Activation Functions

As we mentioned earlier, ANNs are a crucial component of many structures that are helping revolutionize the world around us. But have you ever wondered, how do ANNs deliver state-of-the-art performance to find solutions to real-world problems?

The answer is – Activation Functions.

ANNs use the Activation Function in soft computing (AFs) to perform complex computations in the hidden layers and then transfer the result to the output layer. The primary purpose of AFs or Activation Function in soft computing is to introduce non-linear properties in the neural network.

They convert the linear input signals of a node into non-linear output signals to facilitate the learning of high order polynomials that go beyond one degree for deep networks. A unique aspect of AFs is that they are differentiable – this helps them function during the backpropagation of the neural networks.

In order to determine whether a neuron should be activated or not, the activation function in neural network calculate a weighted total and then adds bias to it. The activation functions' goal is to make a neuron's output less linear.

Explanation :-

We know that neurons in neural networks behave according to weight, bias, and their respective activation roles. We would adjust the weights and biases of the neurons in a neural network based on the output inaccuracy. Back-propagation is the term for this method. Activation functions make back-propagation possible since they provide the gradients and errors required to modify the weights and biases.

Why do we require Non-linear activation functions: An unactivated neural network is essentially a linear regression model. The activation function transforms the input in a non-linear way, enabling it to learn and carry out more difficult tasks. Here is the activation function in neural network example for you.

Mathematical proof:

Think we have a neutral net something like this:

Elements of the diagram :-

Hidden layer i.e. layer 1 :-

$z(1) = W(1)X + b(1)$

$a(1) = z(1)$

Here,

z(1) is the vectorized output of layer 1

W(1) be the vectorized weights assigned to neurons of hidden layer i.e. w1, w2, w3 and w4

X be the vectorized input features i.e. i1 and i2

b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2

a(1) is the vectorized form of any linear function.

(Note: We won't be considering activation function here)

Layer 2 i.e. output layer:

// Note : Input for layer

// 2 is output from layer 1

$z(2) = W(2)a(1) + b(2)$

$a(2) = z(2)$

Calculation at Output layer:

// Putting value of z(1) here

$z(2) = (W(2) * [W(1)X + b(1)]) + b(2)$

$z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]$

Let,

$[W(2) * W(1)] = W$

$[W(2)*b(1) + b(2)] = b$

Final output : $z(2) = W*X + b$

Which is again a linear function

This activation function in neural network example yields a linear function once more even after the addition of a hidden layer, leading us to the conclusion that no matter how many hidden layers we add to a neural network, all of them will behave consistently because of the union of two linear functions yields a linear function as well. With only a linear function attached to it, a neuron cannot learn. It can learn according to the difference in mistake rate given a non-linear activation function.

So, an activation function is required.

What is the need for non-linearity?

If activation functions are not applied, the output signal would be a linear function, which is a polynomial of one degree. While it is easy to solve linear equations, they have a limited complexity quotient and hence, have less power to learn complex functional mappings from data. Thus, without AFs, a neural network would be a linear regression model with limited abilities.

This is certainly not what we want from a neural network. The task of neural networks is to compute highly complicated calculations. Furthermore, without AFs, neural networks cannot learn and model other complicated data, including images, speech, videos, audio, etc.

AFs help neural networks to make sense of complicated, high dimensional, and non-linear Big Data sets that have an intricate architecture – they contain multiple hidden layers in between the input and output layer.

Read: Deep Learning Vs Neural Network

Now, without further ado, let's dive into the different types of activation functions used in ANNs.

Types of Activation Functions

1. Sigmoid Function

In an ANN, the sigmoid function is a non-linear AF used primarily in feedforward neural networks. It is a differentiable real function, defined for real input values, and containing positive derivatives everywhere with a specific degree of smoothness. The sigmoid function appears in the output layer of the deep learning models and is used for predicting probability-based outputs. The sigmoid function is represented as:

$$f(x) = \left(\frac{1}{(1 + exp^{-x})} \right) - (1.4)$$

Source

Generally, the derivatives of the sigmoid function are applied to learning algorithms. The graph of the sigmoid function is 'S' shaped.

Some of the major drawbacks of the sigmoid function include gradient saturation, slow convergence, sharp damp gradients during backpropagation from within deeper hidden layers to the input layers, and non-zero centered output that causes the gradient updates to propagate in varying directions.

2. Hyperbolic Tangent Function (Tanh)

The hyperbolic tangent function, a.k.a., the tanh function, is another type of AF. It is a smoother, zero-centered function having a range between -1 to 1. As a result, the output of the tanh function is represented by:

$$f(x_i) = \frac{exp\ (x_i)}{\sum_j exp\ (x_j)} \quad -(1.12)$$

Source

The tanh function is much more extensively used than the sigmoid function since it delivers better training performance for multilayer neural networks. The biggest advantage of the tanh function is that it produces a zero-centered output, thereby supporting the backpropagation process. The tanh function has been mostly used in recurrent neural networks for natural language processing and speech recognition tasks.

However, the tanh function, too, has a limitation – just like the sigmoid function, it cannot solve the vanishing gradient problem. Also, the tanh function can only attain a gradient of 1 when the input value is 0 (x is zero). As a result, the function can produce some dead neurons during the computation process.