Laboratory Report 3: Activation Functions in Neural Networks

Course: COMP-341L, Artificial Neural Networks Lab
Lab Assignment: 3
Topic: Activation Functions

Student Name:	Zarmeena Jawad
Roll Number:	B23F0115AI125
Section:	AI Red
Submission Date:	January 26, 2026

Department of Computing

Lab Report Submission

Contents

1 Executive Summary

This laboratory assignment focuses on understanding and implementing activation functions used in artificial neural networks. The work includes implementing four key activation functions (Sigmoid, ReLU, Tanh, and Leaky ReLU), visualizing their behaviors, and analyzing their properties, advantages, and limitations. The lab also covers conceptual questions about activation function selection, gradient issues, and typical applications in different layers of a network.

2 Learning Objectives

Upon completion of this lab, students should be able to:

- Implement common activation functions (Sigmoid, ReLU, Tanh, Leaky ReLU)
- Visualize and compare activation function behaviors
- Understand the vanishing gradient problem and its impact
- Identify suitable activation functions for different network layers
- Explain the dying ReLU problem and its solutions
- Understand why Softmax is preferred for multi-class classification

3 Methodology

All experiments were conducted using Python 3 with NumPy and Matplotlib. Each activation function was implemented using its mathematical formulation, tested on sample values, and plotted to visualize its characteristics. Conceptual analysis was then written based on observed behavior and known training dynamics in neural networks.

4 Task 1: Activation Functions Implementation and Visualization

4.1 Purpose

Implement and visualize four fundamental activation functions used in neural networks, and understand their mathematical properties, output ranges, and use cases.

4.2 Activation Functions

4.2.1 Sigmoid (Logistic) Function

Mathematical Formula:

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

Derivative:

$$f'(x) = f(x)\big(1 - f(x)\big)$$

Key Properties:

- Output range: $[0, 1]$
- Smooth S-shaped curve
- Maximum derivative is 0.25 at $x = 0$
- Common use: binary classification output layers
- Limitation: vanishing gradients for large $|x|$

Implementation (Python):

```python
def sigmoid(x):
    x_clipped = np.clip(x, -500, 500)
    return 1.0 / (1.0 + np.exp(-x_clipped))
```

4.2.2 ReLU (Rectified Linear Unit)

Mathematical Formula:

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

Derivative:

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Key Properties:

- Output range: $[0, \infty)$
- Computationally efficient and piecewise linear
- Common use: hidden layers
- Limitation: dying ReLU (neurons can become permanently inactive)

Implementation (Python):

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

4.2.3 Tanh (Hyperbolic Tangent)

Mathematical Formula:

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

Derivative:

$$f'(x) = 1 - (f(x))^2$$

Key Properties:

- Output range: $[-1, 1]$
- Zero-centered and symmetric about the origin

- Maximum derivative is 1.0 at $x = 0$
- Common use: hidden layers (often better gradients than sigmoid)
- Limitation: still can suffer from vanishing gradients for large $|x|$

Implementation (Python):

```
def tanh(x):
    return np.tanh(x)
```

4.2.4 Leaky ReLU (Parametric ReLU)

Mathematical Formula:

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases} \quad \text{(typically } \alpha = 0.01\text{)}$$

Derivative:

$$f'(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$$

Key Properties:

- Output range: $(-\infty, \infty)$ (negative allowed), and $[0, \infty)$ for positive region
- Small non-zero slope for negative inputs
- Common use: hidden layers
- Advantage: reduces dying ReLU risk by keeping gradients non-zero for $x \leq 0$

Implementation (Python):

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

4.3 Results

4.3.1 Test Values and Outputs

Table 1: Outputs of activation functions on sample inputs

Input	Sigmoid	ReLU	Tanh	Leaky ReLU
-2.0	0.1192	0.0000	-0.9640	-0.0200
-1.0	0.2689	0.0000	-0.7616	-0.0100
0.0	0.5000	0.0000	0.0000	0.0000
1.0	0.7311	1.0000	0.7616	1.0000
2.0	0.8808	2.0000	0.9640	2.0000

4.3.2 Key Observations

- Sigmoid maps inputs to $[0, 1]$ and saturates for large negative or large positive values.

