Activation Functions

Name: Sajid Ibna Mahbub

ID: 20-42109-1

Introduction to Activation Function:

An activation function is a mathematical function used in artificial neural networks (ANNs) to introduce non-linearity into the output of a neuron or a layer of neurons. It takes in a weighted sum of the inputs and biases of a neuron and applies a transformation to produce the neuron's output.

The purpose of an activation function is to introduce non-linearity into the model, which allows it to learn more complex relationships between the inputs and outputs. Without non-linearity, the neural network would essentially be limited to representing linear functions, which are not suitable for many real-world problems. The second step of the figure is where the activation function is used in an artificial neuron.

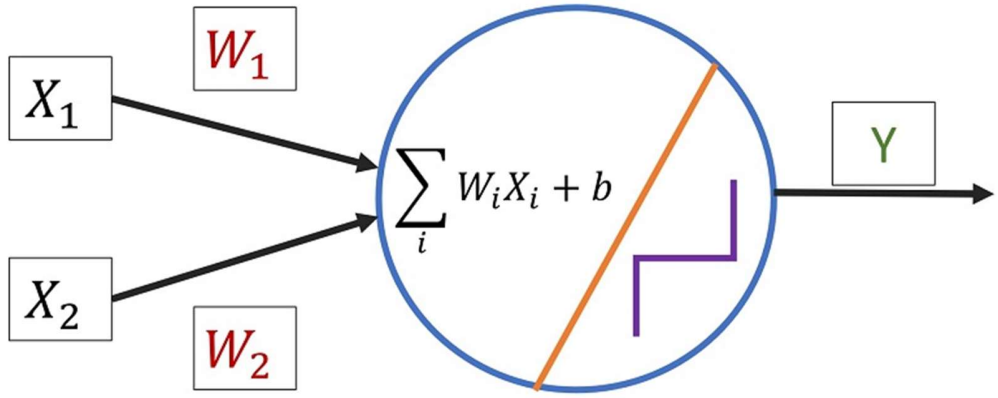


Fig: Perceptron

Some advantages of activation functions are mentioned below,

- ✓ **Introducing non-linearity:** One of the primary advantages of activation functions is that they introduce non-linearity into the model. This allows the neural network to learn complex relationships between the input and output, which is essential for solving many real-world problems.

- ✓ **Enabling deeper neural networks:** Activation functions also allow for the creation of deeper neural networks. Without non-linear activation functions, a deep neural network would behave like a single layer perceptron, limiting its ability to capture complex patterns in the data.

- ✓ **Enhancing model performance:** Different activation functions can be used depending on the type of problem being solved. This can enhance the model's performance and allow it to learn more efficiently.

- ✓ **Preventing overfitting:** Activation functions can also be used to prevent overfitting, which occurs when the model becomes too complex and fits the training data too closely. Certain activation functions, such as Dropout and L1/L2 regularization, can help prevent overfitting.

- ✓ **Interpretability:** Some activation functions, such as the Sigmoid and Tanh functions, produce outputs that can be interpreted as probabilities or scaled values between -1 and 1. This can make it easier to understand the behavior of the model and the relationship between the input and output.

Types of Activation Function:

There are multiple types of activation functions available. Such as,

- Step Function
- Sigmoid Function
- TanH Function
- Relu Function
- Elu Function
- Selu Function

  And many more.

Now let's see the working process and descriptions, advantages and disadvantages of mentioned activation functions,

1. Step

The Step activation function is a simple binary function that produces a value of 1 if the input is greater than or equal to a threshold value, and 0 otherwise. It is also known as the Heaviside step function or unit step function.

Mathematically, the step function can be defined as:

f(x) = 0, if x < 0

f(x) = 1, if x >= 0

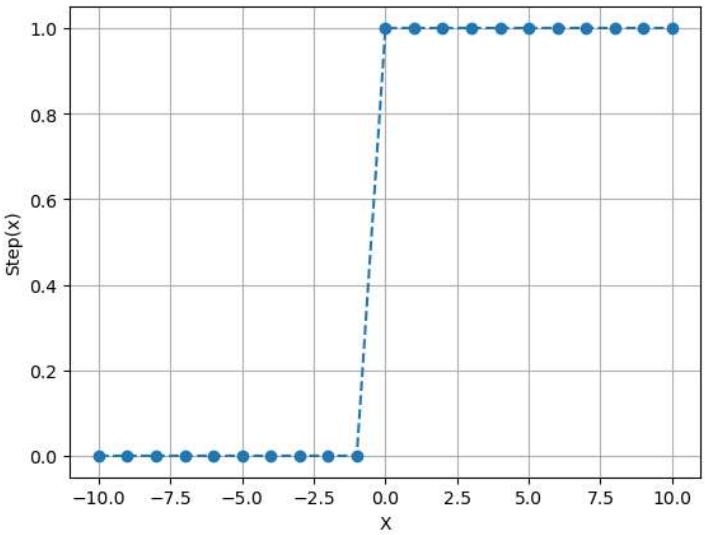
The step function is often used in binary classification problems where the output is either 0 or 1, such as in logistic regression. It can also be used as an activation function in neural networks, although its use is limited due to its discontinuous nature.

A sigmoid activation function is defined as:

Step Activation Function

```
y = np.zeros(len(x))
y[x>=0] = 1

plot_graph(y, "Step(x)")
```

Resulted Graph:



The Step activation function has advantages in its simplicity and interpretability and is useful for problems that require a binary output. However, its discontinuous nature makes it difficult to use in gradient-based optimization algorithms like backpropagation. It also produces a non-smooth output, which is less suitable for regression problems. The threshold value needs to be carefully chosen, as it highly affects the output of the function.

Usage:

The step function is commonly used in binary classification problems where the goal is to predict one of two classes, such as spam or non-spam email. In this case, the threshold value is usually set to 0.5, and the output is either 0 (non-spam) or 1 (spam).

Sigmoid

The sigmoid activation function is a type of mathematical function that maps any input value to a value between 0 and 1. It is commonly used as an activation function in neural networks and logistic regression models. The sigmoid function works by taking the weighted sum of the inputs and applying the sigmoid function to the result. The output of the sigmoid function is a value between 0 and 1, which can be interpreted as the probability of the input belonging to a certain class or category.

The mathematical expression of the Sigmoid activation function is:

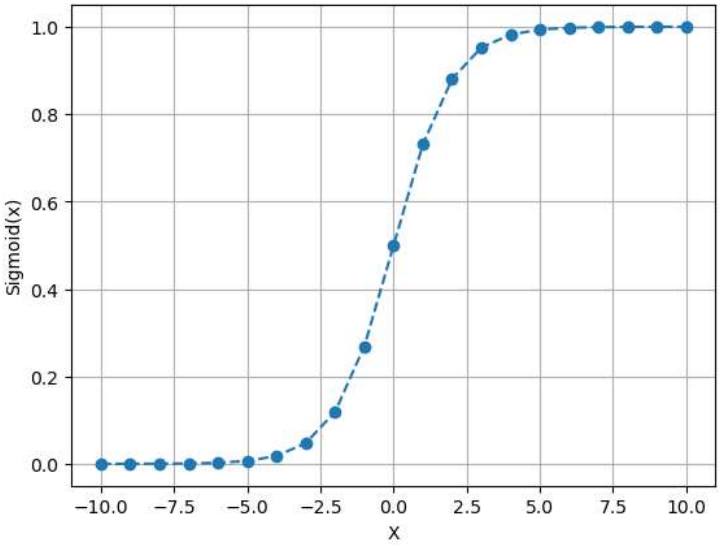
$\sigma(z) = 1 / (1 + e^{(-z)})$

where z is the input to the function, and $\sigma(z)$ is the output, which is a value between 0 and 1. The function maps any real-valued number to a value between 0 and 1, which can be interpreted as a probability.

Sigmoid activation function is defined as:

```
y = 1/(1 + np.exp(-x))

plot_graph(y, "Sigmoid(x)")
```

Resulted Graph:



The sigmoid activation function has the following advantages: it produces output values between 0 and 1, which can be useful in certain types of neural networks and is easy to understand and interpret. However, there are several disadvantages associated with the sigmoid function: it is prone to the vanishing gradient problem, which can make it difficult for deep neural networks to learn complex relationships in the data, it can suffer from saturation at the extremes of its range, which can slow down learning, it is not zero-centered, which can make it harder for the network to converge during training, and it requires the computation of exponentials, which can be computationally expensive. Overall, while the sigmoid function has some advantages, it is not commonly used as an activation function in modern neural networks due to its limitations and the availability of more effective alternatives like the ReLU or its variants.

