
Activation functions in Deep learning

Bhupen Sinha

25-07-2024

GROK KERS
AI FOR EVERYONE

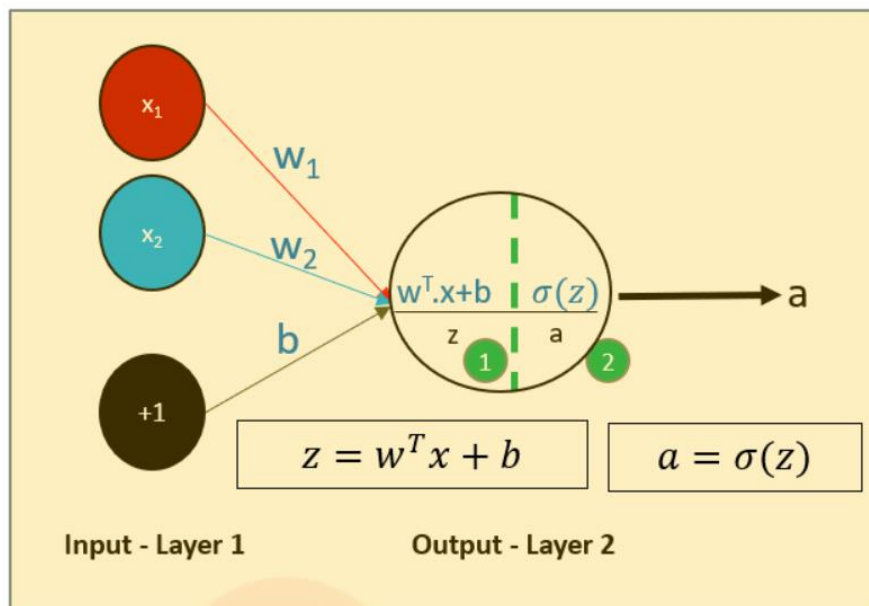
TABLE OF CONTENTS

what are activation functions	3
Synopsis on Using Activation Functions in Neural Networks.....	4
Types of activation functions	6
Heaviside step function.....	6
Sigmoid function.....	8
Tanh Function.....	10
ReLU (Rectified Linear Unit) Function	13
Leaky ReLU: Description.....	15
Guidelines on Choosing Activation Functions	17
Practical Tips	19
Useful Links and references	20

GROKWKERS
AI FOR EVERYONE

WHAT ARE ACTIVATION FUNCTIONS

- Activation functions are mathematical functions applied to the output of neurons in artificial neural networks.
- They introduce `non-linearity` to the network, enabling it to learn complex patterns and relationships in the data.
- they are used `within` the network and/or at the `output` of model



Breakdown:

1. **Inputs ($x_1, x_2, +1$):** Represent input features, including a bias term ($+1$).
2. **Weights (w_1, w_2):** Determine the influence of each input on the output.
3. **Linear Combination ($z = W^T \cdot x + b$):** Calculates a weighted sum of inputs with a bias (b).
4. **Activation ($\sigma(z)$):** Applies the sigmoid function to z , introducing non-linearity.
5. **Output (a):** The final output of the neuron, representing the activation level.

Sigmoid Function:

- Mathematically: $\sigma(z) = 1 / (1 + \exp(-z))$
- Range: 0 to 1
- Shape: S-shaped curve

Purpose:

- Introduces non-linearity, allowing the network to learn complex patterns.
- Can be used for binary classification tasks where the output represents a probability.

SYNOPSIS ON USING ACTIVATION FUNCTIONS IN NEURAL NETWORKS

Model Without Activation Functions

- **Behaviour:** Without activation functions, each layer of the neural network performs a **linear transformation** on the input data. Mathematically, this means the entire network behaves like a single-layer linear model, no matter how many layers it has.
- **Limitations:**
 - **Inability to Learn Complex Patterns:** The network can only learn linear relationships. It cannot capture the complexity and non-linear patterns present in real-world data.
 - **Poor Performance:** As seen in our experiment, a model without activation functions fails to make accurate predictions, as it cannot differentiate between classes effectively.

