22/8/25

Week-5

Aim:- To study different activation functions used in deep learning, implement them, and analyze their role in introducing Non-linearity, improving learning capability.

Algorithm:-

1.) Start
2.) Initialize the Input values.
3.) Define commonly used activation functions:
   - → Sigmoid : $f(x) = \frac{1}{1+e^{-x}}$
   - → Tanh : $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
   - → ReLU (Rectified Linear unit) $f(x) = \max(0, x)$
   - → Soft max : $f(x_i) = \frac{e^{x_i}}{\Sigma_j e^{x_j}}$
4.) Apply each activation function on input data.
5.) Plot / observe the output behaviour for different ranges of input.
6.) Train a small neural Network with each activation function and compare performance.
7.) Stop.

Pseudo code:-

START
Import necessary Libraries
Define the activation functions:

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

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

$$\text{ReLU}(x) = \max(0, x)$$

$$\text{softmax}(x) = \frac{e^{x_i}}{\Sigma_j e^{x_j}}$$

Load dataset (MNIST)
For each observation function in [sigmoid, Tanh, ReLU]
Build a neural Network with that activation
Train the Network on training Data.
Evaluate on Test data
Store accuracy
compare accuracy results
END

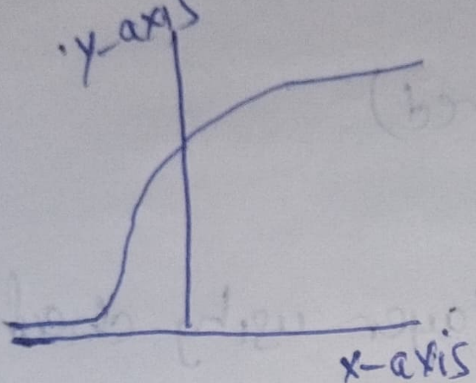
Observations :-

1.) The sigmoid activation function showed slower convergence and achieved lower accuracy compared to others.
2.) Tanh performed better than sigmoid since, it is zero centred, but still suffered from vanishing gradients.
3.) ReLU converged faster and gave accuracy more among all tested activation functions.
4.) The choice of activation function had a significant effect on both training speed and final model accuracy.

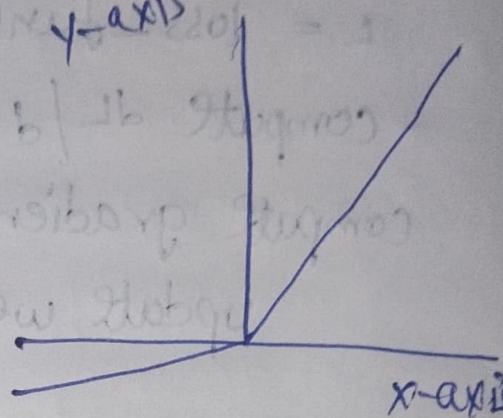
Conclusion :-

→ Activation functions introduce non-linearity and enable neural Networks to learn complex patterns.
→ sigmoid is suitable for probabilistic outputs but not efficient for deep networks.
→ Tanh improves learning over sigmoid but faces gradient issues.
→ ReLU is the most effective for hidden layers, ensuring fast convergence and high accuracy.
→ Thus, selecting the right activation function is crucial for improving deep learning model performance.

Sigmoid :- $g(x) = \frac{1}{1+e^{-x}}$



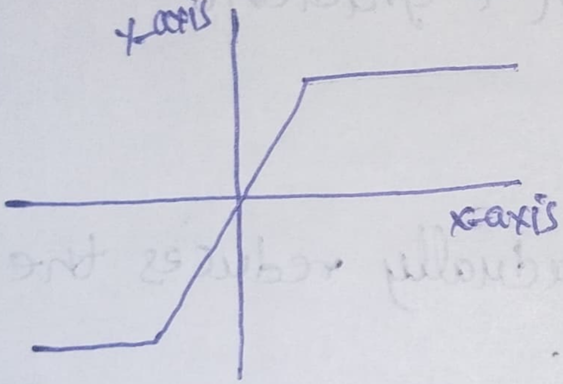
Leaky Relu :-



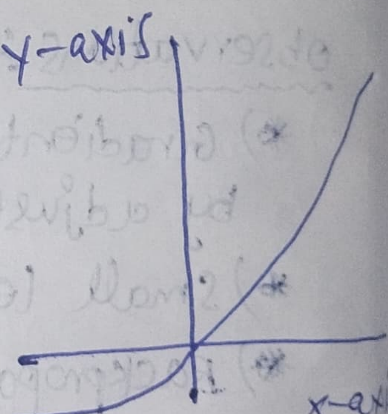
$g(x) = \max(0.1x, x)$

Tanh :-

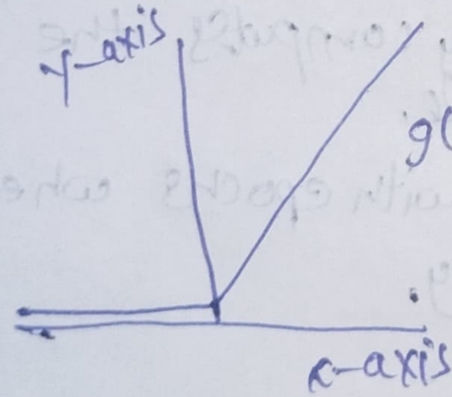
$$g(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Exponential Linear units :-