- ReLU outputs zero for negative inputs and grows linearly for positive inputs.
- Tanh is zero-centered, maps to $[-1, 1]$, and saturates similarly to sigmoid.
- Leaky ReLU behaves like ReLU for $x > 0$, but preserves a small slope for $x \leq 0$.

4.4 Visualizations

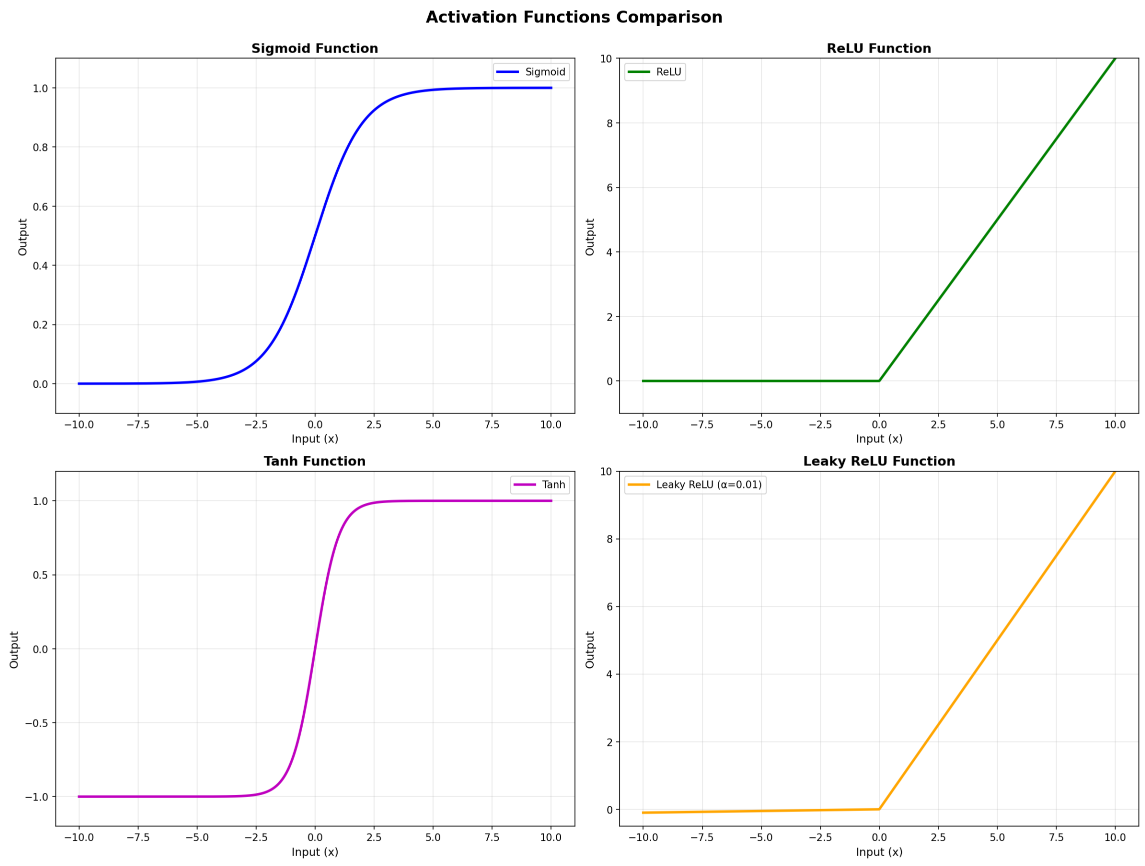

Figure 1: Comparison of Sigmoid, ReLU, Tanh, and Leaky ReLU activation functions

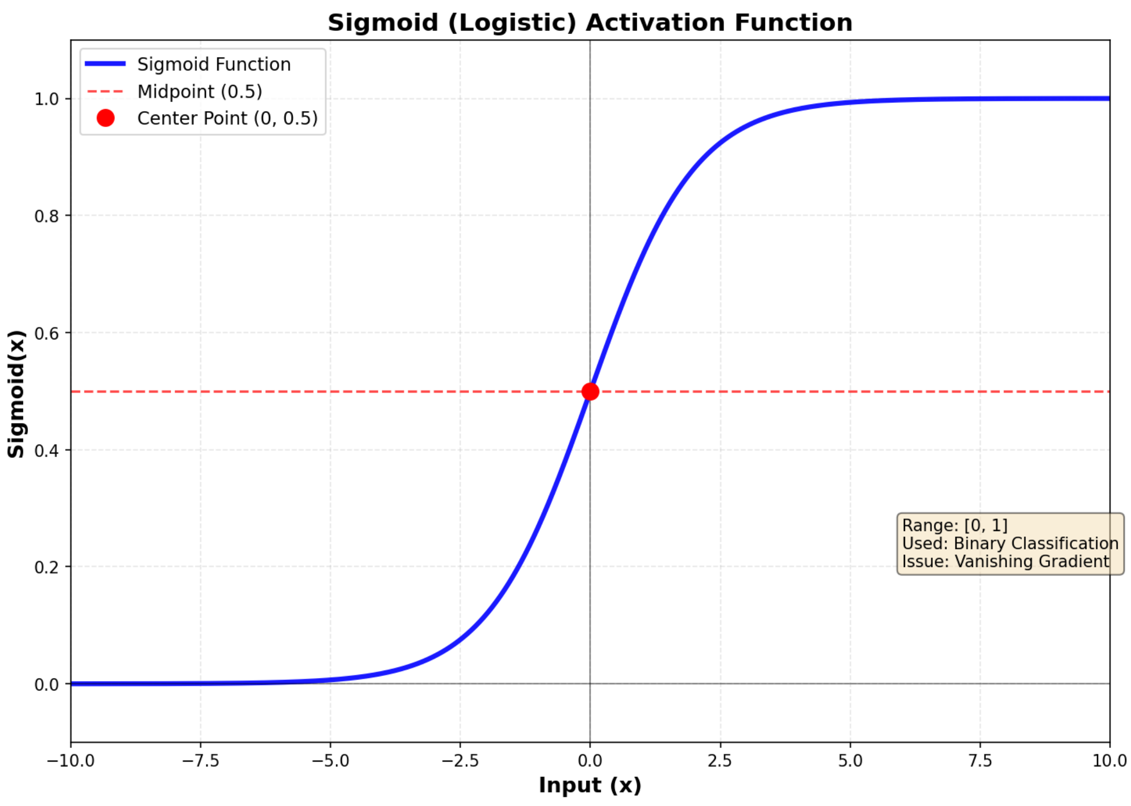

Figure 2: Sigmoid (Logistic) activation function

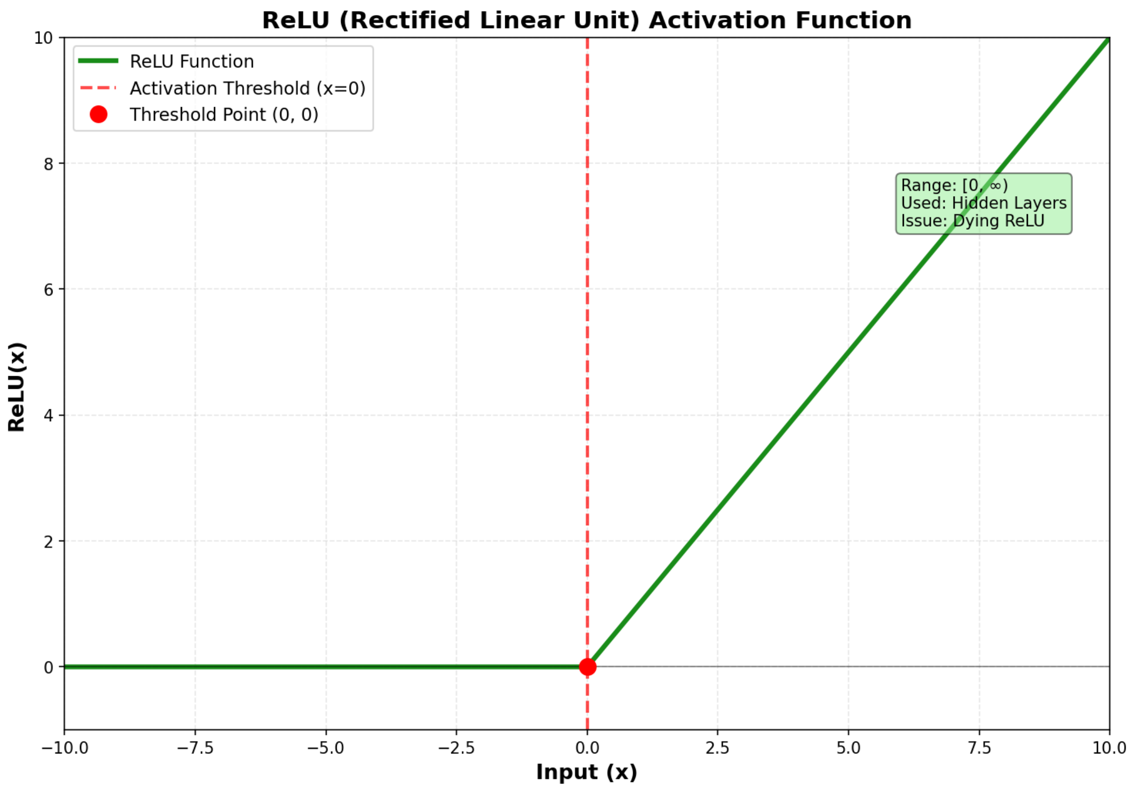

Figure 3: ReLU activation function

