Activation Function

Activation functions in Neural Network are used in a neural network to compute the weighted sum of inputs and biases, which is in turn used to decide whether a neuron can be activated or not. The activation functions are also referred to as *transfer functions* in some literature. These can either be linear or nonlinear depending on the function it represents and is used to control the output of neural networks across different domains.

Several activation functions are discussed below:

1. **Sigmoid Activation Function**

   Sigmoid takes a real value as the input and outputs another value between 0 and 1. Its formula is:

   $$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

   It is commonly used in binary classification and suitable when probabilities are required as output. But it can suffer from vanishing gradient problem and it is computationally expensive due to the exponential function.

2. **Tanh**

   It shares a few things in common with the sigmoid activation function. Unlike a sigmoid function that will map input values between 0 and 1, the Tanh will map values between -1 and 1. Its formula is

   $$\text{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

   It is suitable classification tasks when output is in a bipolar range(-1,1). It can also suffer from vanishing gradient problem, like sigmoid activation function.

3. **Rectified Linear Units**

   In modern neural networks, the default recommendation is to use the Rectified Linear Unit or ReLU, defined by the activation function:

   $$\text{ReLu}(x) = \max(0,x)$$

ReLu outputs the input directly if it is positive, otherwise, it outputs zero.

It accelerates training by avoiding vanishing gradient problem. It is widely used in hidden layer of neural network because of its simplicity and efficiency. It can cause dying ReLu problem if weights of neurons are poorly initialized or learning rate is too high.

4. **Leaky ReLU**

   It is a variation of ReLU that allows small, non-zero gradients for negative inputs. Its formula is:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

It is often used in deep neural network for robust performance and it helps to avoid the dying ReLU problem. It is more computationally expensive than ReLU because of the addition parameter.

Optimization Algorithms

Optimization algorithms are used in neural networks to update model parameters (e.g. weights, bias).It plays an important role in accelerating and increasing the efficiency of the learning process by reducing losses. Some optimization algorithms are given below:

1. **Stochastic Gradient Descent(SGD):**
   SDG is an extension of gradient descent algorithm which uses only one training example to compute the gradient and update the parameters at each iteration. It is sensitive to learning rate.
2. **Root Mean Squared Propagation(RMSProp):**
   In RMSProp, the learning rate is adaptively changed or adjusted for each parameter based on the moving average of the squared gradient. This helps the algorithm to converge faster in the presence of noisy gradient. An interesting fact about RMSProp is that it was proposed by Geoffrey Hinton, who is considered as the godfather of AI.
3. **Adam:**
   Adam stands for adaptive moment estimation which combines the benefits of momentum-based gradient descent, adagrad and RMSProp. The learning rate is adjusted based on the moving average of the gradient and the squared gradient, which allows for faster convergence and better performance on non-convex optimization problems. But sometimes it may overfit the model.

Choosing learning rate is crucial for efficient and effective model training. When learning rate is too small, the optimization process progress very slowly. The model makes slight updates to its parameters in each iteration, leading to sluggish convergence and potentially getting stuck in local minimal.

On other hand, when learning rate is excessively high, then it can cause the optimization algorithm to overshoot the optimal values, leading to divergence or oscillations that hinder convergence.

Achieving the appropriate learning rate is essential. A small learning rate might result in vanishing gradients and slow convergence, while a large learning rate may lead to overshooting and instability.