Usage:

In sentiment analysis, where the goal is to predict the sentiment of a text as positive or negative. The sigmoid function can be used as the activation function in a neural network for this task. The input to the model would be the text, and the output would be a probability between 0 and 1, with values closer to 1 indicating a high probability of the text having a positive sentiment, and values closer to 0 indicating a high probability of the text having a negative sentiment.

2. Tanh

Tanh, or hyperbolic tangent, is an activation function commonly used in neural networks. It is a mathematical function that maps input values to output values between -1 and 1.

Tanh works by taking an input value and applying a transformation that maps it to a value between -1 and 1. Specifically, the function takes the exponential function of the input, subtracts the exponential function of the negative input, and divides this difference by the sum of the exponential function of the input and the exponential function of the negative input.

The mathematical expression of the Tanh activation function is:

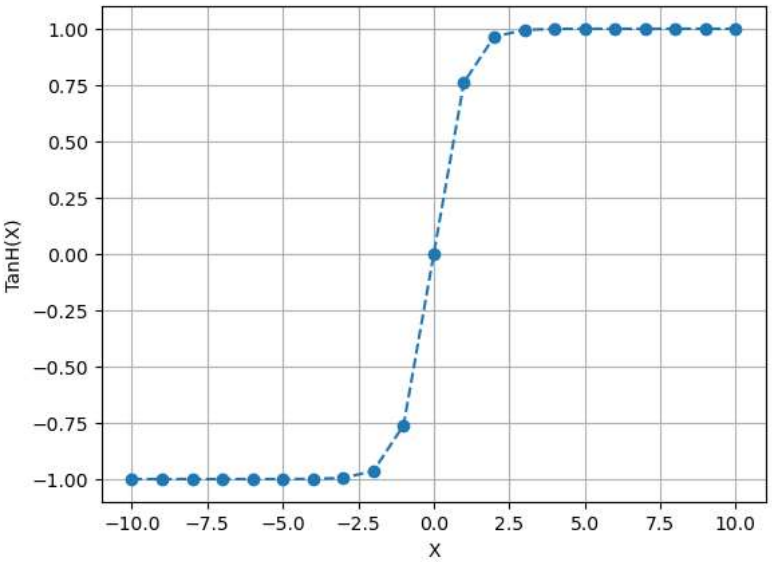
f(x) = (e^x - e^-x) / (e^x + e^-x)

where e is Euler's number (approximately 2.71828) and x is the input to the function. The output of the function f(x) is a value between -1 and 1.

Tanh activation function is defined as:

```
y = (np.exp(2*x) - 1) / (np.exp(2*x) + 1)
plot_graph(y, "TanH(X)")
```

Resulted Graph:



Tanh activation function produces output values between -1 and 1, similar to the sigmoid function. It is non-linear and can model complex relationships between input and output variables. Tanh has a derivative that makes it useful in gradient-based optimization algorithms like backpropagation. However, Tanh is prone to the vanishing gradient problem, which can make it difficult for deep neural networks to learn complex relationships. Like the sigmoid function, Tanh can suffer from saturation at the extremes of its range, which can slow down learning. Additionally, the computation of exponentials required for Tanh can be computationally expensive. In summary, Tanh is a non-linear activation function that produces output values between -1 and 1, making it

useful in certain types of neural networks. However, it is prone to the vanishing gradient problem and saturation, and can be computationally expensive to compute.

Usage:

Example of where the tanh function is used is in speech recognition systems. In speech recognition, the goal is to convert spoken language into text. The input to the model is a speech signal, and the output is a probability distribution over the possible phonemes or words. The tanh function is often used as the activation function in the hidden layers of the neural network used for speech recognition. The neural network takes the speech signal as input and applies a series of transformations using the tanh function, allowing it to learn complex patterns in the speech signal.

3. Relu

The rectified linear activation function, often known as ReLU, is a non-linear or piecewise linear function that outputs the input directly if it is positive and zeroes otherwise. It is now the most prevalent activation function in neural networks, particularly in Convolutional Neural Networks (CNNs) and Multilayer perceptrons. Due to its simplicity, computational efficiency, and ability to minimize the vanishing gradient problem, the ReLU activation function has become increasingly popular in deep learning. The vanishing gradient problem occurs when gradients become extremely small during backpropagation, making it challenging to update the network's weights. ReLU has a non-zero derivative for positive inputs, which facilitates the passage of gradients across the network.

