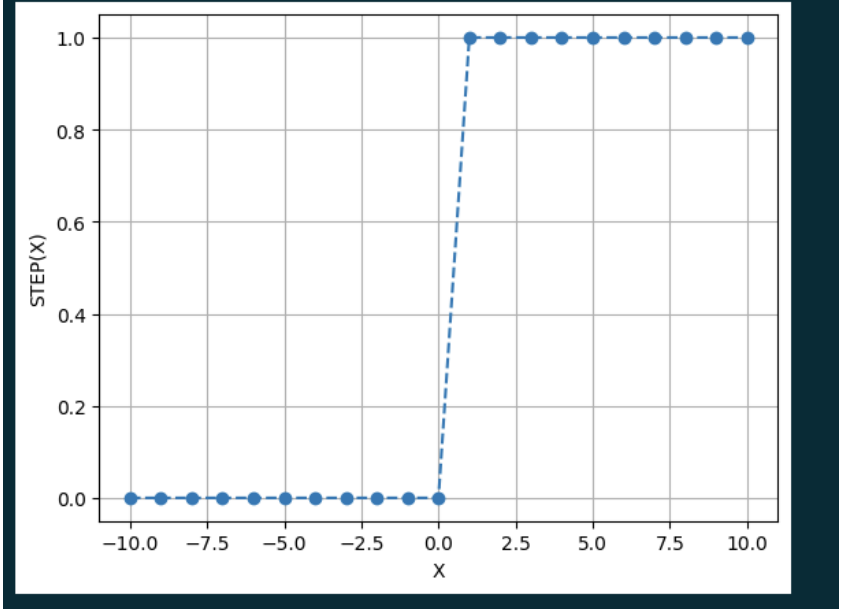
Activation Functions

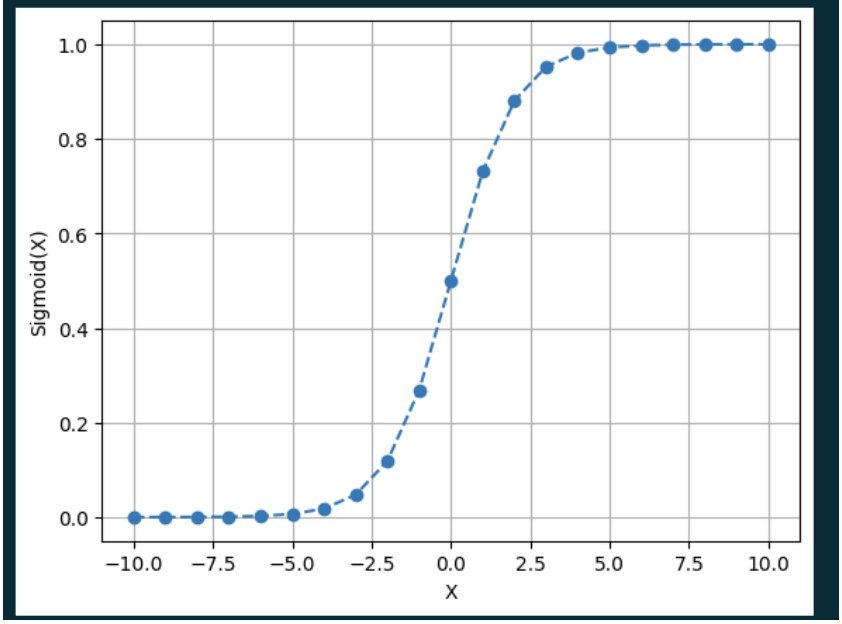
- Step Function:
  - A simple activation function that outputs a binary value of 0 or 1 based on the input value.
  - It has a value of 0 for all negative inputs and a value of 1 for all positive inputs.
  - It is discontinuous and not differentiable, making it unsuitable for use in gradient-based optimization algorithms.
  - It can be useful in binary classification problems where the output should be either 0 or 1.

```
y = list(map(lambda n: 1 if n>0.5 else 0, x))
plot_graph(y,"STEP(X)")
```



