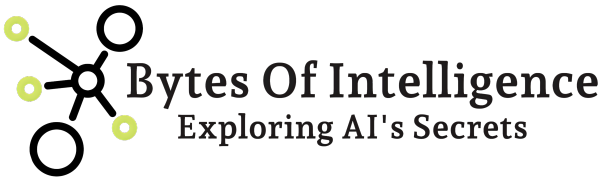

Activation Functions With Math

Email ahammadmejbah@gmail.com | GitHub @BytesOfIntelligences | LinkedIn Mejbah Ahammad
Website Bytes of Intelligence | YouTube BytesofIntelligence | ResearchGate Mejbah Ahammad
Phone +8801874603631 | Hackerrank ahammadmejbah

Sigmoid Activation Function

Introduction to the Sigmoid Activation Function

The sigmoid function, commonly represented as $\sigma(x)$, plays a critical role in neural networks, particularly in binary classification problems. It converts a real-valued number into a probability score between 0 and 1, making it especially useful for models where outputs are interpreted as probabilities.

Mathematical Definition:

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

Step-by-Step Derivation of the Sigmoid Function

Understanding the mathematical derivation of the sigmoid function helps in grasping its behavior:

1. **Exponential Component:** The function includes an exponential component which we calculate as follows:

   e^{-x}, where x is the input to the neuron.

2. **Formulation of the Function:** We then formulate the function by adding 1 to the exponential component:

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

Simplified Form: This can be further simplified for computational efficiency:

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

Derivative of the Sigmoid Function

The derivative of the sigmoid function is crucial for the backpropagation algorithm used in training neural networks. Here is how it is computed:

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

This shows that the derivative of the sigmoid function can be expressed in terms of the function itself, which is a significant property because it simplifies the computation during the learning process.

Example Calculation of the Sigmoid Function

Let's calculate the sigmoid function for an input value of 2:

1. **Calculate the exponential term:** First, we compute the exponential part for $x = 2$.

   $$e^{-2} = \frac{1}{e^2} \approx 0.1353$$

2. **Compute the denominator:** Next, add 1 to the result from the exponential calculation:

   $$1 + e^{-2} = 1 + 0.1353 \approx 1.1353$$

3. **Final sigmoid output:** Finally, compute the output of the sigmoid function:

   $$\sigma(2) = \frac{1}{1.1353} \approx 0.8808$$

Applications and Limitations

The sigmoid function is historically important for binary classifications in neural networks. However, it has limitations like vanishing gradients, which can impede the learning of neural networks, particularly in deep networks.

Conclusion

The sigmoid function's ability to output values between 0 and 1, its mathematical properties, and its implications in learning dynamics make it a fundamental concept in neural network design and analysis. This detailed exploration provides an understanding of its calculations and applications.

Tanh Activation Function

Introduction to the Tanh Function

The hyperbolic tangent function, or tanh, is a sigmoidal (S-shaped), continuous, and differentiable function that maps real-valued inputs to the range (-1, 1). It is often preferred over the logistic sigmoid function because it centers the output around zero, which improves the efficiency of backpropagation in neural networks.

Mathematical Definition

The tanh function is defined as:

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

Derivation of the Tanh Function

To understand the tanh function, let's break down its derivation:

1. **Exponential Functions:** The function utilizes the difference and sum of the exponential function and its inverse:

   $e^x \text{and} e^{-x}$

2. **Numerator:** Calculate the difference between e^x and e^{-x}:

   $e^x - e^{-x}$

3. **Denominator:** Calculate the sum of e^x and e^{-x}:

   $e^x + e^{-x}$

4. **Combining Terms:** Combine the results of the numerator and denominator to form the tanh function:

   $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

Example Calculation

Let's compute the tanh function for an input value of 1:

1. **Calculate the exponentials:** Compute e^1 and e^{-1}.