Mathematically, the ReLU function can be expressed as:

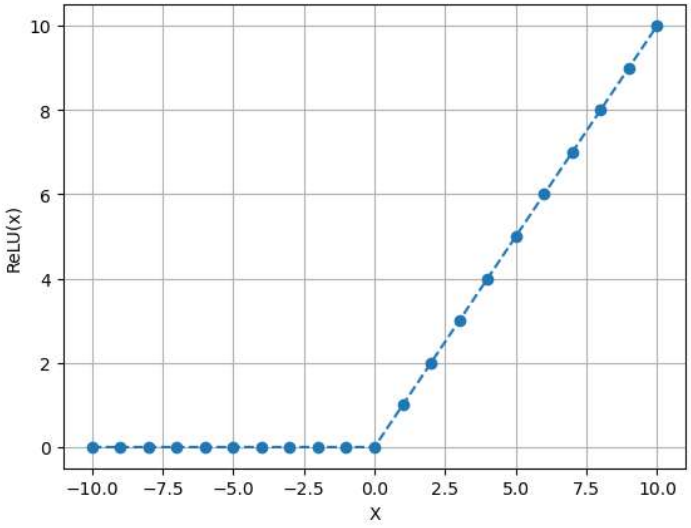
$f(x) = \max(0, x)$ [where x is the input to the function and f(x) is the output]

When a neuron receives an input, it multiplies each input value by a weight, adds a bias term, and passes the result through an activation function. The output of the activation function is then passed to the next layer of the neural network. The ReLU function is typically used as the activation function in the hidden layers of a neural network. To compute the ReLU function, we first take the input value x and compare it to zero. If x is greater than or equal to zero, then the output f(x) is equal to x. If x is negative, then the output f(x) is equal to zero.

ReLU activation function is defined as:

```
y = np.maximum(0, x)

plot_graph(y, "ReLU(x)")
```

Resulted Graph:



The ReLU function is a non-linear function that introduces non-linearity to the output of a neural network. This non-linearity is important for learning complex patterns and relationships between inputs and outputs. Additionally, ReLU has a non-zero derivative for positive inputs, which means that gradients can flow through the network more easily and help mitigate the vanishing gradient problem.

In neural networks, the Rectified Linear Unit (ReLU) is a prominent activation function. ReLU's benefits include computing efficiency, non-linearity, sparsity, and the mitigation of the vanishing gradient issue. ReLU is computationally efficient and simple to implement in either hardware or software. ReLU is non-linear, allowing the network to learn complicated input-output correlations. ReLU is capable of producing sparse representations, which can aid in reducing computation and preventing overfitting. ReLU provides a non-zero derivative for positive inputs, thus gradients can flow through the network more easily, minimizing the problem of disappearing gradients. There are, however, possible drawbacks to utilizing the ReLU function, such as "dead" neurons, unbounded output, unsuitability for some jobs, and sensitivity to initialization. ReLU might result in "dead" neurons that never activate, hence diminishing the network's efficiency. ReLU is not

constrained above, so the output can grow indefinitely, generating numerical instability and making it more challenging to train the network. ReLU may not be suitable for certain jobs in which negative values are essential for describing particular characteristics. Finally, ReLU can be sensitive to initialization, therefore it may be required to employ careful initialization strategies for optimal learning.

Usage:

One application of ReLU is in image recognition, where the goal is to classify images into different categories. For example, a company that sells clothing might want to build a system that can automatically categorize images of clothing based on their type, color, and other features. To accomplish this task, the company could train a convolutional neural network (CNN) with ReLU activation function in the hidden layers. The input to the network would be an image, and the output would be a probability distribution over the possible clothing categories

4. Elu

ELU stands for Exponential Linear Unit, and it is an activation function used in neural networks. ELU activation function is a powerful tool for building neural networks that can learn complex relationships between inputs and outputs. Its ability to mitigate the vanishing gradient problem, improve performance, and robustness to noise make it a popular choice in deep learning.