Model With Activation Functions (ReLU or Tanh)

1. ReLU (Rectified Linear Unit) Activation Function

- **Formula:** The ReLU function outputs the input directly if it is positive; otherwise, it outputs zero.
- **Range:** Outputs range from zero to positive infinity.
- **Characteristics:**
 - **Non-Linearity:** Introduces non-linearity while retaining simplicity.
 - **Sparse Activation:** Activates only a subset of neurons, making the network efficient.
 - **Gradient:** Avoids vanishing gradients, making it suitable for deep networks.
- **Pros:**
 - Efficient computation.
 - Helps in mitigating the vanishing gradient problem.
- **Cons:**
 - Can suffer from the dying ReLU problem, where neurons can sometimes become inactive for all inputs.

2. Tanh (Hyperbolic Tangent) Activation Function

- **Formula:** The Tanh function outputs values that range between -1 and 1, effectively scaling the input.
- **Range:** Outputs range from negative one to positive one.
- **Characteristics:**
 - **Non-Linearity:** Provides non-linear transformation.
 - **Zero-Centered:** Outputs are zero-centered, which helps in gradient descent optimization.
 - **Gradient:** Larger gradients compared to sigmoid, reducing vanishing gradient issues.
- **Pros:**
 - Better gradient flow compared to sigmoid.
 - Zero-centered output helps in balanced learning.
- **Cons:**
 - Still susceptible to the vanishing gradient problem, though less so than sigmoid.

Performance Comparison

- **Without Activation Functions:**
 - **Accuracy:** Poor, as the model cannot learn complex relationships.
 - **Confusion Matrix:** Shows inability to correctly classify the data.
 - **Example:** In the Breast Cancer dataset, the model predicted a single class for all inputs, resulting in an accuracy of 62.28%.
- **With Activation Functions (Tanh in the Example):**
 - **Accuracy:** Significantly improved, as the model can learn non-linear relationships.
 - **Confusion Matrix:** Shows a balanced ability to classify different classes correctly.
 - **Example:** In the Breast Cancer dataset, using tanh activation functions, the model achieved an accuracy of 96.49%.

TYPES OF ACTIVATION FUNCTIONS

HEAVISIDE STEP FUNCTION

The `Heaviside step function`, also known as the `unit step` function or the `step function`, is a mathematical function denoted as $H(x)$.

It has various representations, but the most common one is:

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

In other words,

- the Heaviside step function outputs
 - 0 for negative inputs and
 - 1 for non-negative inputs.

It's a simple yet fundamental function with several uses and benefits:

Uses

- **Indicator Function:** It serves as an indicator function that "turns on" when a certain condition is met (i.e., when $X \geq 0$) and remains "off" otherwise. This property makes it useful for defining piecewise functions or constraints in mathematical expressions.
- **Signals and Control Systems:** In engineering and control theory, the step function is used to model abrupt changes or transitions in signals or systems. It represents an instantaneous change from one state to another, making it valuable for analysing and designing control systems.
- **Impulse Response:** In signal processing, the step function is related to the impulse response of a system. Convolution with the step function can be used to extract or analyze specific components of signals or systems.
- **Modelling Thresholds:** It's commonly used in **machine learning and neural networks** to introduce **non-linearity** and model thresholding behaviour.

For example, in binary classification tasks, the step function can be used as an activation function to determine whether the output of a neuron should be 0 or 1 based on a certain threshold.

Limitations

- **Non-Differentiability:** The Heaviside step function is not differentiable at $x=0$. This poses **challenges** for optimization algorithms that rely on derivatives, such as **gradient descent**, as they cannot be applied directly to functions containing the step function.
- **Discontinuity:** The function has a jump discontinuity at $x=0$. While this property is useful for defining piecewise functions or indicating threshold behaviour, it can **complicate mathematical operations involving integration, differentiation, or convolution**.
- **Imprecise Modelling:** The step function represents an **abrupt transition** from one value to another at a specific threshold. However, real-world phenomena often exhibit more **gradual** or nuanced changes, making the step function overly simplistic for modelling complex systems.
- **Limited Application:** The step function is primarily useful for binary decision-making or indicating the presence or absence of a condition. It may not be suitable for tasks requiring smooth transitions or continuous representations of data.

