CLASS: B.E. E &TC **SUBJECT: ML**
EXPT. NO.: 11 **DATE: 08/03/21**
ROLL NO.: 42428

TITLE: Activation Function

CO 3: Apply the fundamentals of Artificial Neural Network (ANN) to design and implement Back propagation algorithm and Convolution neural networks for various applications like classification, Self-Organizing Feature Maps, Learning vector quantization, Radial Basis Function networks and object recognition. Using a substantial training and test data set, validate the efficacy of designed neural network in-terms of confusion matrix, accuracy and recognition rate.

CO 4: Carry out experiments as an individual and in a team, comprehend and write a laboratory record and draw conclusions at a technical level.

SOFTWARES REQUIRED: MATLAB 7.0 or Python

THEORY:

Basic terms used in Neural Networks:

1. **Dataset:**
   Dataset is the data used for machine learning application.
2. **Model Architecture and Model Weights:**
   Model architecture is the type of algorithm that we apply on our data. A model architecture uses random weights in the beginning but as it is trained on data, it learns better weights.
3. **Train and Test Data**
   - Training phase is when we are improving our model by changing the model weights. We do this by showing new data to our model and correcting model weights to improve prediction or classification task.
   - Testing phase is when we are only predicting and classifying.

4. Train and Test Data:

- Our learning models require a unit that can quantify the overall performance of our model on our dataset. Our training phase relies heavily on our data set and on our defined parameters. However, using the entire dataset would mean that our goal seems to be fitting our model to our dataset alone, which was never the objective in the first place. Exhaustively utilizing the dataset and tweaking the parameters would possibly result in an ideal model and not a prediction model.
- One good way to avoid this would be to split our dataset into two parts of some defined ration - a training set and testing set. We never show test dataset to our model and train our model by using only the train dataset. This allows us to evaluate our model accuracy on unseen data.

5. Weights and Biases:

- Weights are model weights. These are just floating point numbers (normally).
- During training these numbers are changed and towards the end a good collection of these numbers are obtained.
- Biases are also floating point numbers which are similarly set during training. For example, y = Wx +B Here, W is weight and B is bias.

6. Forward pass and Backward pass:

- Forward pass is calculating the predictions.
- Backward pass involves resolving the error in predictions by changing weights.
- Forward pass happens in both training and testing phase. But backward pass only happens during training.

7. Batch, Mini-batch and Epoch:

- A batch is collection of entire training data samples or testing data samples.
- Mini-batch is subset of this training data.
- An epoch is running forward and backward pass on whole of the training data.

8. Error and Loss:

- Error is the difference between observed prediction and actual prediction of the algorithm. There are different ways to calculate this error.

- Loss is defined as average error over all examples. Loss can be either training loss or testing loss. Training loss is calculated over all or mini-batch of training examples.
- In short, loss is a parameter that explains accuracy of our model. The less the loss, better the model.

9. **Overfitting (Variance) and Under-fitting (Bias):**
   - Overfitting happens when your model just remembers all the data that you have.
   - This is a common trait to see, if your model is complex and data is small in quantity. When this happens the training loss reduces while testing loss keeps increasing.
   - Under-fitting is when your model cannot fit the training data.
   - This happens if model is not complex enough to model the data properly.

10. **Regression and Classification:**

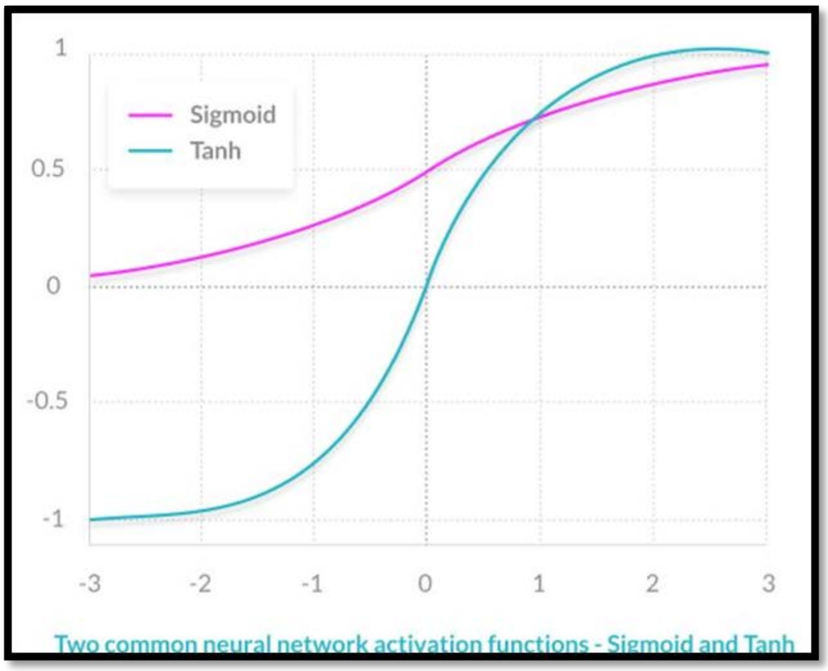
    In Regression, the output variable can take any real values.