Equation:

$f(x) = x, x >= 0$

$f(x) = alpha * (exp(x) - 1), x < 0$

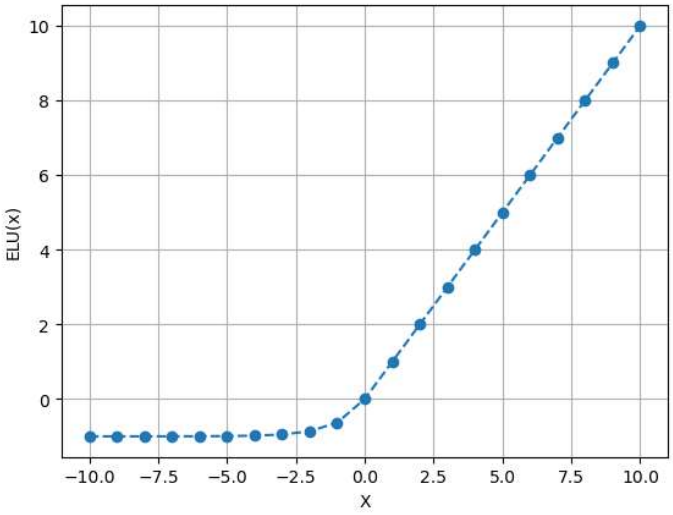
ELU activation function is defined as:

```
alpha = 1.0

y = np.where(x>=0, x, alpha*(np.exp(x)-1))

plot_graph(y, "ELU(x)")
```

where x is the input to the function, and alpha is a small positive constant. When x is positive or zero, the ELU function returns the input x unchanged. This is similar to the identity function. When x is negative, the ELU function applies an exponential function to x, subtracts 1, and multiplies by alpha. This creates a smooth curve that transitions from negative to positive values. The alpha

parameter controls the slope of this curve, and a small value is typically used to ensure that the function is nearly linear for small negative values.

Resulted Graph:



where x is the input to the function, and alpha is a small positive constant. When x is positive or zero, the ELU function returns the input x unchanged. This is similar to the identity function. When x is negative, the ELU function applies an exponential function to x, subtracts 1, and multiplies by alpha. This creates a smooth curve that transitions from negative to positive values. The alpha parameter controls the slope of this curve, and a small value is typically used to ensure that the function is nearly linear for small negative values.

The Exponential Linear Unit (ELU) activation function offers various advantages over other activation functions, such as the ReLU activation function. Using a non-zero derivative for negative inputs helps minimize the vanishing gradient problem, allowing gradients to flow more easily through the network. ELU has also been demonstrated to outperform alternative activation functions in a variety of situations, especially when training deep neural networks.ELU can generate negative values, making the network more noise-resistant. It is smooth and differentiable everywhere, including at x=0, which is advantageous for gradient-based optimization strategies. ELU can result in faster training convergence than other activation functions.In contrast to some other activation functions, such as ReLU, the ELU function is computationally expensive to compute. There may be less information available on how to use it effectively, as it is not as popular.

The ELU function can be more difficult to implement than other activation functions in certain hardware or software environments. It can also saturate for large negative inputs, resulting in very tiny derivatives and unstable gradients. Lastly, the ELU function may not be appropriate for certain

tasks, such as picture segmentation, in which negative values are essential for representing particular characteristics.

ELU provides various advantages over other activation functions, including enhanced performance, a reduced gradient vanishing problem, and increased noise resistance. Yet, it may be computationally costly and challenging to implement in certain contexts. In addition, it may not be appropriate for many jobs, such as picture segmentation, where negative values are essential for representing particular features.

Usage:

One application of ELU is in natural language processing (NLP), where the goal is to process and understand human language. For example, a company might want to build a chatbot that can communicate with customers in a natural language and answer their questions. To accomplish this task, the company could train a recurrent neural network (RNN) with ELU activation function in the hidden layers. The input to the network would be text, and the output would be a response to the customer's question. During training, the ELU activation function helps the neural network learn to recognize and represent complex patterns in the text data, such as word order and syntax. ELU also helps to prevent the vanishing gradient problem, which can occur when the activation function is too shallow, making it difficult for the model to learn.

5. Selu

The SELU (scaled exponential linear unit) activation function is a type of activation function used in artificial neural networks. It was introduced in 2017 by Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter in their paper "Self-Normalizing Neural Networks. Normalization is a technique used to adjust the scale of numeric data to a common range. There are three types: input, batch, and internal normalization. Input normalization involves scaling the values to be between zero and one. Batch normalization involves changing the values between each layer of the network so that their mean is zero and their standard deviation is one. Internal normalization, which is the key idea behind SELU, involves each layer keeping the previous layer's mean and variance to help stabilize and improve the performance of deep neural networks. The SELU function is designed to automatically normalize the outputs of each layer in a neural network, which can help prevent the vanishing gradient problem and make training more efficient.