   $e^1 \approx 2.71828 \, e^{-1} \approx 0.36788$

2. **Calculate numerator and denominator:**

   Numerator: $2.71828 - 0.36788 = 2.3504$
   Denominator: $2.71828 + 0.36788 = 3.08616$

3. **Final tanh output:**

$$\tanh(1) = \frac{2.3504}{3.08616} \approx 0.7616$$

Applications and Benefits

The tanh function is extensively used in neural networks where the output needs to be normalized around zero, thus helping gradients flow better during the training process. It is particularly effective in deep neural networks and recurrent neural networks (RNNs).

Conclusion

The tanh function, with its zero-centered output, is an essential activation function in neural networks. This detailed breakdown provides a comprehensive understanding of its mathematical basis and practical applications.

ReLU Activation Function

Introduction to the ReLU Function

The Rectified Linear Unit (ReLU) activation function is a piecewise linear function that outputs the input directly if it is positive; otherwise, it outputs zero. It has become the default activation function for many types of neural networks because it allows models to converge faster and perform better by solving the vanishing gradient problem inherent in traditional activation functions like sigmoid or tanh.

Mathematical Definition

The ReLU function is defined as:

$$f(x) = \max(0, x)$$

Derivation and Properties

Unlike sigmoid and tanh, ReLU does not have a complex mathematical derivation. Its definition is straightforward:

If $x \geq 0$, then $f(x) = x$. If $x < 0$, then $f(x) = 0$.

This simplicity leads to its two main properties:

- **Sparse Activation:** In practical terms, ReLU will output zero for any negative input, leading to sparse activations in neural network layers.
- **Nonlinear:** Although ReLU is a linear function when $x \geq 0$, it introduces a nonlinearity at $x = 0$ which makes it a nonlinear function overall (which is a critical property for learning complex patterns in data).

Example Calculation

Let's calculate the ReLU function for two input values, -3 and 5:

- For $x = -3$: $f(-3) = \max(0, -3) = 0$
- For $x = 5$: $f(5) = \max(0, 5) = 5$

Applications and Benefits

ReLU is extensively used in hidden layers of neural networks due to its several benefits:

- **Faster Convergence:** ReLU often converges much faster than sigmoid or tanh during training because it does not saturate in the positive range.
- **Reduced Likelihood of Vanishing Gradient:** Since the derivative of ReLU is 1 for all positive values and 0 otherwise, it reduces the problem of vanishing gradients encountered with sigmoid and tanh in deep networks.

Conclusion

ReLU has proven to be an effective activation function that improves neural network training speed and performance across many tasks and architectures. Its simple max function introduces the desired nonlinearity while allowing the network to remain computationally efficient.

Leaky ReLU Activation Function

Introduction to the Leaky ReLU Function

The Leaky Rectified Linear Unit (Leaky ReLU) is a variant of the standard ReLU activation function used in neural networks to help prevent what is known as the dying ReLU problem. This issue arises when neurons effectively die, outputting 0 for all inputs and thus, no longer adapting during further training because the gradient at 0 is zero.

Mathematical Definition

The Leaky ReLU function is defined as:

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

Where α is a small coefficient (usually a small, positive number such as 0.01) that provides a small gradient when the unit is inactive (i.e., $x < 0$).

Derivation and Properties

Leaky ReLU modifies the standard ReLU by allowing a small, negative slope when the input is less than zero. This change helps maintain a gradient flow through the network, which can be critical during the training process:

- **Non-zero Gradient:** By allowing a non-zero gradient when $x < 0$, Leaky ReLU mitigates the dying neurons problem of standard ReLU.
- **Computational Efficiency:** Like ReLU, Leaky ReLU remains computationally efficient, making it suitable for deep learning models that require fast computation.

Example Calculation

Let's calculate the Leaky ReLU function for two input values, -3 and 5, with $\alpha = 0.01$:

- For $x = -3$: $f(-3) = 0.01 \times (-3) = -0.03$
- For $x = 5$: $f(5) = 5$

Applications and Benefits

Leaky ReLU is especially useful in scenarios where model reliability is critical, and dead neurons would hinder performance:

- **Robustness:** Leaky ReLU improves the robustness of the model by maintaining active gradient flow across all neurons.
- **Versatility:** It can be used in almost any neural network model where ReLU is applicable, providing a simple yet effective way to improve model performance.