    For example, predicting house prices. Since house prices can be any real number within a range, it is a regression problem. One thing to notice is that the number itself has some value in case of regression. For example, a lower number for house price suggest that the price is low.

Neural Network Activation Function:

Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.

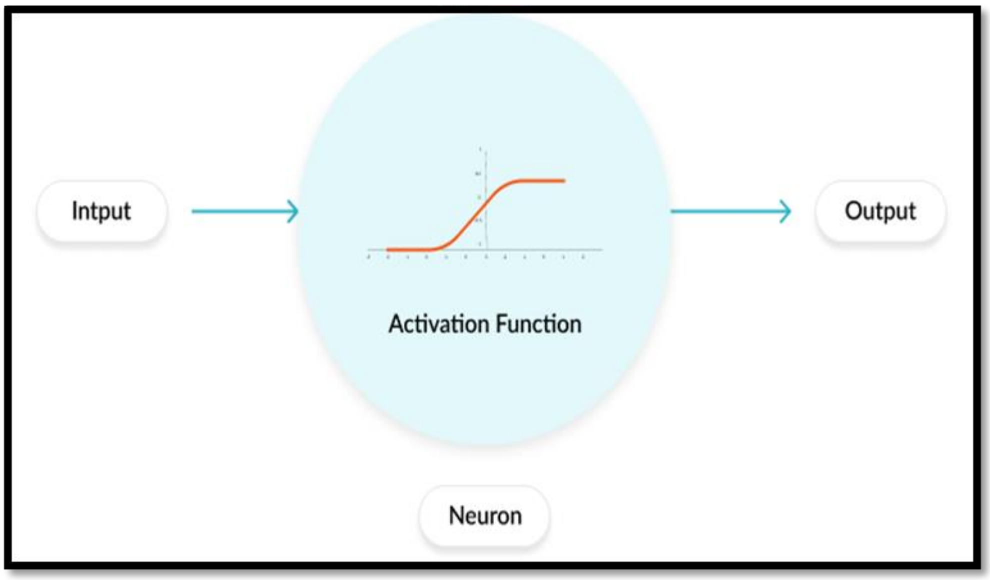
An additional aspect of activation functions is that they must be computationally efficient because they are calculated across thousands or even millions of neurons for each data sample. Modern neural networks use a technique called backpropagation to train the model, which places an increased computational strain on the activation function, and its derivative function.



Artificial Neural Networks and Deep Neural Networks:
Artificial Neural Networks (ANN) are comprised of a large number of simple elements, called neurons, each of which makes simple decisions. Together, the neurons can provide accurate answers to some complex problems, such as natural language processing, computer vision, and AI.

Role of the Activation Function in a Neural Network Model:
In a neural network, numeric data points, called inputs, are fed into the neurons in the input layer. Each neuron has a weight, and multiplying the input number with the weight gives the output of the neuron, which is transferred to the next layer.

The activation function is a mathematical “gate” in between the input feeding the current neuron and its output going to the next layer. It can be as simple as a step function that turns the moron output on and off, depending on a rule or threshold. It can be a transformation that maps the input signals into output signals that are needed for the neural network to function.

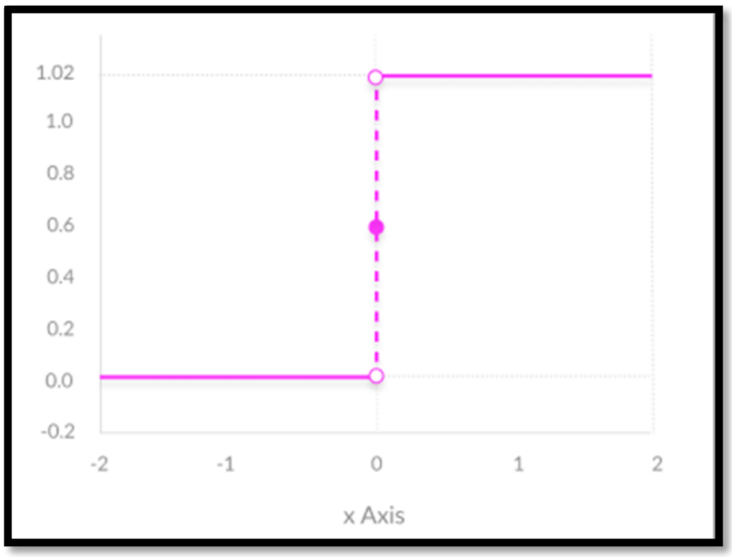


Increasingly, neural networks use non-linear activation functions, which can help the network learn complex data, compute and learn almost any function representing a question, and provide accurate predictions.

Types of Activation Functions:

BINARY STEP FUNCTION:

A binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.