Summary

- The Heaviside step function has historical significance in early neural network models but is **largely obsolete** in modern deep learning due to its non-differentiability.
- Contemporary activation functions that are smooth and differentiable (such as ReLU, sigmoid, and tanh) have largely replaced the Heaviside step function for training deep neural networks effectively.
- In generative models, alternative methods and smoother approximations are generally preferred to handle binary or discrete outputs in a way that supports gradient-based learning.

SIGMOID FUNCTION

Overview

The sigmoid function is a mathematical function commonly used in neural networks as an activation function. It transforms its input into a value between 0 and 1, making it especially useful for binary classification tasks.

Mathematical Definition

The sigmoid function is defined as:

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

where x is the input to the function, and e is the base of the natural logarithm (approximately 2.718).

Characteristics

1. Output Range:

- The function squashes input values to a range between 0 and 1. As x approaches positive infinity, the output approaches 1. As x approaches negative infinity, the output approaches 0.

2. Shape:

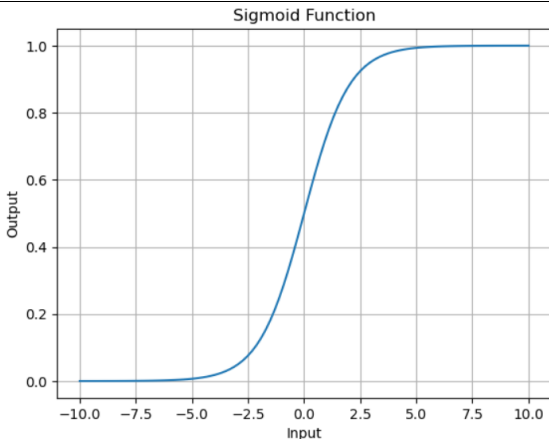
- The sigmoid function has an S-shaped curve (sigmoid curve) that smoothly transitions between 0 and 1. This curve is symmetric around the origin, meaning that the function is the same for positive and negative values of x , but mirrored around 0.

3. Gradient:

- The gradient of the sigmoid function is computed as:

$$\sigma'(x) = \sigma(x) \times (1 - \sigma(x))$$

This means that the gradient is highest near the centre of the curve (where $\sigma(x)$ is close to 0.5) and very small at the extremes (near 0 and 1).

Sigmoid function	Uses
 <p>The graph shows the Sigmoid Function, which maps any real-valued number into the range (0, 1). The x-axis is labeled 'Input' and ranges from -10.0 to 10.0. The y-axis is labeled 'Output' and ranges from 0.0 to 1.0. The curve is S-shaped, passing through (0, 0.5).</p>	<p>Binary Classification: In binary classification tasks, the sigmoid function is commonly used as the activation function in the output layer. It maps the network's raw output to a probability score between 0 and 1, indicating the likelihood of the input belonging to the positive class.</p> <p>Logistic Regression: Sigmoid functions are the core of logistic regression models. They model the probability that an input belongs to a certain class, making them suitable for binary classification tasks.</p> <p>Neural Networks: Historically, the sigmoid function was widely used as an activation function in neural networks. It introduced non-linearity to the network, allowing it to learn complex patterns. However, it has been largely replaced by other activation functions like ReLU due to certain limitations.</p>