The SELU activation function is mathematically expressed as:

f(x) = lambda * x if x > 0

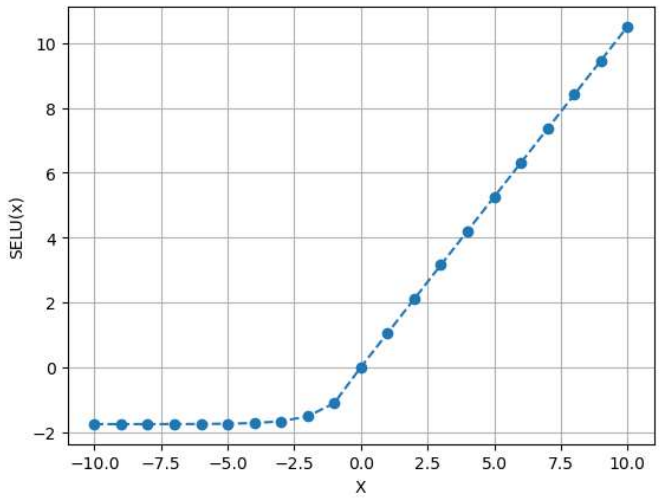
lambda * alpha * (exp(x) - 1) if x <= 0

where alpha and scale are constants that are calculated based on the mean and standard deviation of the input data. To calculate the constants alpha and scale, the mean and standard deviation of the input data are first calculated. Then, alpha is set to a value of about 1.67326, and scale is set to a value of about 1.0507. These values ensure that the output of the SELU function will have a mean close to zero and a standard deviation close to one, even for very deep neural networks.

SeLU activation function is defined as

```
def selu(x, scale=1.0507, alpha=1.6733):
    return np.where(x > 0, scale*x, scale*(alpha*np.exp(x)-alpha))

plot_graph(selu(x), "SELU(x)")
```
✓ 0.3s

Resulted Graph:



The advantages of using the SELU activation function include its self-normalizing property, which means that it can help prevent the vanishing gradient problem. SELU has been shown to improve the performance of deep neural networks, particularly for networks with many layers. Another advantage is that SELU does not require additional techniques like dropout or batch normalization to achieve good performance, and it works well with large datasets without overfitting. However, there are also some disadvantages to using SELU. Initialization of network weights is crucial for SELU to work properly, and it can be a complex process. Additionally, SELU may not work as well for networks with certain types of architectures or datasets. Furthermore, the performance of SELU can be sensitive to the scaling of the input data, which means that preprocessing may be required to ensure that the inputs are within the appropriate range.

Usage:

One application of SELU is in medical image analysis, where the goal is to automatically detect and diagnose medical conditions from images such as X-rays, MRIs, and CT scans. For example, a hospital might want to build a system that can automatically diagnose lung cancer from chest X-rays. To accomplish this task, the hospital could train a deep convolutional neural network (CNN) with SELU activation function in the hidden layers. The input to the network would be an X-ray image, and the output would be a probability distribution over possible medical conditions. During training, the SELU activation function helps the neural network learn to recognize and represent complex patterns in the image data, such as edges, textures, and shapes. The use of the SELU activation function also helps to improve the accuracy and robustness of the model, making it more reliable for medical diagnosis.

Conclusion:

In conclusion, activation functions play a crucial role in neural network architectures. They are used to introduce non-linearity into the model, allowing it to learn complex patterns and make more accurate predictions. Choosing the right activation function for a specific task can have a significant impact on the performance of the model.

Through this report, we have explored some of the commonly used activation functions such as sigmoid, ReLU, and tanh, and discussed their properties, advantages, and disadvantages. It is important to note that there is no one-size-fits-all activation function, and the selection of an activation function depends on various factors such as the architecture of the model, the nature of the problem, and the size of the dataset. In recent years, several new activation functions have been proposed, such as Swish, Mish, and GELU, which have shown promising results on different tasks. It is evident that further research is required to develop new activation functions that can improve the performance of neural networks.

In conclusion, activation functions are a critical component of neural network models, and their proper selection can significantly impact the performance of the model. Therefore, it is essential to carefully evaluate and choose the most appropriate activation function for a specific task.