Conclusion

The Leaky ReLU function provides an effective solution to the limitations of the ReLU function by preventing neurons from dying during training. Its simple modification can lead to significant improvements in learning dynamics and model performance in deep learning architectures.

Exponential Linear Unit (ELU) Activation Function

Introduction to the ELU Function

The Exponential Linear Unit (ELU) is an activation function for neural networks that improves on the properties of the rectified linear unit (ReLU) by offering a smoother curve when the input values are negative. This function was designed to speed up learning in neural networks through faster and more stable convergence.

Mathematical Definition

The ELU function is defined as follows:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{if } x < 0 \end{cases}$$

Where α is a hyperparameter, typically chosen to be a small positive number.

Derivation and Properties

The ELU activation function reduces the vanishing gradient effect by introducing an exponential component for negative inputs, which unlike ReLU's zero slope, offers a non-zero gradient, allowing for continued learning during backpropagation:

- **Continuity:** ELU is continuous across all points, including $x = 0$, which helps in reducing training time and improving network predictions.
- **Non-zero Gradient for Negative Inputs:** The use of an exponential function for negative inputs ($\alpha(\exp(x) - 1)$) provides a non-zero gradient, thus addressing the dying neuron problem seen with ReLU.

Example Calculation

Let's compute the ELU function for two input values, -1 and 2, assuming $\alpha = 1$:

- For $x = -1$: $f(-1) = 1(\exp(-1) - 1) \approx -0.6321$
- For $x = 2$: $f(2) = 2$

Applications and Benefits

ELU can be particularly beneficial in deeper neural networks, where vanishing gradients can severely impede the learning process:

- **Improved Learning Characteristics:** By allowing small gradients when the input is negative, ELU helps maintain active and effective weight updates across more layers of the network.

- **Higher Robustness:** ELU reduces the problem of gradients nearing zero, contributing to more stable activations across updates and layers.

Conclusion

The ELU function is a powerful tool in the neural network activation function arsenal, offering the benefits of ReLU and its variants while further mitigating some of their inherent drawbacks. It provides an effective balance between linear and non-linear characteristics, facilitating faster convergence and potentially better performance on complex tasks.

Softplus Activation Function

Introduction to the Softplus Function

The Softplus activation function is a non-linear function used in neural networks to introduce a gradual switching mechanism that is smoother than the ReLU function. The smoothness of Softplus allows it to approximate ReLU while maintaining differentiability across all input values, unlike ReLU which is non-differentiable at zero.

Mathematical Definition

The Softplus function is defined as:

$$f(x) = \log(1 + e^x)$$

This function gradually approaches zero as x approaches negative infinity and behaves like a linear function as x approaches positive infinity, mimicking the ReLU function in the positive domain.

Derivation and Properties

Deriving the Softplus function involves understanding its transformation from the exponential function:

- The Softplus function is essentially the smooth approximation of the ReLU function.
- It introduces the logistic function's integral, which sums the area under the curve, resulting in a continuously increasing function that never quite reaches zero, maintaining a gradient that is always positive.

Example Calculation

Let's calculate the Softplus function for two input values, -1 and 3:

- For $x = -1$: $f(-1) = \log(1 + e^{-1}) \approx \log(1 + 0.3679) \approx 0.3133$
- For $x = 3$: $f(3) = \log(1 + e^{3}) \approx \log(1 + 20.085) \approx 3.0486$

Applications and Benefits

Softplus is used in scenarios where smooth, gradual activation is required:

- **Gradient-Based Optimization:** The differentiability of Softplus across all points makes it suitable for gradient-based optimization methods that rely on the computation of derivatives.
- **Probabilistic Models:** Softplus can be used in log probability models because of its relationship to the logistic sigmoid, which models probabilities.

Conclusion

The Softplus function provides a smooth and continuously differentiable alternative to ReLU, making it ideal for applications requiring smooth activations and gradients. Its mathematical characteristics facilitate effective training in deep neural networks where maintaining gradient flow is crucial.