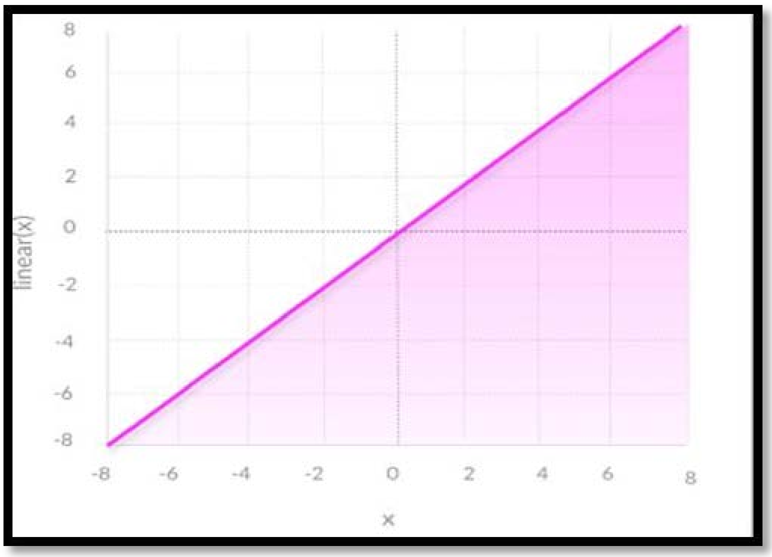


The problem with a step function is that it does not allow multi-value outputs. For example, it cannot support classifying the inputs into one of several categories.

LINEAR ACTIVATION FUNCTION:

A linear activation function takes the form

$$A = cx$$



It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no.

However, a linear activation function has two major problems:

1. **Not possible to use backpropagation** (gradient descent) to train the model—the derivative of the function is a constant, and has no relation to the input, X. So, it's not possible to go back and understand which weights in the input neurons can provide a better prediction.
2. **All layers of the neural network collapse into one**—with linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function). So, a linear activation function turns the neural network into just one layer. A

neural network with a linear activation function is simply a linear regression model. It has limited power and ability to handle complexity varying parameters of input data.

NON-LINEAR ACTIVATION FUNCTIONS:

Modern neural network models use non-linear activation functions. They allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modelling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality.

Almost any process imaginable can be represented as a functional computation in a neural network, provided that the activation function is non-linear.

Non-linear functions address the problems of a linear activation function:

1. They allow backpropagation because they have a derivative function which is related to the inputs.
2. They allow "stacking" of multiple layers of neurons to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex data sets with high levels of accuracy.

Common Nonlinear Activation Functions and How to Choose an Activation Function:

SIGMOID / LOGISTIC:

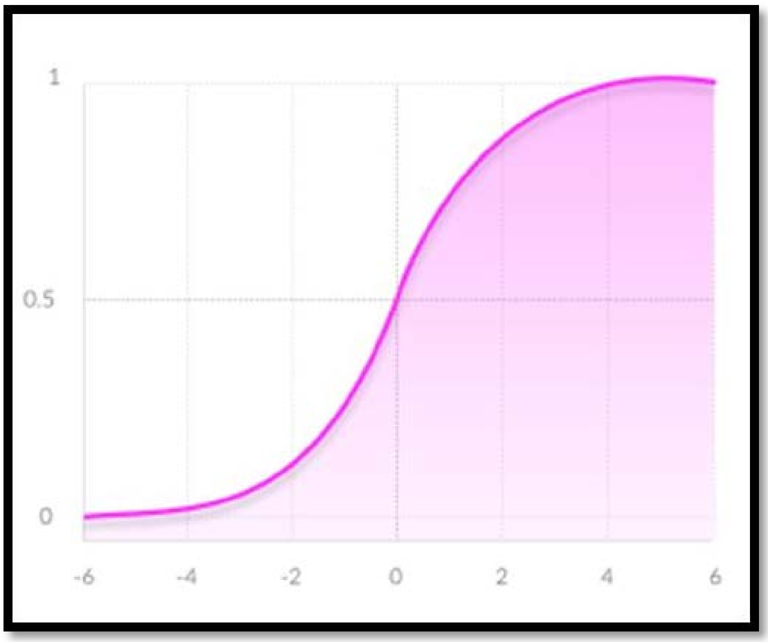
ADVANTAGES:

- Smooth gradient, preventing "jumps" in output values.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.

DISADVANTAGES:

- Vanishing gradient—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.

