Activation Function

Activation Functions in Neural Networks:
Activation functions are mathematical functions used in neural networks to determine the output of neurons. They "activate" neurons by applying a transformation to the weighted sum of their inputs and biases. This process allows neural networks to make complex decisions and approximations, similar to how biological neurons process signals.

How Activation Functions Work

1. A neuron receives weighted inputs plus a bias(W1X1 + W2X2 + …..+WnXn+b)
2. The weighted sum is passed through an activation function (*f*).
3. The activation function maps this value to a desired range (e.g., sigmoid maps to [0,1]).

Activation Functions used for:

1. **Introduce Non-Linearity**: Non-linear activation functions enable neural networks to approximate complex functions and solve non-linear problems. Without them, networks would behave as simple linear models.
2. **Stabilize Training**: Activation functions can map inputs to known ranges, such as [0,1] or [-1,1], which aids in stabilizing training and output behavior.
3. **Map to Desired Outputs**: For example, sigmoid is ideal for probabilities, and ReLU works well for hidden layers.

Importance of Differentiability

Differentiable activation functions are critical because neural networks learn through backpropagation, which relies on derivatives to adjust weights and biases based on errors. Common issues include:

- **Vanishing Gradients**: Gradients become too small in deeper layers, slowing or halting learning.
- **Exploding Gradients**: Gradients grow excessively, destabilizing training.

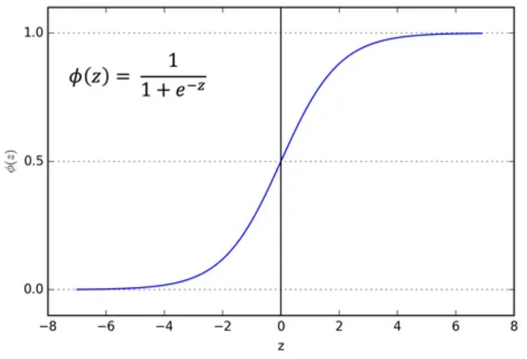
Challenges with Differentiable Functions

- **Vanishing Gradients**: Often seen in sigmoid or tanh functions, where gradients diminish in deeper layers.
- **Exploding Gradients**: May occur in deep networks if gradients grow exponentially during backpropagation.

1. Sigmoid Activation Function:

The sigmoid function is a special form of the logistic function and is usually denoted by sig(x). It is given by:

The graph of sigmoid function is an S-shaped curve as shown by the red line in the graph below:



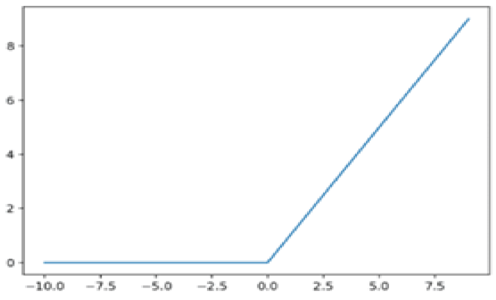
- **Characteristics**:
    - Output range: (0, 1), making it suitable for binary classification.
    - Smooth and differentiable.
- **Use Cases**: Binary logistic regression, output layer in binary classification tasks.
- **Limitations**:
    - Vanishing gradient problem: Gradients become very small for large or small inputs, slowing learning.
    - Non-Zero-Centered Output: Leads to inefficient gradient updates as all outputs have the same sign, causing slower convergence.

2. ReLU (Rectified Linear Unit)

The rectified linear unit (ReLU) or rectifier activation function introduces the property of nonlinearity to a deep learning model and solves the vanishing gradients issue. It interprets the positive part of its argument. It is one of the most popular activation functions in deep learning. ReLU is defined as:

ReLU(x)=max(0,x)

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



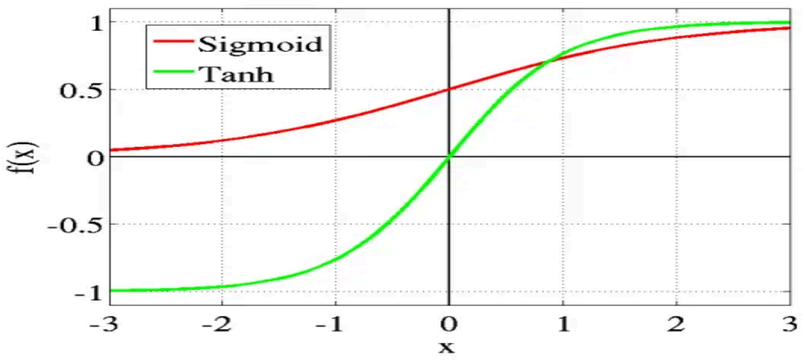
- **Characteristics**:
  - Output range: [0, ∞), allowing efficient computation.
  - Introduces sparsity in neural activations..
- **Use Cases**: Most hidden layers in deep learning due to simplicity and efficiency..
- **Limitations**:
  - Dying ReLU problem: Neurons can become inactive if weights lead to negative outputs.

3. Tanh (Hyperbolic Tangent) Activation Function:

The hyperbolic tangent (Tanh) activation function is defined mathematically as:

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

This function maps the input values to a range between −1-1−1 and 111. It is a scaled and shifted version of the sigmoid function but symmetric about zero, which makes it zero-centered.



