American International University-Bangladesh (AIUB)

Department of Computer Science
Faculty of Science &Technology (FST)
Spring 22-23
COMPUTER VISION AND PATTERN RECOGNITION

Section: A

Report of Assignment 2

Date: 11 – March - 2023

Submitted By:

Serial No.	Student Name	Student ID
1.	Sultanul Arifeen Hamim	20-42017-1

Submitted to:

DR. DEBAJYOTI KARMAKER

Associate Professor, Faculty

Computer Science

American International University-Bangladesh (AIUB)

Activation Function

➔An activation function in artificial neural networks is a formula that is applied to each neuron's output. Based on the input it gets from the preceding layer, the activation function decides whether the neuron should be "activated" or not.

The neural network can learn intricate correlations between inputs and outputs thanks to activation functions, which add non-linearity into the system. A neural network would essentially be a linear model without activation functions since each layer would just be a linear combination of the outputs of the previous layer.

There are numerous categories of activation functions, each with unique benefits and drawbacks.Sigmoid, tanh, ReLU, are a few of the most popular activation functions.

The challenge at hand and the neural network's architecture affects the choice of activation function. ReLU, for instance, is frequently employed in deep learning because it is computationally effective and reduces the vanishing gradient problem. Meanwhile, because they can provide output values between 0 and 1, sigmoid and tanh are frequently used in binary classification tasks.So it can be said that an activation functions are an essential part of artificial neural networks because they enable them to learn intricate non-linear correlations between inputs and outputs.

In this report, I am going to discuss **Step, Sigmoid, Tanh, ReLU, Elu, and Selu**.

Step Activation Function

➔The Step activation function is a simple binary function that returns 1 when the input value exceeds or is equal to a threshold value and 0 when it is not. It is sometimes referred to as the Unit Step Function or the Heaviside Step Function.

The step function has the following mathematical definition:

f(x) = 0, if x 0,

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

In binary classification situations, where the output can only be either 0 or 1, like in logistic regression, the step function is frequently utilized. Although its utility is restricted due to its discontinuous character, it can also be utilized as an activation function in neural networks.

Step activation function is defined as:

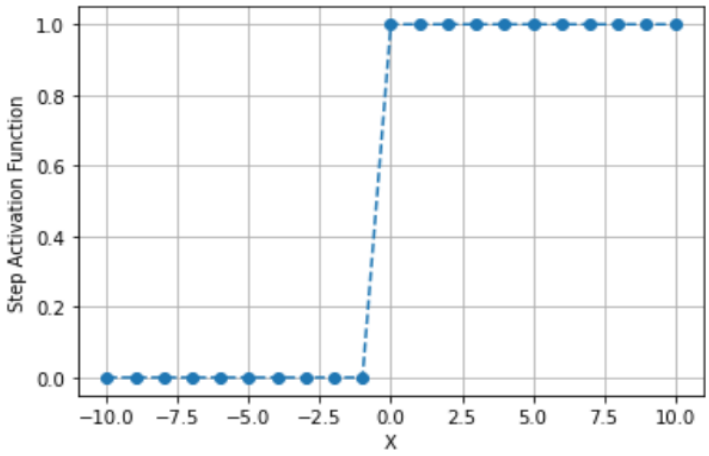
```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-10, 11, 1)
print(x)
def plot_graph(y,ylabel):
    plt.figure()
    plt.plot(x,y, 'o--')
    plt.grid(True)
    plt.xlabel("X")
    plt.ylabel(ylabel)
    plt.show()

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

plot_graph(y, "Step Activation Function")
```

Resulted Graph:



The step activation function is a simple and efficient way to produce binary output in certain types of binary classification problems. It is easy to understand and interpret, as its output is a clear decision boundary. However, it has some disadvantages. The function is not continuous, which makes it difficult to use in gradient-based optimization methods like backpropagation. Also, it is not differentiable at the threshold value, which can cause problems in some optimization algorithms. The step function can be sensitive to small changes in input, which makes it unstable and difficult to train. Overall, the step function is useful in specific scenarios, but its limitations make it less commonly used in modern neural networks

Sigmoid Activation Function

➔ The sigmoid function is a type of mathematical function that is widely used in neural networks and logistic regression models as an activation function. It transforms any input value into an output value between 0 and 1. This function calculates the weighted sum of the inputs and then applies the sigmoid function to the result. The output of the sigmoid function can be interpreted as the probability of the input belonging to a specific category.

