does not happen

Binary Step Function



Binary Step Function Gradient



Linear Function

$f(x) = ax$

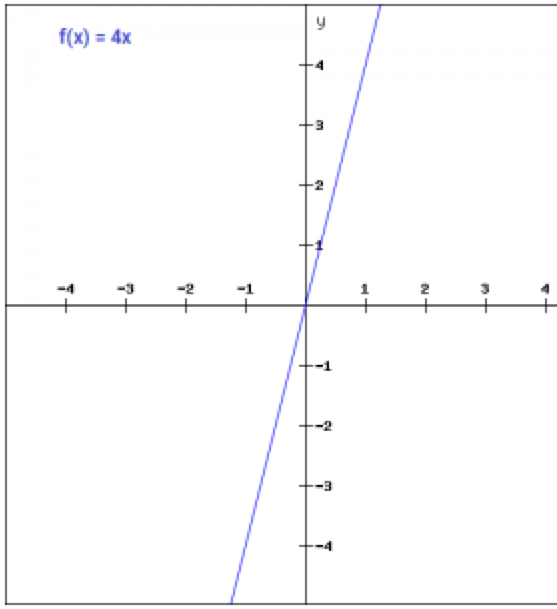
Advantages:

- Activation is proportional to the input
- Can handle classification of multiple classes

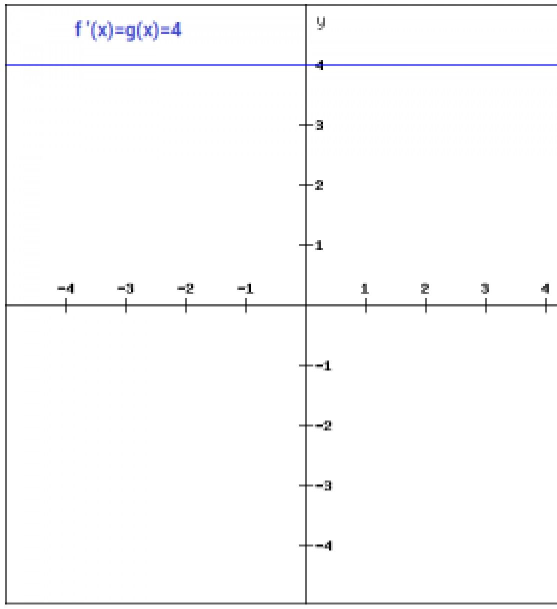
Disadvantages:

- The gradient is a constant value (4 in the example to the right):
    - Cannot improve model with back-propagation because the gradient is always constant
    - If every layer uses linear activation function the output is just a linear transformation of the input (for this same reason the linear function is good for cases when interpretability is desired)

Linear Function



Linear Function Gradient



Sigmoid Function

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

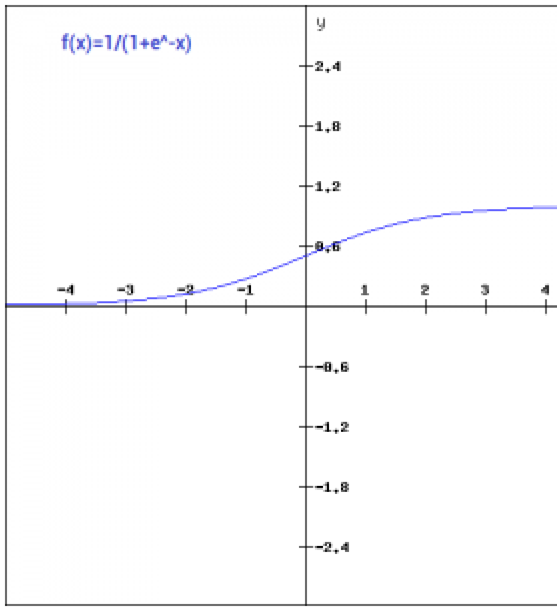
Advantages:

- Non-linear
- Good for classification
  - Gradient is high for input values close to zero and low in other regions
  - Function attempts to push values towards extremes (yes/no)

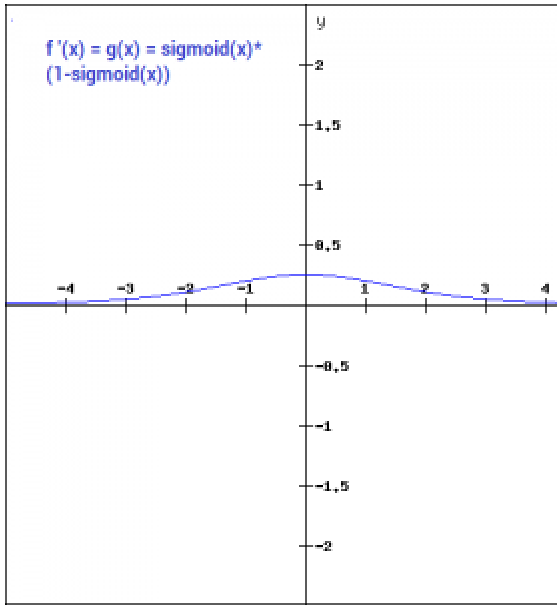
Disadvantages:

- Vanishing gradient:
  - The gradient is flat at input values far from zero
  - When the function approaches these regions the network stops truly learning
- Output is always positive
  - There are times when it would be more desirable to not have all output from each layer be of the same sign

Sigmoid Function



Sigmoid Function Gradient



Tanh Function

$f(x) = 2/(1+e^{-2x}) - 1$

Simply a scaled version of the sigmoid function

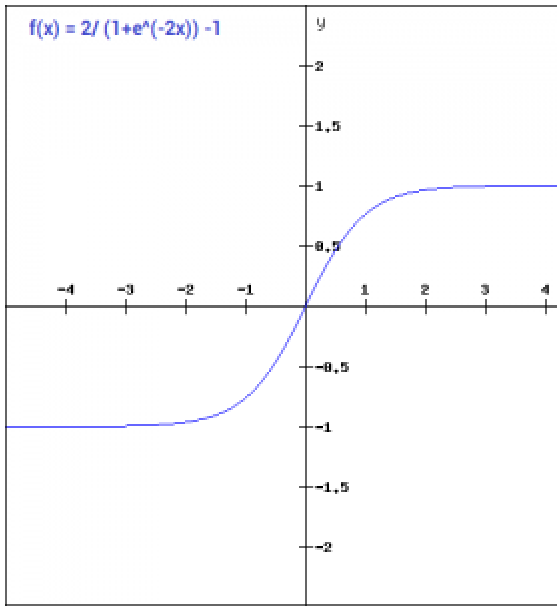
Advantages:

- Solves the sigmoid function problem of all output values all having the same sign
- Gradient is steeper than the sigmoid function which for some problems can be an advantage

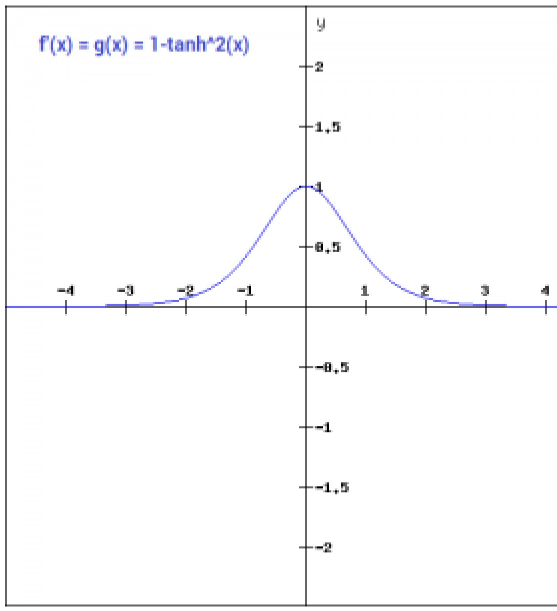
Disadvantages:

- Also has a vanishing gradient:
  - The gradient is flat at input values far from zero
  - When the function approaches these regions the network stops really learning

Tanh Function



Tanh Function Gradient



ReLU Function

f(x) = max(0,x)

Most widely used activation function

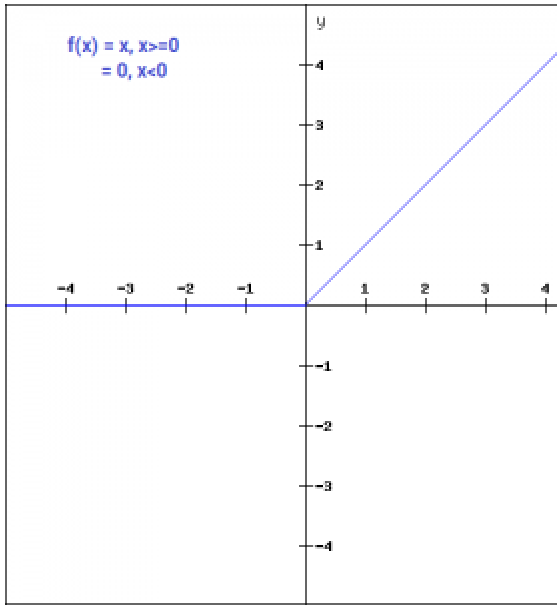
Advantages:

- Non-linear
- Sparse activation - all neurons are not activated at the same time (negative inputs yield zero)
- Makes computation more efficient