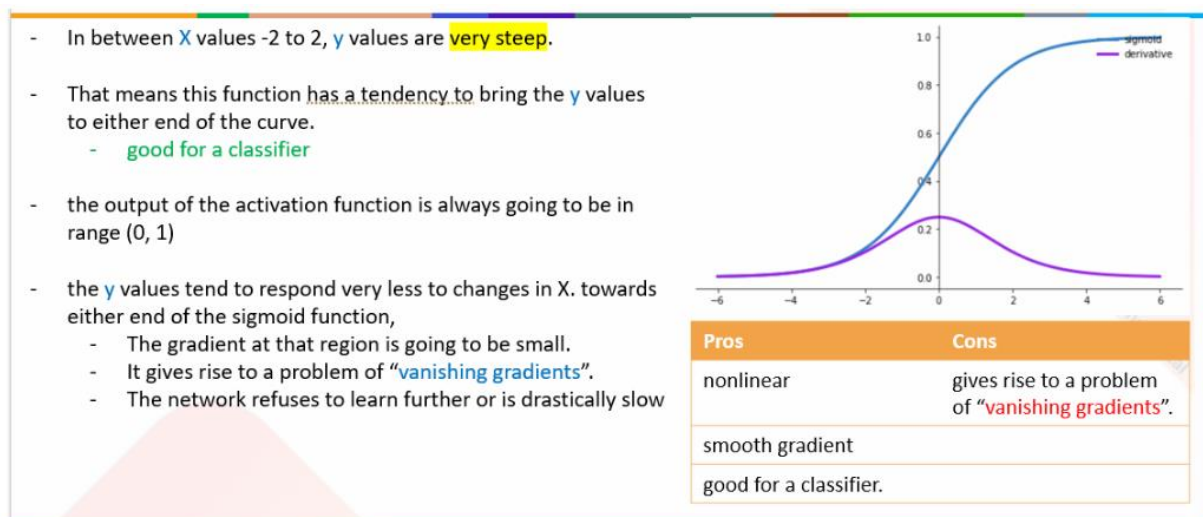
limitations

Vanishing Gradient Problem:

- The sigmoid function can suffer from the vanishing gradient problem, especially for very high or very low input values.
- This occurs because the gradients become very small in these regions, slowing down learning and making it difficult for the network to train effectively.

Output Biases:

- Sigmoid functions tend to output values close to 0.5 for inputs around 0, which can lead to vanishing gradients and slow learning in the early stages of training.



TANH FUNCTION

The tanh (hyperbolic tangent) function is a commonly used activation function in neural networks. It maps input values to outputs ranging from -1 to 1, making it useful for problems where inputs need to be centered around zero.

Mathematical Definition

The tanh function is defined as:

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

where x is the input to the function, and e is the base of the natural logarithm (approximately 2.718).

Characteristics

1. Output Range:

- The function squashes input values to a range between -1 and 1. As x approaches positive infinity, the output approaches 1. As x approaches negative infinity, the output approaches -1.

2. Shape:

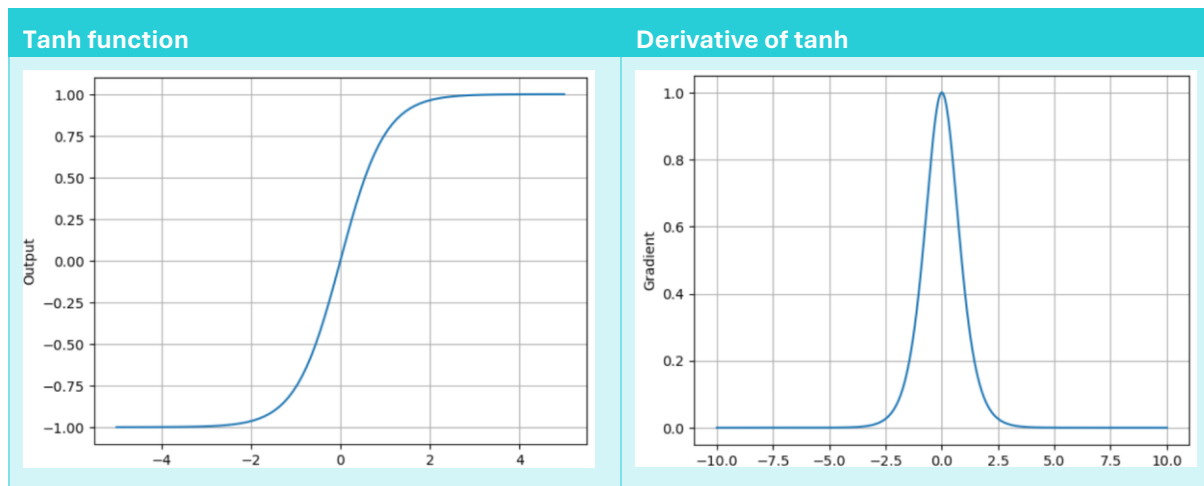
- The tanh function has an S-shaped curve (sigmoid-like curve) that smoothly transitions between -1 and 1. This curve is symmetric around the origin, meaning that the function outputs are centered around zero.