The sigmoid activation function is defined by the mathematical expression:

σ(z) = 1 / (1 + e^(-z)),

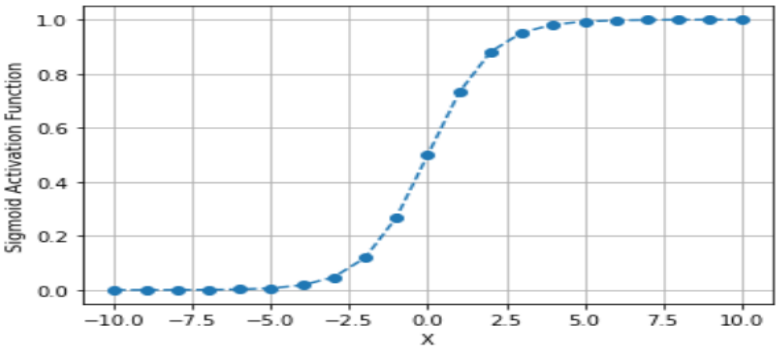
where z represents the input value, and σ(z) is the output value between 0 and 1. This function maps any real number to a probability value between 0 and 1.

Sigmoid activation function is defined as:

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

plot_graph(y, "Sigmoid Activation Function")
```

Resulted Graph:



The sigmoid activation function has certain benefits, such as producing output values between 0 and 1 that are easily interpretable. However, it also has several drawbacks, including the potential for the vanishing gradient problem, saturation at the extremes of its range, and computational inefficiency due to the need to compute exponentials. In addition, it is not zero-centered, which can make it challenging for neural networks to converge during training. As a result, although the sigmoid function has some advantages, it is not commonly used in modern neural networks due to its limitations. Instead, alternative activation functions like ReLU or its variants are often used because they are more effective in overcoming these challenges.

Tanh Activation Function

➔A popular activation function in neural networks is Tanh, which is also called a hyperbolic tangent. It is a mathematical formula that turns numbers given as input into numbers between -1 and 1. Tanh uses a transformation to change the value you give it into a number between -1 and 1. In more detail, the function multiplies the exponential function of the input by the sum of the exponential functions of the input and the negative input, subtracts the exponential function of the negative input, and then divides the result by the exponential function of the input.

The mathematical expression of the Tanh activation function is:

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

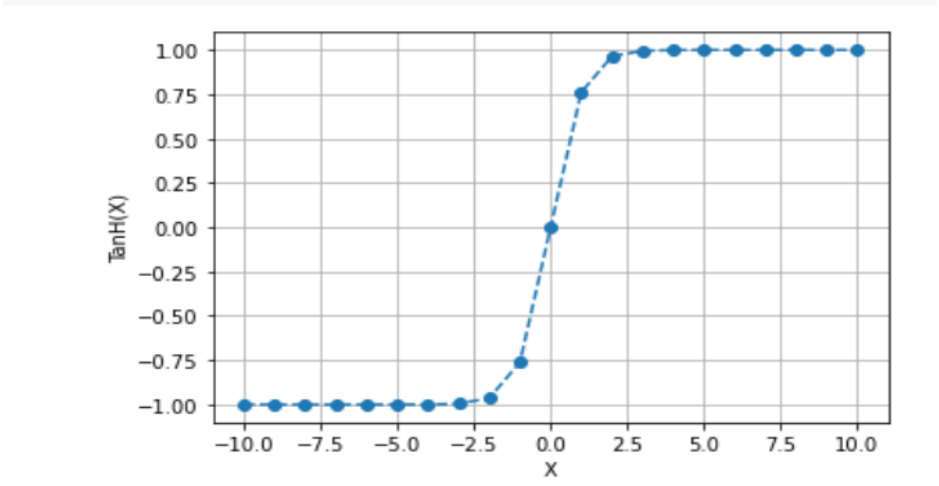
where e is Euler's number (approximately 2.71828) and x is the input to the function. The output of the function f(x) is a value between -1 and 1.

Tanh activation function is defined as:

```
y = np.tanh(x)

plot_graph(y, "Tanh Activation Function")
```

Resulted Graph:



Like the sigmoid function, the Tanh activation function gives values between -1 and 1 as output. It can simulate intricate interactions between input and output variables and is non-linear. Tanh is advantageous in gradient-based optimization methods like backpropagation since it has a derivative. Deep neural networks may find it challenging to understand complex associations because Tanh is susceptible to the vanishing gradient problem. Tanh can experience saturation at the extremes of its range, similar to the sigmoid function, which can impede learning. Moreover, Tanh's exponential computation requirements might be computationally expensive. Tanh is a non-linear activation function that, in particular kinds of neural networks, yields output values between -1 and 1. To sum up, Tanh However, it can be computationally expensive to compute and is susceptible to saturation and the vanishing gradient problem.

Relu Activation Function

➔The ReLU, or rectified linear activation function, is a piecewise linear or nonlinear function that outputs the input directly if it is positive and zeros if it is negative. At the moment, it is the main activation function for convolutional neural networks (CNNs) and multilayer perceptrons. The ReLU activation function has become popular in deep learning because it is easy to understand, easy to compute, and helps solve the "vanishing gradient" problem. The network's weights must be updated because of the vanishing gradient problem, which happens when gradients are incredibly small during backpropagation. ReLU has a non-zero derivative for positive inputs, allowing gradients to move more easily throughout the network.

Mathematically, the ReLU function can be expressed as:

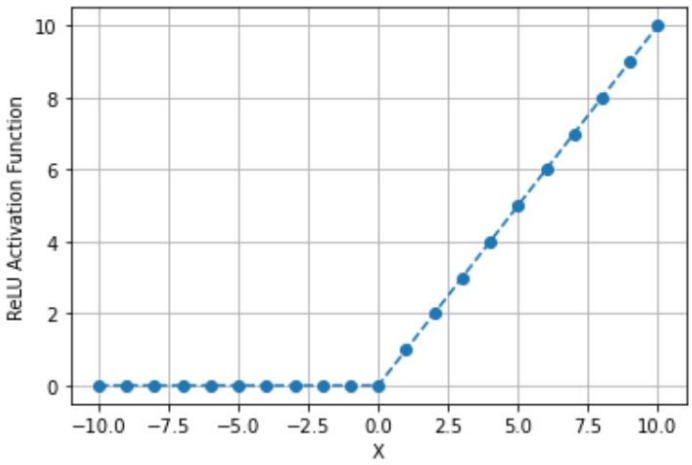
f(x) = max (0, x) [where x is the input to the function and f(x) is the output]

ReLU activation function is defined as:

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

plot_graph(y, "ReLU Activation Function")
```

Resulted Graph:



Neurons in a neural network receive inputs, and each input value is multiplied by a weight, added to a bias term, and passed through an activation function. The Rectified Linear Unit (ReLU) is a popular activation function used in the hidden layers of neural networks. ReLU introduces non-linearity to the output of a neural network, which is critical for learning complex patterns and relationships between inputs and outputs. When the input is greater than or equal to zero, ReLU outputs the input value, and when the input is negative, the output is zero. The non-zero derivative of ReLU for positive inputs enables gradients to flow through the network more efficiently, which helps mitigate the vanishing gradient problem.

ReLU has several benefits, including computing efficiency, non-linearity, sparsity, and the ability to mitigate the vanishing gradient problem. It is computationally efficient and easy to implement, making it a popular choice for activation functions in deep learning. ReLU is non-linear, allowing the network to learn complicated input-output correlations, and it can produce sparse representations that aid in reducing computation and preventing overfitting. However, there are also possible drawbacks to utilizing the ReLU function. For instance, it may result in "dead" neurons that never activate, reducing the efficiency of the network. Additionally, ReLU is not constrained above, so the output can grow indefinitely, leading to numerical instability and making it harder to train the network.