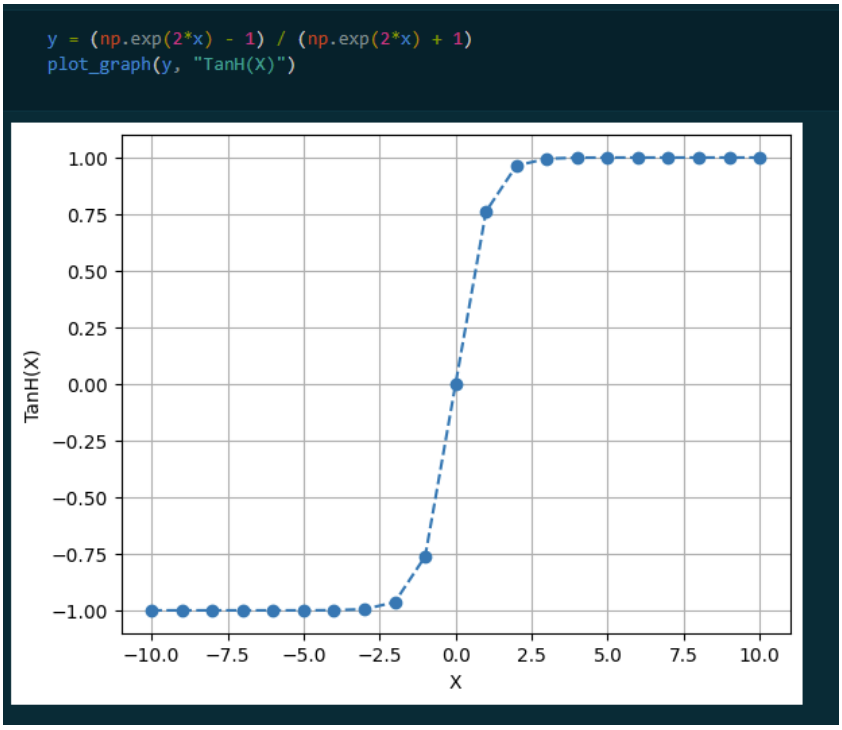
- Sigmoid or Logistic Activation Function:
  - A smooth, S-shaped function that maps any input value to a value between 0 and 1.
  - It is widely used in binary classification problems as it can be interpreted as the probability of the input belonging to a certain class.
  - It suffers from the vanishing gradient problem, where the gradients become very small as the input value moves away from 0, making it difficult to optimize deep neural networks with many layers.

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



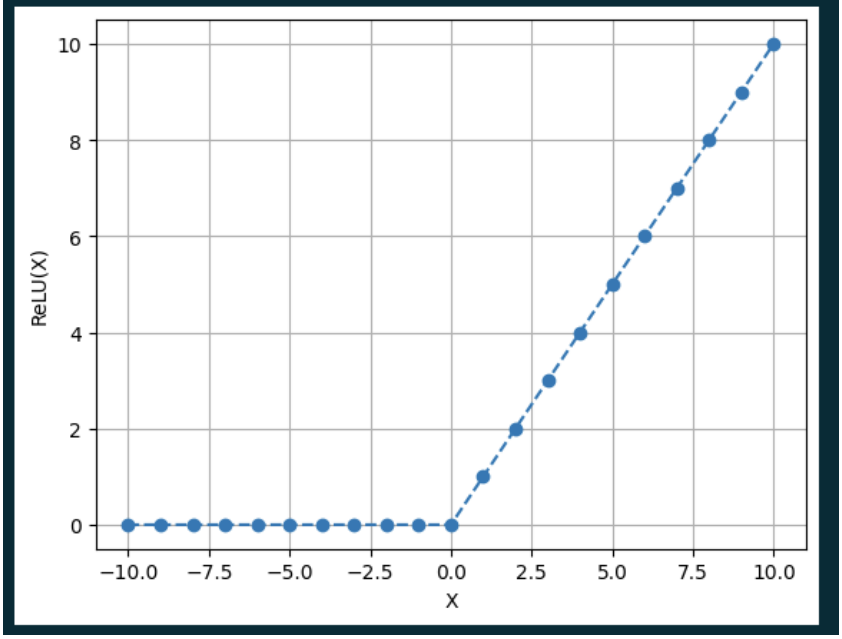
- Hyperbolic Tangent(TanH) Function:
  - A function that maps any input value to a value between -1 and 1.
  - It is like the sigmoid function but has a range that is symmetric around 0.
  - It is also prone to the vanishing gradient problem and is rarely used as an activation function in the hidden layers of deep neural networks.

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



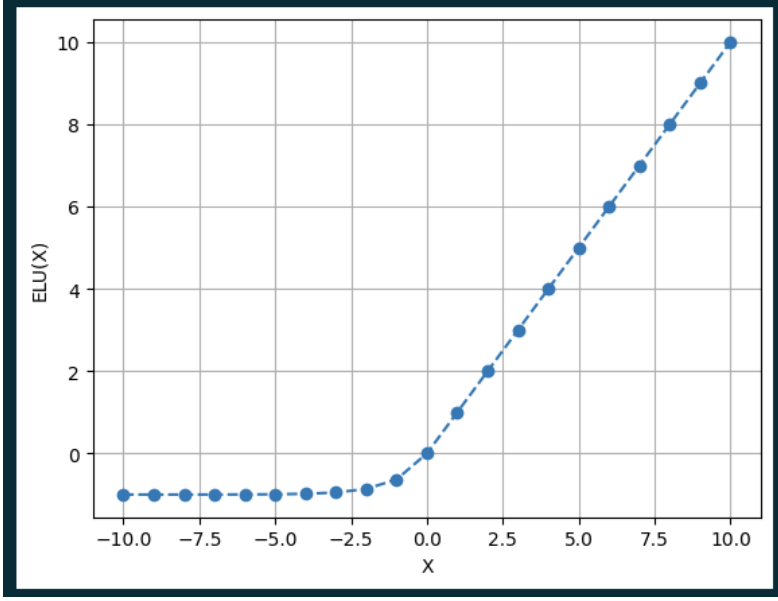
- ReLU (Rectified Linear Unit) Activation Function:
  - A simple and widely used activation function that returns the input value if it is positive and 0 otherwise.
  - It is linear for all positive inputs and non-linear for all negative inputs.
  - It is computationally efficient and can be used in deep neural networks with many layers.
  - It can suffer from the "dying ReLU" problem, where the neuron becomes inactive and stops learning if the input value is negative.

```
y = list(map(lambda a: a if a>=0 else 0, x))
plot_graph(y,"ReLU(X)")
```