- Outputs not zero centered
- Computationally expensive



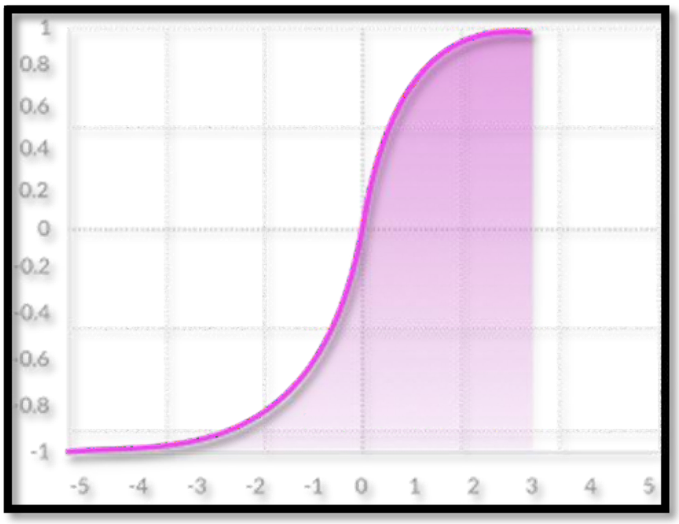
TANH / HYPERBOLIC TANGENT:

ADVANTAGES:

- Zero centered—making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
- Otherwise like the Sigmoid function.

DISADVANTAGES:

- Like the Sigmoid function



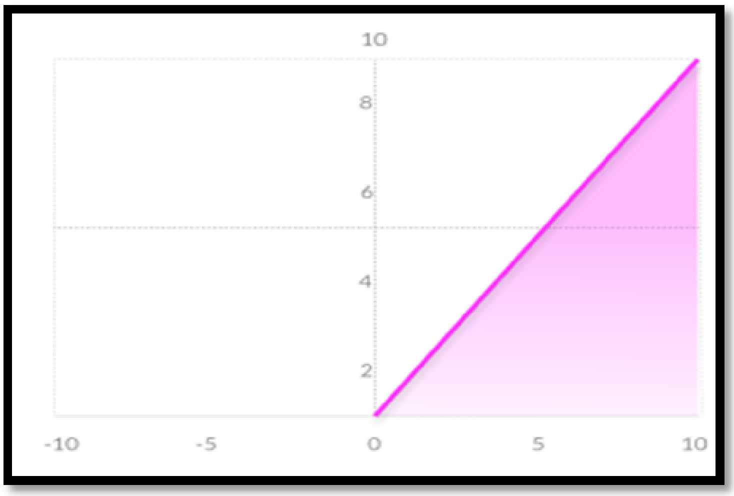
RELU:

ADVANTAGES:

- Computationally efficient—allows the network to converge very quickly
- Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation

DISADVANTAGES

- The Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.



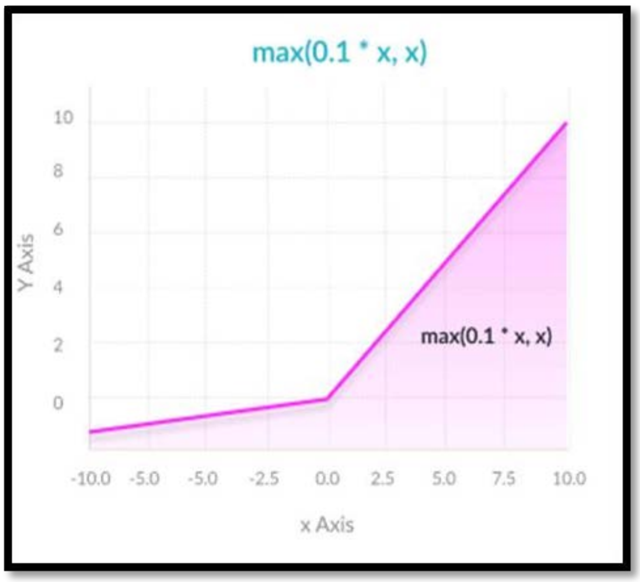
LEAKY RELU:

ADVANTAGES:

- Prevents dying ReLU problem—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values
- Otherwise like ReLU

DISADVANTAGES:

- Results not consistent—leaky ReLU does not provide consistent predictions for negative input values.



PARAMETRIC RELU:

ADVANTAGES:

- Allows the negative slope to be learned—unlike leaky ReLU, this function provides the slope of the negative part of the function as an argument. It is, therefore, possible to perform backpropagation and learn the most appropriate value of α.
- Otherwise like ReLU

DISADVANTAGES:

- May perform differently for different problems.

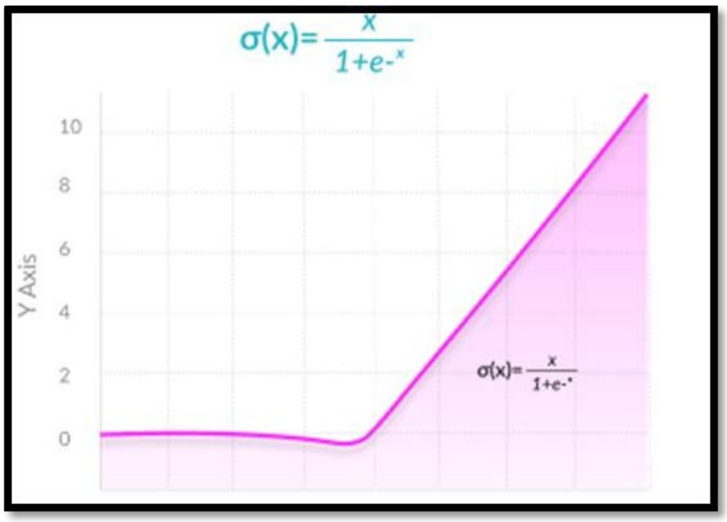
SOFTMAX:

ADVANTAGES:

- Able to handle multiple classes only one class in other activation functions— normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.
- Useful for output neurons—typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

SWISH:

Swish is a new, self-gated activation function discovered by researchers at Google. According to their paper, it performs better than ReLU with a similar level of computational efficiency. In experiments on ImageNet with identical models running ReLU and Swish, the new function achieved top - 1 classification accuracy 0.6-0.9% higher.