3. Gradient:

- The gradient of the tanh function is computed as:

$$\tanh'(x) = 1 - \tanh(x)^2$$

- This means that the gradient is highest near the center of the curve (where $\tanh(x)$ is close to 0) and very small at the extremes (near -1 and 1).



Properties

1. Non-Linearity:

- The tanh function introduces non-linearity into the network. This non-linearity allows the network to learn complex patterns and representations.

2. Differentiability:

- The tanh function is differentiable, which means it can be used in gradient-based optimization methods like backpropagation. The gradient is essential for updating the weights during training.

3. Zero-Centered Output:

- Unlike the sigmoid function, the tanh function's output is zero-centered, meaning that the mean of the outputs is close to zero. This property can help in gradient descent optimization by balancing the positive and negative values, leading to faster convergence.

4. Vanishing Gradient Problem:

- Like the sigmoid function, the tanh function can suffer from the vanishing gradient problem, especially for very high or very low input values. This occurs because the gradients become very small in these regions, slowing down learning and making it difficult for the network to train effectively.

Use Cases of tanh

1. Hidden Layers in Neural Networks:

- The tanh function is commonly used in the hidden layers of neural networks. Its zero-centered output can help in making the training process more efficient compared to the sigmoid function.

2. Recurrent Neural Networks (RNNs):

- Tanh is often used in RNNs and Long Short-Term Memory (LSTM) networks due to its ability to handle inputs that vary over time and produce outputs that are centered around zero.

3. Binary Classification:

- While not as common as sigmoid for binary classification, tanh can still be used in tasks where the data distribution requires outputs centered around zero.

Summary

- The **tanh** function is an important activation function in neural networks. It maps input values to a range between -1 and 1, introducing non-linearity and enabling the network to learn complex patterns.
- Its **zero-centered output** helps in gradient descent optimization, making it more efficient in certain scenarios compared to the sigmoid function.
- However, **like the sigmoid function**, it can suffer from the **vanishing gradient problem**, which can affect training in deep networks. Despite this limitation, tanh remains a valuable tool in various neural network architectures.

RELU (RECTIFIED LINEAR UNIT) FUNCTION

Overview

The ReLU (Rectified Linear Unit) function is one of the most widely used activation functions in deep learning and neural networks. It introduces non-linearity into the model, which helps the network learn complex patterns and representations.

Mathematical Definition

The ReLU function is defined as:

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

where x is the input to the function.

Characteristics

1. Output Range:

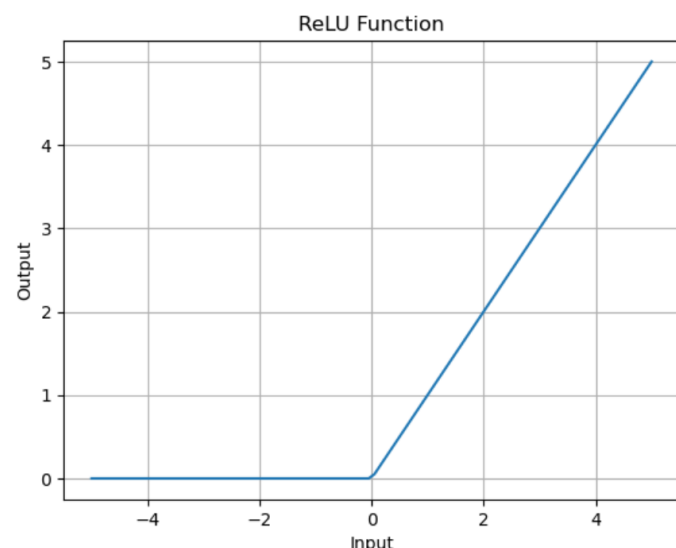
- The ReLU function outputs the input directly if it is positive; otherwise, it outputs zero. Therefore, its range is $[0, \infty)$

2. Shape:

- The ReLU function has a piecewise linear shape, making it simple and efficient to compute. It is linear for positive inputs and zero for negative inputs.

3. Gradient:

- The gradient of the ReLU function is:
 - 1 for $x > 0$
 - 0 for $x \leq 0$
- This piecewise gradient allows for efficient computation during backpropagation.



Properties

1. Non-Linearity:

- ReLU introduces non-linearity into the network, enabling it to learn and model complex data distributions. This non-linearity is crucial for learning intricate patterns.