Moreover, ReLU may not be suitable for certain tasks in which negative values are necessary to describe specific features. Finally, ReLU can be sensitive to initialization, so it may require careful initialization strategies for optimal learning. Overall, ReLU is a powerful activation function that helps neural networks learn complex relationships and patterns in data.

Elu Activation Function

➔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 The ELU activation The

ELU activation function is defined as:

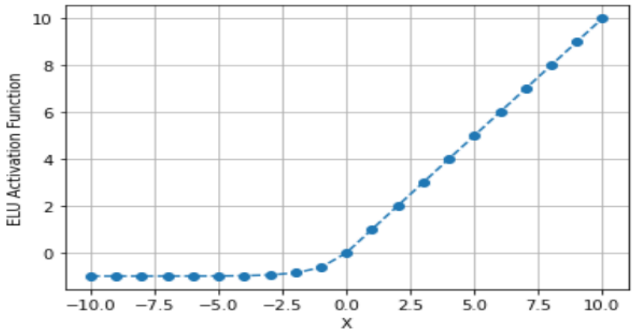
```
alpha = 1.0

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

plot_graph(y, "ELU Activation Function")
```

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:



The Exponential Linear Unit (ELU) is an activation function that applies an exponential function to negative inputs, resulting in a smooth curve that transitions from negative to positive values. The alpha parameter controls the slope of the curve. ELU offers various advantages over other activation functions, such as the ReLU activation function. It helps minimize the vanishing gradient problem by using a non-zero derivative for negative inputs, allowing gradients to flow more easily through the network. ELU has also been demonstrated to outperform alternative activation functions, 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. However, it is computationally expensive to compute and may be more difficult to implement in certain hardware or software environments. ELU can saturate for large negative inputs, resulting in tiny derivatives and unstable gradients. Lastly, the ELU function may not be suitable for certain tasks, such as picture segmentation, where negative values are essential for representing particular characteristics.

In summary, ELU provides various benefits over other activation functions, including enhanced performance and reduced gradient vanishing problems. It is more noise-resistant and differentiable, making it suitable for gradient-based optimization strategies. However, it may be computationally costly and challenging to implement in some contexts and may not be appropriate for certain tasks.

Selu Activation Function

➔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". The 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. There are three types of normalization techniques used in neural networks, input, batch, and internal normalization. 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 activation function is mathematically expressed as a piecewise function that returns lambda times the input if the input is positive, and lambda times a scaled exponential function minus alpha if the input is negative.

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

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

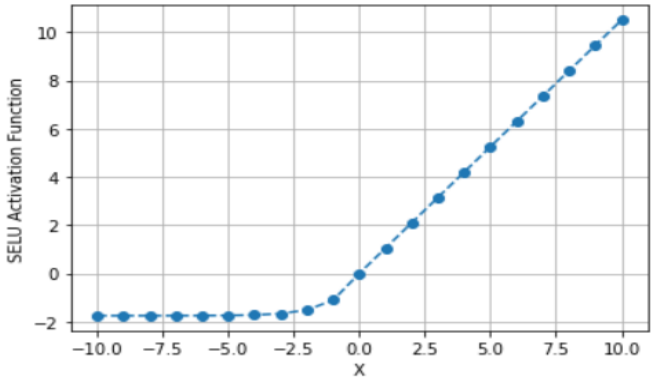
The constants lambda and alpha are calculated based on the mean and standard deviation of the input data, and ensure that the output of the SELU function will have a mean close to zero and a standard deviation close to one. The SELU activation function has been shown to outperform other activation functions, such as the ReLU function, in certain situations. However, the SELU function is not appropriate for all tasks and may be computationally expensive to compute. In summary, the SELU activation function provides automatic normalization of each layer in a neural network, which can lead to improved performance and more efficient training.

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 Activation Function")
```

Resulted Graph:



Advantages of using 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.