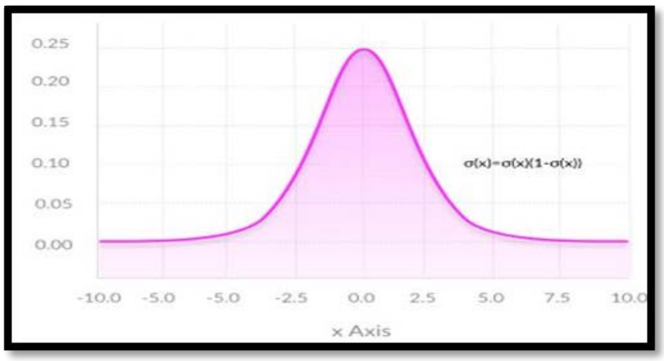
Derivatives or Gradients of Activation Functions:

- The derivative also known as a gradient of an activation function is extremely important for training the neural network.
- Give a man a fish and he will eat for a day. Teach a man how to fish and you feed him for a lifetime.
- Neural networks are trained using a process called back propagation this is an algorithm which traces back from the output of the model,

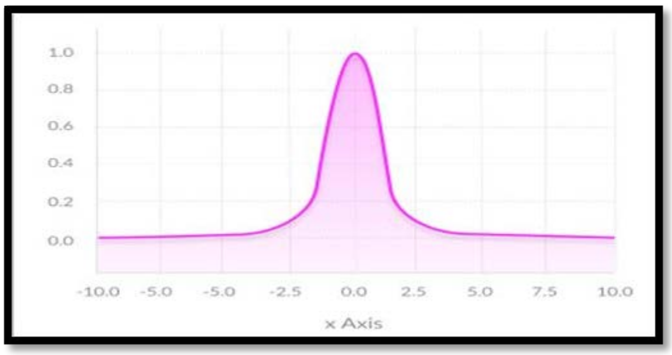
through the different neurons which were involved in generating that output, back to the original weight applied to each neuron. Backpropagation suggests an optimal weight for each neuron which results in the most accurate prediction.

ACTIVATION FUNCTIONS AND THEIR DERIVATIVE GRAPH (USED FOR BACKPROPAGATION):

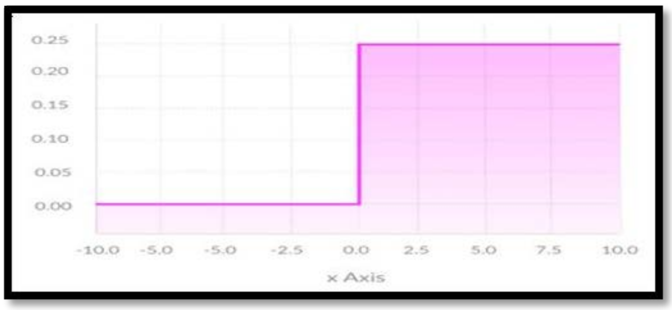
SIGMOID:



TANH:



RELU:



CONCLUSION:

Activation function is an important part of an artificial neural network. They basically decide whether a neuron should be activated or not. Thus it bounds the value of the net input.

The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output. Sigmoid is a smooth function and is continuously differentiable. The biggest advantage that it has over step and linear function is that it is non-linear. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. If you look at the ReLU function if the input is negative it will convert it to zero and the neuron does not get activated.

In practice, optimization is easier in Tanh method hence in practice it is always preferred over Sigmoid function. And it is also common to use the tanh function in a state to state transition model (recurrent neural networks).

REFERENCES:

i. Laurene Fausett, “Fundamentals of Neural Networks: Architectures, AlgorithmsAnd Applications”, Pearson Education, Inc, 2008.

ii. Winston, Patrick Henry. “Artificial Intelligence” 3rd Edition, Addison-Wesley, 1992.

iii. Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, “ImageNet Classification with Deep Convolutional Neural Networks”, University of Toronto

iv. Laurene Fausett ,"Fundamentals of Neural Networks: Architectures, Algorithms and Applications", Pearson Education, Inc, 2008.

v. S. N. Sivanandam , S. Sumathi, S. N. Deepa, "Introduction to Neural Networks using MATLAB", McGraw Hill, 2006.