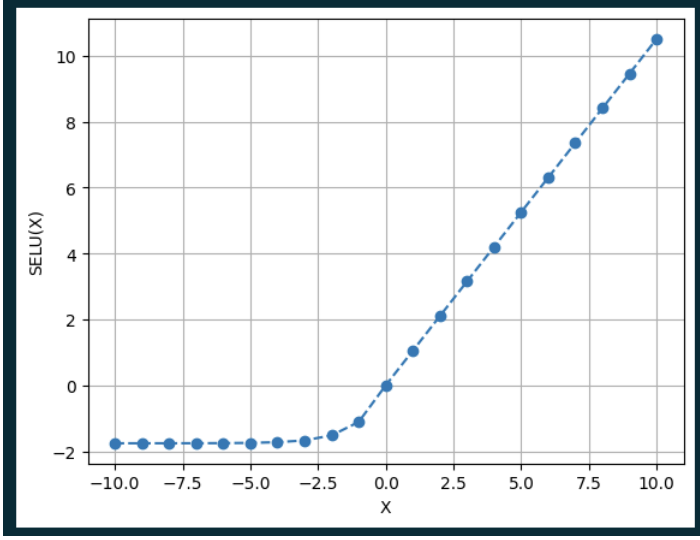
- ELU (Exponential Linear Unit) Activation Function:
  - A function that is like the ReLU function but uses an exponential function for negative input values.
  - It is designed to overcome the "dying ReLU" problem by allowing negative input values to produce non-zero gradients.
  - It is also less prone to the vanishing gradient problem compared to the sigmoid and TanH functions.

```python
def elu(x, alpha=1.0):
    #return np.where(x >= 0, x, alpha*(np.exp(x)-1))
    return list(map(lambda a: a if a >= 0 else alpha * (np.exp(a) - 1), x))
y = elu(x)
plot_graph(y, "ELU(X)")
```



- SELU (Scaled Exponential Linear Unit) Activation Function:
  - A function that is like the ELU function but uses a scaled exponential function for negative input values.
  - It is designed to further improve the performance of deep neural networks by enforcing a certain type of self-normalization of the hidden units.
  - It ensures that the output of each layer has zero mean and unit variance by using the two hyperparameters. This helps to reduce the vanishing and exploding gradient problems that can occur in deep neural networks.
  - It has been shown to significantly improve the accuracy and training speed of deep neural networks.
  - However, it requires specific initialization of the weights and can only be used in feed-forward neural networks.

```python
def selu(x, scale=1.0507, alpha=1.6733):
    #return scale * np.where(x >= 0, x, alpha * (np.exp(x) - 1))
    return list(map(lambda a: scale * (a if a >= 0 else alpha * (np.exp(a) - 1)), x))
y = selu(x)
plot_graph(y, "SELU(X)")
```



Summarizing the activation functions in their terms with valid references:

Each activation function has its own advantages and limitations. The sigmoid function is easy to understand and implement, but it suffers from the vanishing gradient problem, which can make training deep neural networks difficult (Hochreiter, 1998). The ReLU function is computationally efficient and helps alleviate the vanishing gradient problem, but it can lead to dead neurons, which can cause the network to stop learning (Nair & Hinton, 2010).

The ELU function solves the problem of dead neurons and has been shown to produce better results than ReLU in some cases (Clevert, Unterthiner, & Hochreiter, 2016). However, ELU has a slower test time than ReLU, which can be a disadvantage in some applications.

SELU is an extension of ELU and has been shown to perform well in deep neural networks with weight initialization and activation functions that meet certain conditions (Klambauer, Unterthiner, Mayr, & Hochreiter, 2017). However, more research is needed to fully understand its properties and advantages.

Choosing the appropriate activation function depends on the specific task and dataset. Researchers and practitioners continue to explore and experiment with different activation functions to improve the performance of deep learning models.

References:

Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). arXiv preprint arXiv:1511.07289.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02), 107-116.

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. In Advances in neural information processing systems (pp. 971-980).

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).