2. Differentiability:

- While the ReLU function is not differentiable at $x=0$, it is sub differentiable. This means that during training, we can set the gradient at $x=0$ to zero or any value within the subdifferential, typically set to zero.

3. Sparsity:

- ReLU produces sparse activations, meaning that for a large portion of the input, the output will be zero. This sparsity can make the network more efficient by reducing the number of active neurons and speeding up computation.

4. Avoidance of Vanishing Gradient Problem:

- ReLU helps mitigate the vanishing gradient problem that can occur with activation functions like sigmoid and tanh. By having a gradient of 1 for positive inputs, it maintains strong gradients and promotes efficient learning in deep networks.

5. Dying ReLU Problem:

- A potential issue with ReLU is the "dying ReLU" problem, where neurons can become inactive and output zero for all inputs. This occurs if the input to a neuron is consistently negative, causing the gradient to be zero and preventing updates to the weights.
- Variants of ReLU, such as Leaky ReLU and Parametric ReLU, address this issue.

Use Cases

1. Hidden Layers in Deep Neural Networks:

- ReLU is extensively used in the hidden layers of deep neural networks. Its simplicity and efficiency make it a popular choice for a wide range of tasks, including image and speech recognition, natural language processing, and more.

2. Convolutional Neural Networks (CNNs):

- In CNNs, ReLU is commonly used after convolutional and pooling layers to introduce non-linearity and help the network learn hierarchical features.

3. Generative Adversarial Networks (GANs):

- ReLU is used in the generator and discriminator networks of GANs to improve the training stability and efficiency.

LEAKY RELU: DESCRIPTION

The Leaky Rectified Linear Unit (Leaky ReLU) is a type of activation function used in artificial neural networks. It addresses some limitations of the traditional Rectified Linear Unit (ReLU).

Key Properties

1. Definition:

- Leaky ReLU modifies the ReLU function by allowing a small, non-zero gradient when the unit is not active.
- The function is defined as:

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

where α is a small constant (e.g., 0.01).

2. Non-linearity:

- Like ReLU, Leaky ReLU introduces non-linearity into the network, which helps in learning complex patterns.

3. Preventing Dying ReLU:

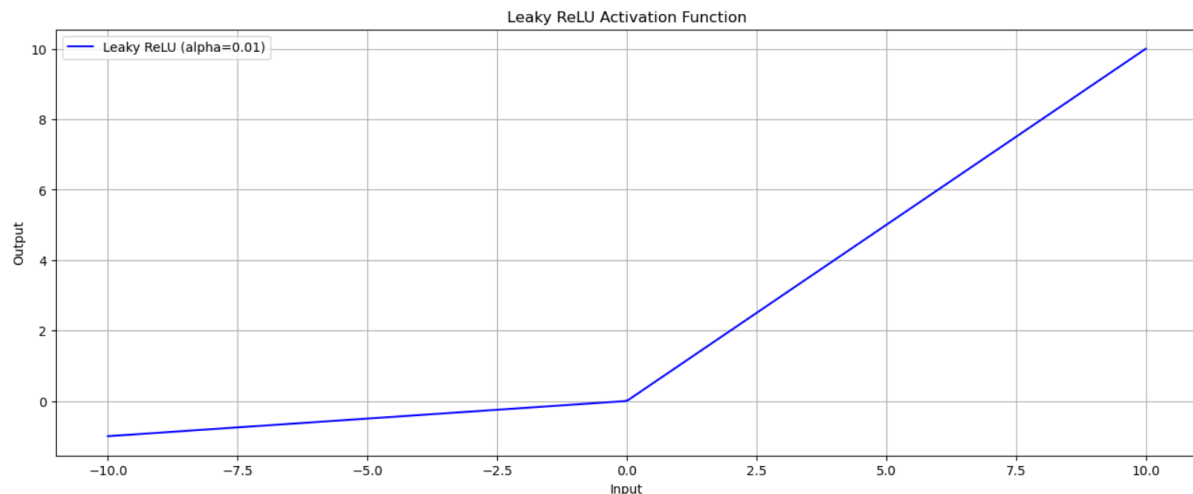
- One of the main advantages of Leaky ReLU over ReLU is that it helps prevent the "dying ReLU" problem, where neurons get stuck during training and only output zero for any input.
- Leaky ReLU ensures a small gradient flow even when the input is negative.