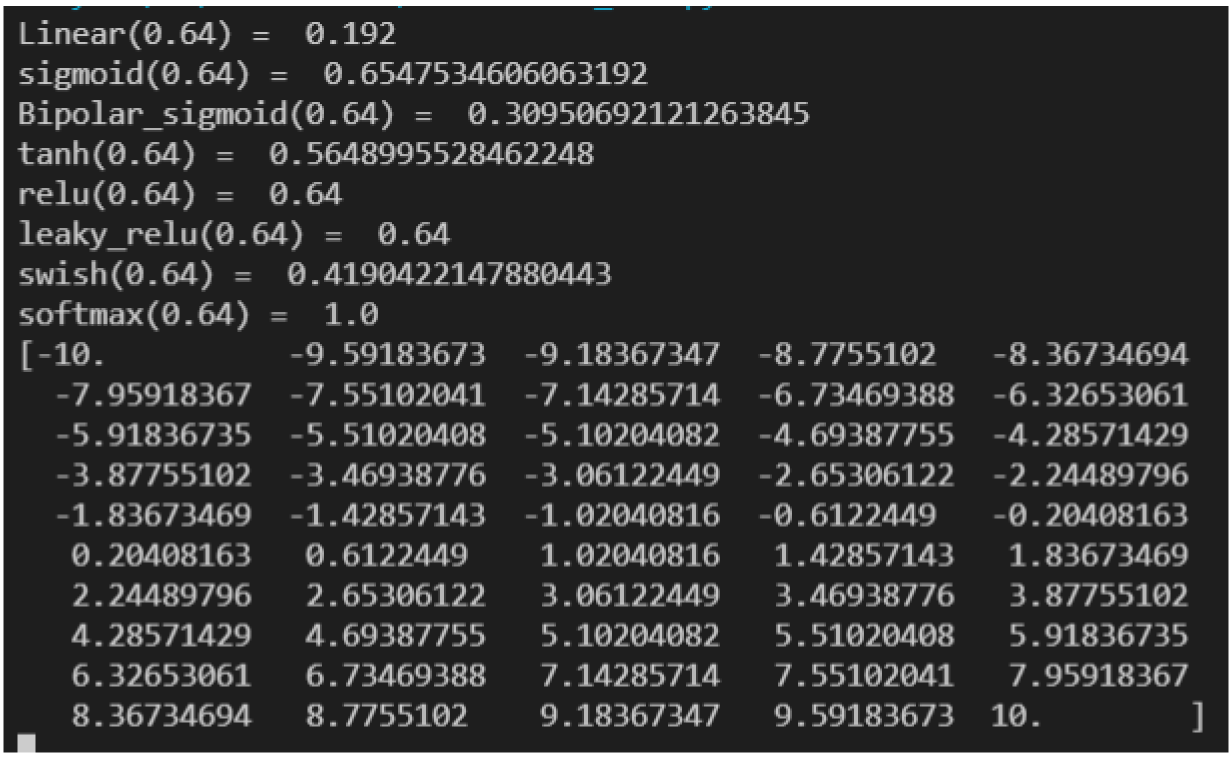
(Course Teacher)

Code :

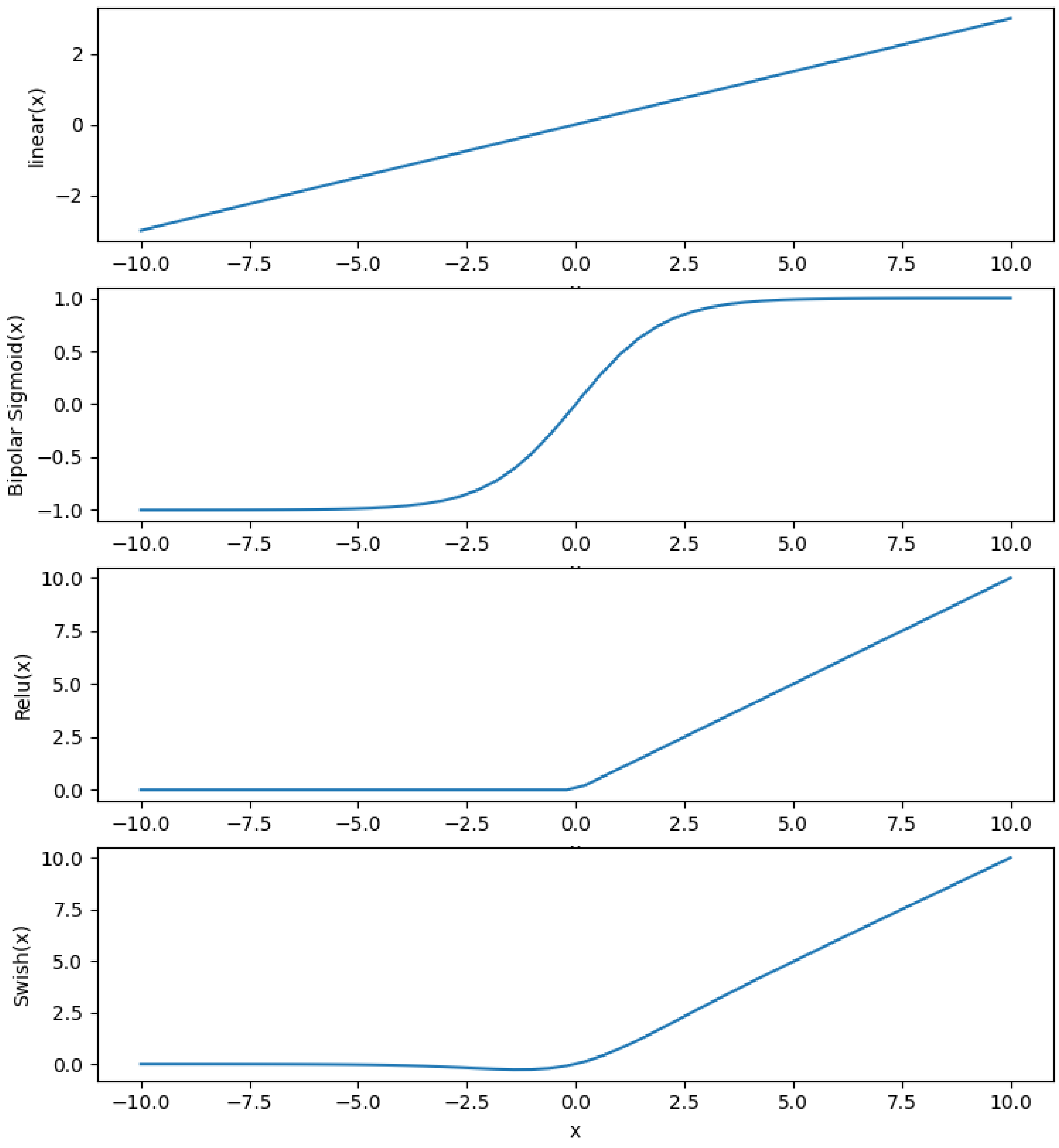
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