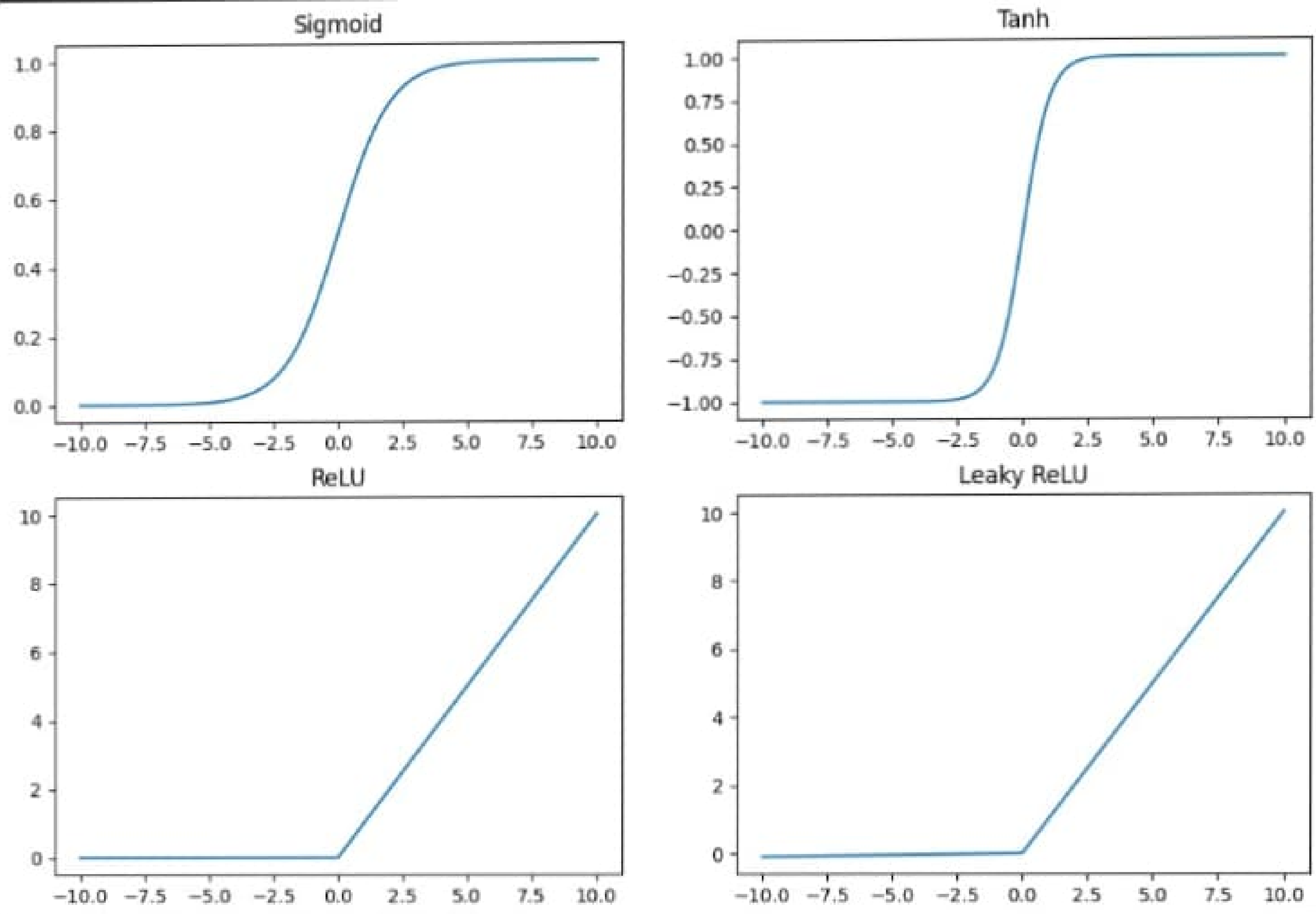


Relu :-



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

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

# Input range
x = np.linspace(-10, 10, 400)

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

# Tanh
def tanh(x):
    return np.tanh(x)

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

# Leaky ReLU
def leaky_relu(x, alpha=0.01):
    return np.where(x > 0, x, alpha * x)

# Softmax (for vector input)
def softmax(x):
    exp_x = np.exp(x - np.max(x))  # stability improvement
    return exp_x / exp_x.sum(axis=0)

# Plot functions
plt.figure(figsize=(12,8))

plt.subplot(2,2,1)
plt.plot(x, sigmoid(x))
plt.title("Sigmoid")

plt.subplot(2,2,2)
plt.plot(x, tanh(x))
plt.title("Tanh")

plt.subplot(2,2,3)
plt.plot(x, relu(x))
plt.title("ReLU")

plt.subplot(2,2,4)
plt.plot(x, leaky_relu(x))
plt.title("Leaky ReLU")

plt.show()

# Example Softmax
scores = np.array([2.0, 1.0, 0.1])
print("Softmax output:", softmax(scores))
```



Sigmoid
1.0
0.8
0.6
0.4
0.2
0.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0
Tanh
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0
ReLU
10
8
6
4
2
0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0
Leaky ReLU
10
8
6
4
2
0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0