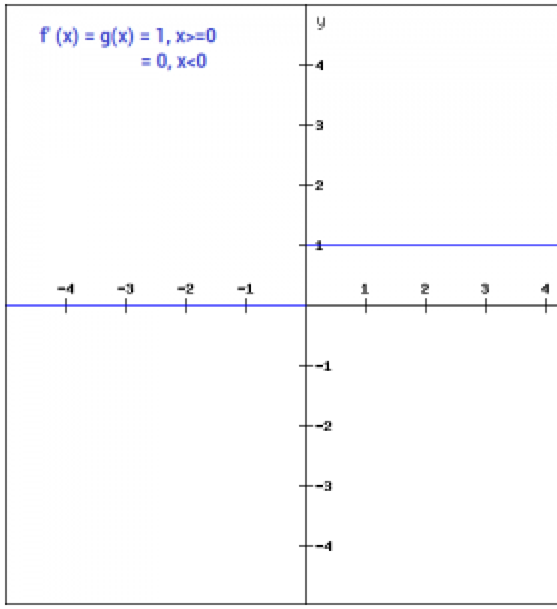
Disadvantages:

- Gradient is zero for negative inputs
- For negative input values weights are not updated during back-propagation
- Can create dead neurons that never get activated

ReLU Function



ReLU Function Gradient



eaky ReLU Function

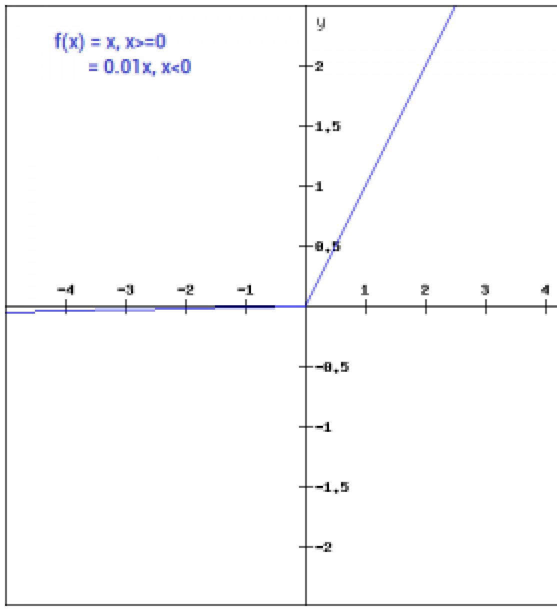
f(x) = ax for x<0, f(x) = x for x>=0

Slight modification of the ReLU function

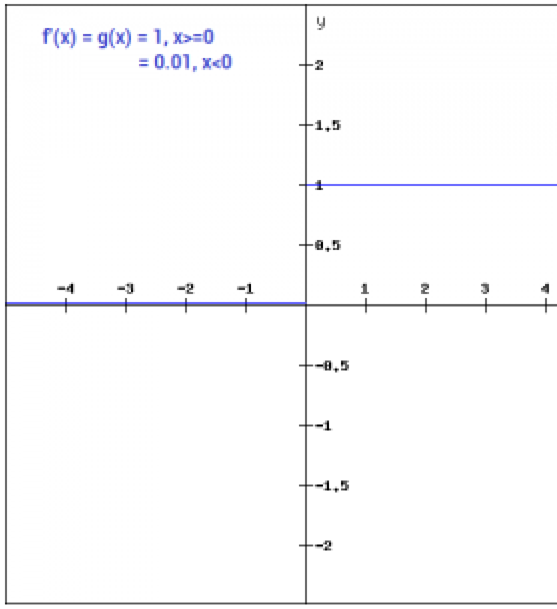
Advantages:

- Addresses the issue of dead neurons in ReLU
  - Gradient is non-zero (but usually very small) for negative input values
  - Neurons do not get caught in the dead region

Leaky ReLU Step Function



Leaky ReLU Function Gradient



oftmax Function

Type of sigmoid function

Commonly used in the output layer of a classifier

Useful for handling classification problems with multiple classes and essentially gives the probability of input being in a particular class

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Squeezes the output for each class between zero and one

hoosing the Correct Activation Function

There is not set rule of thumb, however depending on the problem we ma
be able to make better choices to get better convergence

Listed below are some guidelines for activation functions

- The ReLU function is a general activation function and should be used in most cases
- Sigmoid type functions (Sigmoid, Tanh) generally work better in the case of classifiers
- Sigmoid type functions are sometimes avoided due to the vanishing gradient problem
- If dead neurons are encountered leaky ReLU function is the best choice
- ReLU functions should only be used in the hidden layers (likely softmax for the output)
- Start with ReLU function and then move over to other activation functions if ReLU doesn’t provide optimum results

Sources

Analytics Vidya - Fundamentals of Deep Learning – Activation Functions and When to Use Them? - 102317

Alex Minaar - Neural Networks, Backpropagation, and Stochastic Gradient Descent

Wikipedia – Backpropagation

Stack Overflow - What is the difference between SGD and Back-Propagation