def linear(x):
    return 0.3*x

def sigmoid(x):
    return 1/(1 + np.exp(-x))

def Bipolar_sigmoid(x):
    return (1 - np.exp(-x))/(1 + np.exp(-x))

def tanh(x):
    return 2*sigmoid(2*x) - 1

def relu(x):
    return np.maximum(0,x)

def leaky_relu(x):
    return np.maximum(0.05*x,x)

def swish(x):
    return x/(1 + np.exp(-x))

def softmax(x):
    return np.exp(x) / np.sum(np.exp(x))
```

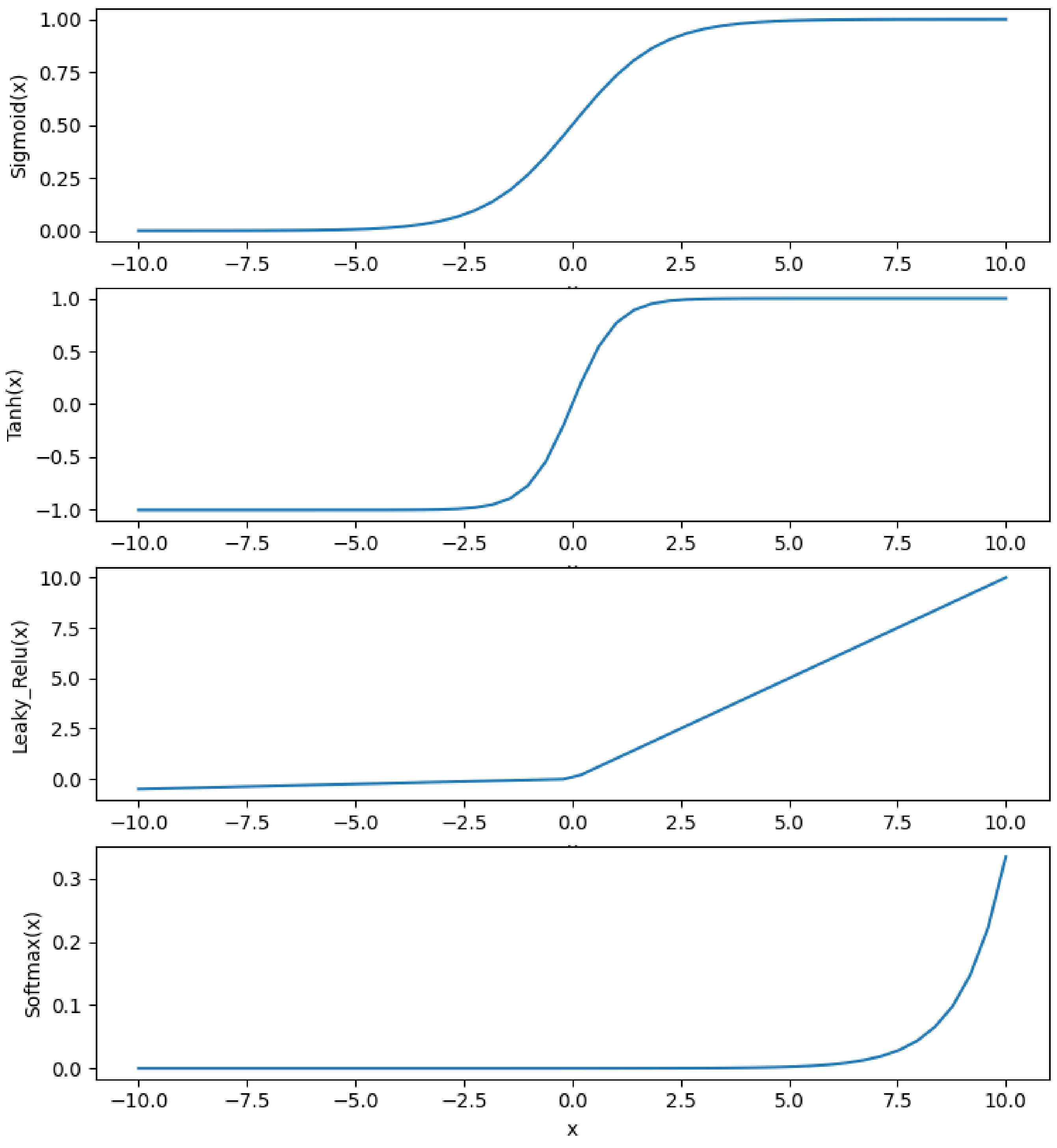
```
print("Linear(0.64) = ", linear(0.64))
print("sigmoid(0.64) = ", sigmoid(0.64))
print("Bipolar_sigmoid(0.64) = ", Bipolar_sigmoid(0.64))
print("tanh(0.64) = ", tanh(0.64))
print("relu(0.64) = ", relu(0.64))
print("leaky_relu(0.64) = ", leaky_relu(0.64))
print("swish(0.64) = ", swish(0.64))
print("softmax(0.64) = ", softmax(0.64))