- **Characteristics**:
  - Output Range: Produces values between −1-1−1 and 111, which makes it particularly useful for zero-centered data.
  - Non-linear: Enables the modeling of complex data patterns.
  - Smoothing Property: It is smooth and differentiable, facilitating gradient-based optimization

- **Use Cases:**

1. **Hidden Layers:** Useful in hidden layers of neural networks where zero-centered activation is beneficial to avoid biased gradient updates.
2. **Intermediate Layers:** Often used in tasks where the distribution of input data is centered around zero.
3. **Autoencoders:** In autoencoders and other networks requiring symmetric outputs, Tanh can be advantageous.

Limitations:

1. **Vanishing Gradient Problem:**
   - When inputs to neurons are in the saturation regions (∣x∣≫0|x| \gg 0∣x∣≫0), gradients become very small, slowing down the learning process.
   - This issue becomes more prominent in deep networks.
2. **Computationally Expensive:**
   - Requires exponential calculations, which can be slower compared to simpler activation functions like ReLU.
3. **Output Saturation:**

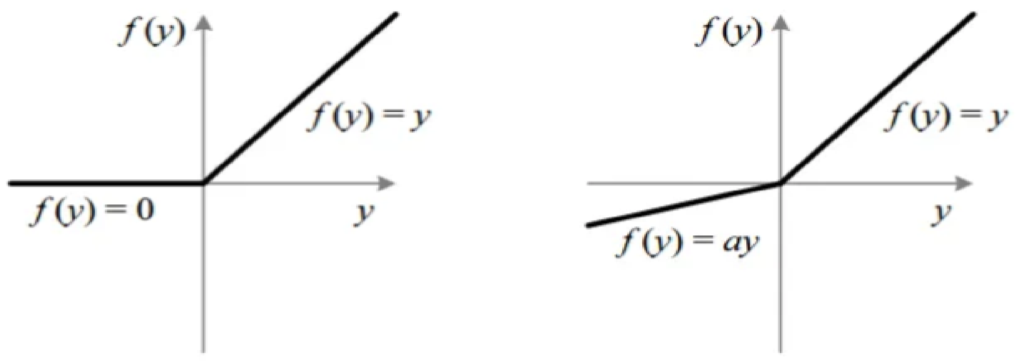
    ○ For large positive or negative inputs, the output becomes almost constant at 111 or −1-1−1, making the gradient close to zero.

4 .**Leaky ReLU Activation Function**

Definition:
Leaky ReLU is a modification of the ReLU activation function that introduces a small, constant slope (α\alphaα) for negative inputs to prevent neurons from "dying." The function is defined as:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \le 0 \end{cases}$$

Where α\alphaα is a small positive constant



Use Cases:

- Deep Neural Networks: Prevents dying neurons and stabilizes training.
- Feature Extraction: Keeps more neurons active for diverse data representation.
- Gradient Flow: Reduces the vanishing gradient issue in deep layers.

Limitations:

- Noisy Updates: Small negative slope may amplify irrelevant data.
- Parameter Tuning: Improper choice of α\alphaα can hinder learning or slow convergence.

- Non-Adaptive Slope: Fixed α\alphaα may not suit all tasks, unlike variants like Parametric ReLU.

Reference :
1 .https://www.superannotate.com/blog/activation-functions-in-neural-networks
2. Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310-316.

Optimization Algorithm:

Optimization algorithms like Gradient Descent, Stochastic Gradient Descent (SGD), and their variants (AdaGrad, RMSprop, Adam, etc.) are the backbone of deep learning model training. They iteratively refine the model's weights and biases to minimize a cost function, ensuring the model makes accurate predictions.

Concepts in Optimization:

1. **Gradient Descent Variants**:
    - **Batch Gradient Descent**: Uses the entire dataset to compute gradients, ensuring stable updates but computationally intensive.
    - **Stochastic Gradient Descent (SGD)**: Updates parameters using one training example at a time, introducing noise for faster convergence but less stability.
    - **Mini-batch Gradient Descent**: Combines benefits of both by using a subset of data, balancing efficiency and stability.
2. **Adaptive Algorithms**:

   - **AdaGrad**: Adjusts learning rates based on past gradients. Effective for sparse data but suffers from diminishing learning rates over time.
   - **RMSprop**: Builds on AdaGrad by using a moving average of squared gradients, avoiding excessive shrinking of learning rates.
   - **Adam (Adaptive Moment Estimation)**: Combines momentum (first moment) and adaptive scaling (second moment), providing robust and efficient updates.
3. **Challenges in Optimization**:
   - **Learning Rate Tuning**: Choosing an optimal learning rate is critical; values too small lead to slow convergence, while large values risk overshooting.
   - **Local Minima and Saddle Points**: In non-convex loss landscapes, optimizers may get stuck, making methods like SGD or Adam with inherent randomness advantageous.
4. **Applications of Optimization Algorithms**:
   - Effective training of deep learning models on large-scale datasets.
   - Online learning where data arrives incrementally.
   - Handling complex, high-dimensional parameter spaces with varying scales.
   -

Reference :

https://neptune.ai/blog/deep-learning-optimization-algorithms

https://www.d2l.ai/chapter_optimization/optimization-intro.html