Swish Activation Function

Introduction to the Swish Function

The Swish activation function is a relatively newer function that has been proposed as an alternative to the traditional activation functions like ReLU. It was discovered through automated machine learning techniques aimed at searching activation functions that might improve learning performance in neural networks.

Mathematical Definition

The Swish function is defined as:

$$f(x) = x \cdot \sigma(\beta x)$$

Where $\sigma(x)$ is the sigmoid function, and β is a parameter that can either be a constant or a trainable parameter. When $\beta = 1$, it simplifies to $x \cdot \sigma(x)$.

Properties and Derivation

The Swish function combines the properties of sigmoid and ReLU, aiming to maintain the non-linearity of sigmoid with the bounded-zero property of ReLU. Its properties include:

- **Smoothness:** Swish is smooth everywhere, which facilitates the optimization algorithm's ability to find minimums in the loss landscape.
- **Non-monotonicity:** Unlike ReLU, Swish is non-monotonic which has been suggested to help neural networks learn more diverse functions.
- **Adaptiveness:** With β as a trainable parameter, Swish can adapt during training to better suit the specific dataset.

Example Calculation

Let's calculate the Swish function for two input values, -1 and 3, assuming $\beta = 1$:

- For $x = -1$: $f(-1) = -1 \cdot \sigma(-1) = -1 \cdot \frac{1}{1 + e^{-(-1)}} \approx -1 \cdot 0.2689 \approx -0.2689$
- For $x = 3$: $f(3) = 3 \cdot \sigma(3) = 3 \cdot \frac{1}{1 + e^{-3}} \approx 3 \cdot 0.9526 \approx 2.8578$

Applications and Benefits

Swish is often used in scenarios where deep learning models benefit from both the properties of ReLU and the smoothness of sigmoid. It has been shown to improve performance in a variety of deep learning tasks:

- **Deep Networks:** Particularly useful in very deep networks where vanishing gradients can be an issue.
- **Dynamic Ranges:** Useful in tasks that benefit from a dynamic range of activation due to its non-monotonic nature.

Conclusion

The Swish activation function represents a significant advancement in activation function design, combining elements of both ReLU and sigmoid to potentially offer better performance in certain neural network architectures.

Parametric Rectified Linear Unit (PReLU) Activation Function

Introduction to the PReLU Function

The Parametric Rectified Linear Unit (PReLU) improves upon the ideas of the ReLU and Leaky ReLU by allowing the negative slope to be learned during the training process. This adaptability can lead to better performance in practice by fine-tuning the activation function to specific tasks.

Mathematical Definition

The PReLU function is defined as follows:

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

Where α is a small coefficient that is learned during training. Unlike Leaky ReLU, α is not a hyperparameter but a parameter that adapts to the data.

Properties and Derivation

PReLU retains some of the beneficial properties of ReLU while adding the flexibility of adapting the function based on the training data:

- **Adaptiveness:** The coefficient α is learned, potentially leading to a customized activation function that can optimize performance for specific datasets or tasks.
- **Non-zero Gradient for Negative Inputs:** Like Leaky ReLU, PReLU provides a gradient for negative inputs, which helps mitigate the dying neuron problem in neural networks.

Example Calculation

Let's calculate the PReLU function for two input values, -2 and 3, assuming a learned $\alpha = 0.1$:

- For $x = -2$: $f(-2) = 0.1 \times (-2) = -0.2$
- For $x = 3$: $f(3) = 3$

Applications and Benefits

PReLU is particularly useful in deep learning architectures where the ability to learn activation function parameters can lead to significant improvements in model accuracy and training dynamics:

- **Deep Networks:** PReLU is effective in very deep networks, particularly where preventing neurons from dying is crucial to maintaining a robust learning process.
- **Customizable Activation:** The learning aspect of α allows PReLU to adapt to different types of data and tasks more effectively than fixed-parameter activation functions.

Conclusion

The Parametric Rectified Linear Unit (PReLU) represents a significant advancement in activation function technology, offering customized activation dynamics that can optimize neural network training and performance.