4. Gradient Flow:

- By allowing a small gradient when $x \leq 0$, Leaky ReLU ensures that neurons can still update during backpropagation, even if their outputs are negative.

5. Parameter α :

- The parameter α is a small positive value, usually much less than 1 (e.g., 0.01). It can be fixed or treated as a hyperparameter to be tuned during training.



Use Cases

1. Deep Learning Models:

- Leaky ReLU is commonly used in deep neural networks, including Convolutional Neural Networks (CNNs) and Fully Connected Networks (FCNs), to improve learning dynamics and avoid issues associated with ReLU.

2. Computer Vision:

- In image recognition and processing tasks, Leaky ReLU can help stabilize training and improve performance due to its ability to handle negative inputs better than ReLU.

3. Natural Language Processing (NLP):

- In NLP tasks, such as text classification and language modeling, Leaky ReLU can be used to maintain gradient flow and improve convergence rates.

4. Generative Models:

- Leaky ReLU is often employed in Generative Adversarial Networks (GANs) to enhance the stability of the training process.

GUIDELINES ON CHOOSING ACTIVATION FUNCTIONS

Choosing the right activation function is crucial for the performance and convergence of neural networks. Here are some general guidelines to help you select the appropriate activation function for different layers and types of networks:

1. ReLU (Rectified Linear Unit)

- **Pros:**
 - Simple and computationally efficient.
 - Helps mitigate the vanishing gradient problem.
 - Sparsity: Encourages sparsity by zeroing out negative values.
- **Cons:**
 - Can suffer from the "dying ReLU" problem, where neurons get stuck with zero output.
- **Use Cases:**
 - Default choice for hidden layers in most neural networks, especially in Convolutional Neural Networks (CNNs).

2. Leaky ReLU

- **Pros:**
 - Prevents the dying ReLU problem by allowing a small, non-zero gradient when the input is negative.
- **Cons:**
 - Additional hyperparameter (alpha) to tune.
- **Use Cases:**
 - When experiencing dying ReLUs with standard ReLU. Useful in deeper networks.

3. Parametric ReLU (PReLU)

- **Pros:**
 - Similar to Leaky ReLU but with learnable alpha parameter.
 - Can adapt during training to optimize performance.
- **Cons:**
 - Additional parameter increases computational complexity.
- **Use Cases:**

- Networks where fine-tuning of the negative slope is beneficial, like in deep CNNs.

4. Sigmoid

- **Pros:**
 - Outputs values in the range (0, 1), suitable for probability estimation.
- **Cons:**
 - Prone to vanishing gradient problem.
 - Can saturate and kill gradients during backpropagation.
- **Use Cases:**
 - Output layer for binary classification problems.

5. Tanh (Hyperbolic Tangent)

- **Pros:**
 - Zero-centered output which can help with centering the data.
 - Better than sigmoid in practice due to its range (-1, 1).
- **Cons:**
 - Still susceptible to the vanishing gradient problem.
- **Use Cases:**
 - Hidden layers of networks where inputs may be highly varied.

6. Softmax

- **Pros:**
 - Converts logits into probabilities, useful for multiclass classification.
- **Cons:**
 - Not suitable for hidden layers due to normalization of outputs.
- **Use Cases:**
 - Output layer in multiclass classification problems.

PRACTICAL TIPS

1. Start Simple:

- Begin with ReLU for hidden layers and Softmax/Sigmoid for output layers based on the task.

2. Experimentation:

- If encountering issues like dying neurons or slow convergence, experiment with variants like Leaky ReLU, ELU, or newer functions like Swish.

3. Consider the Task:

- Choose activation functions that align with the specific requirements of your task (e.g., probability outputs for classification).

4. Keep an Eye on Computational Costs:

- Balance the benefits of more complex activation functions with their computational overhead.

GROKWORKERS
AI FOR EVERYONE

USEFUL LINKS AND REFERENCES

-
-

GROKWKERS
AI FOR EVERYONE

INDEX

No index entries found.

GROKWKERS
AI FOR EVERYONE