Activation Functions

13 July 2023 11:29 PM

Activation Functions are mathematical functions that determine the output of neural network. It ideally helps in identifying if the neuron should be activated or not.

Benefits of Activation Function:

1. Helps normalize the output of each neuron to range eg. 0 to1, -1 to 1, 0 to ∞.
2. Helps to make your neural network deal with non-linear data.

Activation Function List:

1. Linear Activation Function
2. Logistic Activation Function (sigmoid)
3. Hyperbolic Tangent Activation Function (Tan h)
4. Rectified Linear Unit (ReLU)
5. Leaky Rectified Linear Unit (LeakyReLU)
6. Softmax Activation Function

Use cases and Explanation of Activation Functions:

a. ***Linear Activation Function:***

Equation: $f(x) = x$
Keras code: activation = 'linear' OR activation = tf.keras.activations.linear
Which layer to be used: Output Layer
ML problems where this can be used: Regression

b. ***Logistic Activation Function (sigmoid):***

Equation: $f(x) = \frac{1}{1+e^{-\lambda z}}$ $\quad where\ \lambda = 1$

Output Range: 0 and 1
Default Threshold: Less than 0.5 ----> 0
Greater that 0.5 ----> 1
Disadvantages:

Vanishing Gradient Problem => For any given very high value or very low value of x, there is almost no change to the prediction, resulting in network refuse to learn.

$$\frac{\partial \text{error}}{\partial \omega_1} = \frac{\partial \text{error}}{\partial \text{output}} * \frac{\partial \text{output}}{\partial h_5} * \frac{\partial h_5}{\partial h_4} * \frac{\partial h_4}{\partial h_3} * \frac{\partial h_3}{\partial h_2} * \frac{\partial h_2}{\partial h_1} * \frac{\partial h_1}{\partial \omega_1}$$

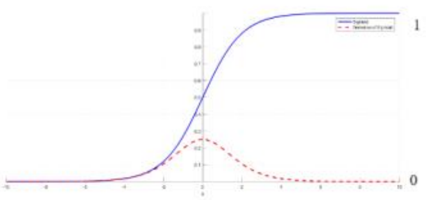
We know that for sigmoid, derivative is recall ==> $O_j(1 - O_j)$

Let's denote the sigmoid function as $\sigma(x) = \frac{1}{1+e^{-x}}$.

The derivative of the sigmoid is $\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$.

Here's a detailed derivation:

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{d}{dx}(1+e^{-x})^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}$$

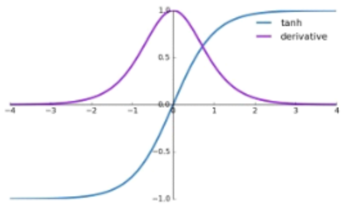


\# **Multiplying smaller vales results in smallest values thus we loss gradient value**

Solution: The root of the problem is the nature of sigmoid activation function derivative. Hence use a function which do not have this property of squashing(range bound) e.g. Rectified Linear Unit **(ReLu)**
Keras code: activation = 'sigmoid' OR activation = tf.keras.activations.sigmoid
Which layer to be used: Output Layer
ML problems where this can be used: Binary Classification / Multi-class Classification

c. ***Hyperbolic Tangent Activation Function (Tanh(z)):***

Equation: Equation: $f(x) = \frac{2}{1+e^{-2z}}$



Output Range: *-1 to 1*
Keras code: activation = 'Tanh' OR activation = tf.keras.activations.tanh
Which layer to be used: Hidden Layer
ML problems where this can be used: Binary Classification
Disadvantages: i) Vanishing Gradient Problem ii) Slow Training time

d. Rectified Linear Unit (ReLU):

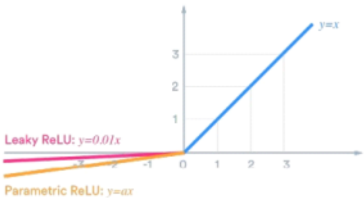
Equation: $f(x) = \max(o, z)$
Keras code: activation = 'relu' OR activation = tf.keras.activations.relu
Which layer to be used: Hidden Layer
ML problems where this can be used: Classification
Advantages: i) Behaviour of ReLU derivative rectifies vanishing gradient problem
ii) Computation is Faster as compared to **sigmoid** and **tanh**
Disadvantages: Dead Neuron [Dying Neuron] { Reason: Zero derivatives for every negative value }

$$\frac{\partial \text{error}}{\partial \omega_1} = \frac{\partial \text{error}}{\partial \text{output}} * \frac{\partial \text{output}}{\partial h_5} * \frac{\partial h_5}{\partial h_4} * \frac{\partial h_4}{\partial h_3} * \frac{\partial h_3}{\partial h_2} * \frac{\partial h_2}{\partial h_1} * \frac{\partial h_1}{\partial \omega_1}$$

Solution: Leaky ReLU

e. Leaky Rectified Linear Unit (LeakyReLU):

Equation: α = 0.01



Keras code: activation = 'leakyrelu' OR activation = tf.keras.activations.LeakyReLU(alpha=0.01)
Which layer to be used: Hidden Layer
ML problems where this can be used: Classification

f. Softmax Activation Function

Equation: $S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$

Returns probability of each class in a multi-class label

e.g. *Output z* *Softmax* *Probabilities*

$$\begin{bmatrix}1.3\\5.1\\2.2\\0.7\\1.1\end{bmatrix} \quad \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad \begin{bmatrix}0.02\\0.90\\0.05\\0.01\\0.02\end{bmatrix}$$

Keras code: activation = 'softmax' OR activation = tf.keras.activations.softmax
Which layer to be used: Output Layer is MULTICLASS single label classification
ML problems where this can be used: Classification