x = np.linspace(-10,10,50)
print(x)

p1 = linear(x)
p2 = sigmoid(x)
p3 = Bipolar_sigmoid(x)
p4 = tanh(x)
p5 = relu(x)
p6 = leaky_relu(x)
p7 = swish(x)
p8 = softmax(x)

plt.subplot(4,2,1)
plt.xlabel("x")
plt.ylabel("linear(x)")
plt.plot(x,p1)

plt.subplot(4,2,2)
plt.xlabel("x")
plt.ylabel("Sigmoid(x)")
plt.plot(x,p2)

plt.subplot(4,2,3)
plt.xlabel("x")
plt.ylabel("Bipolar Sigmoid(x)")
plt.plot(x,p3)

plt.subplot(4,2,4)
plt.xlabel("x")
plt.ylabel("Tanh(x)")
plt.plot(x,p4)

plt.subplot(4,2,5)
plt.xlabel("x")
plt.ylabel("Relu(x)")
plt.plot(x,p5)

plt.subplot(4,2,6)
plt.xlabel("x")
plt.ylabel("Leaky_Relu(x)")
plt.plot(x,p6)

plt.subplot(4,2,7)
plt.xlabel("x")
plt.ylabel("Swish(x)")
plt.plot(x,p7)

plt.subplot(4,2,8)
plt.xlabel("x")
plt.ylabel("Softmax(x)")
plt.plot(x,p8)

plt.show()
```

Output :



```
Linear(0.64) =  0.192
sigmoid(0.64) =  0.6547534606063192
Bipolar_sigmoid(0.64) =  0.30950692121263845
tanh(0.64) =  0.5648995528462248
relu(0.64) =  0.64
leaky_relu(0.64) =  0.64
swish(0.64) =  0.4190422147880443
softmax(0.64) =  1.0
[-10.          -9.59183673  -9.18367347  -8.7755102   -8.36734694
  -7.95918367  -7.55102041  -7.14285714  -6.73469388  -6.32653061
  -5.91836735  -5.51020408  -5.10204082  -4.69387755  -4.28571429
  -3.87755102  -3.46938776  -3.06122449  -2.65306122  -2.24489796
  -1.83673469  -1.42857143  -1.02040816  -0.6122449   -0.20408163
   0.20408163   0.6122449    1.02040816   1.42857143   1.83673469
   2.24489796   2.65306122   3.06122449   3.46938776   3.87755102
   4.28571429   4.69387755   5.10204082   5.51020408   5.91836735
   6.32653061   6.73469388   7.14285714   7.55102041   7.95918367
   8.36734694   8.7755102    9.18367347   9.59183673  10.        ]
```


linear(x)
Bipolar Sigmoid(x)
Relu(x)
Swish(x)
x


Sigmoid(x)
1.00
0.75
0.50
0.25
0.00
Tanh(x)
1.0
0.5
0.0
−0.5
−1.0
Leaky_Relu(x)
10.0
7.5
5.0
2.5
0.0
Softmax(x)
0.3
0.2
0.1
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x