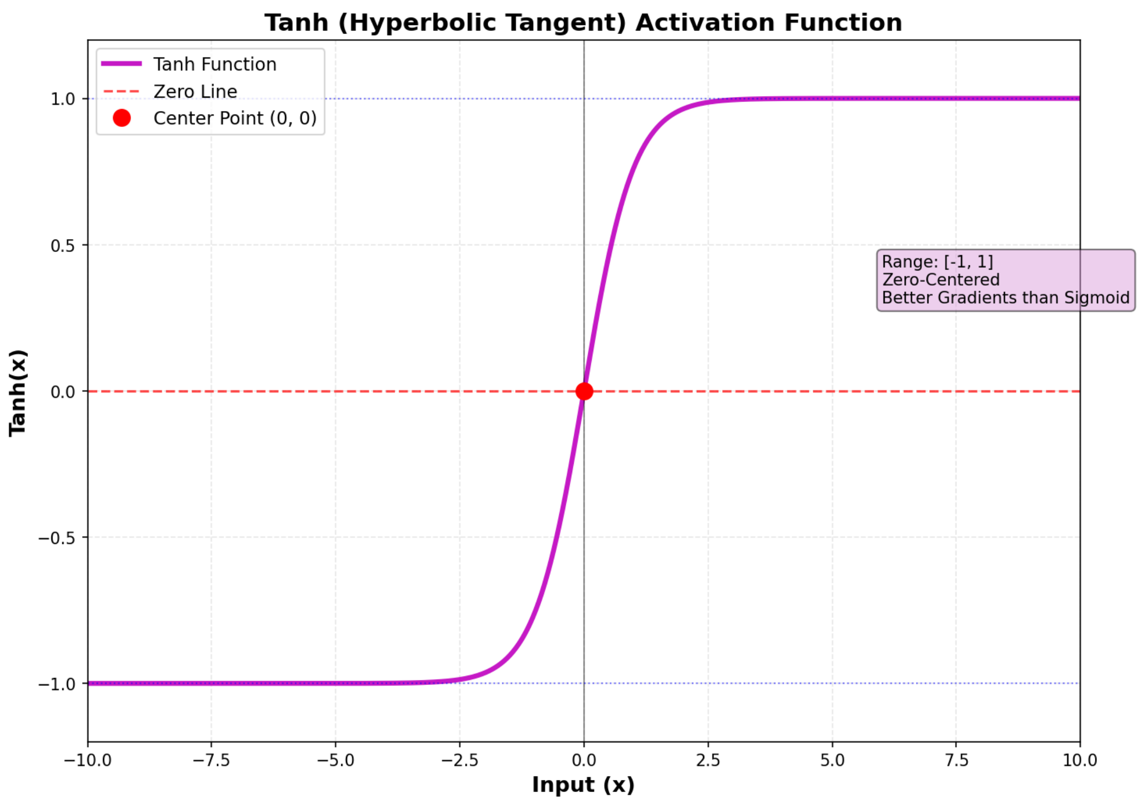

Figure 4: Tanh activation function

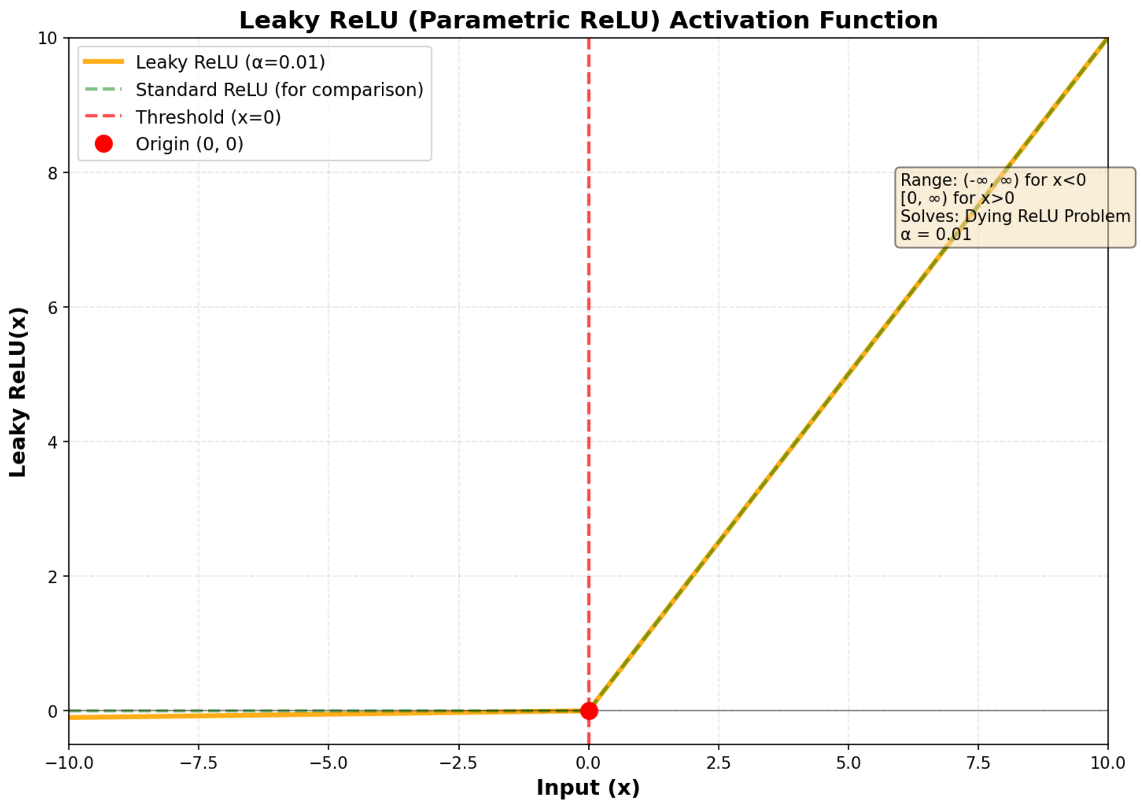

Figure 5: Leaky ReLU (Parametric ReLU) activation function

5 Task 2: Conceptual Analysis of Activation Functions

5.1 Question 1: Most Suitable Activation Function for Hidden Layers

Answer: ReLU (Rectified Linear Unit) is most suitable for hidden layers.

Reasons:

1. **Computational efficiency:** It is a simple threshold operation, which makes training faster.
2. **Gradient behavior:** For active neurons ($x > 0$), the gradient is constant (1), which helps deep networks train better.
3. **Sparsity:** Negative inputs produce zeros, which can create sparse activations and may reduce overfitting.
4. **Practical effectiveness:** ReLU has become a standard default in many deep learning architectures due to strong empirical performance.

5.2 Question 2: Vanishing Gradient Problem

Definition: The vanishing gradient problem occurs when gradients become extremely small during backpropagation, especially in deep networks. As gradients are repeatedly multiplied through layers, they can shrink exponentially, causing early layers to learn very slowly.

Consequences:

- Early layers receive tiny updates and fail to learn useful feature representations
- Training becomes very slow, or the network stops improving
- Deep networks become difficult to train without careful design choices

Activation functions affected:

- **Sigmoid:** severely affected due to saturation and small maximum derivative (0.25).
- **Tanh:** moderately affected, better than sigmoid near zero, but still saturates for large $|x|$.
- **ReLU:** not affected for $x > 0$, but has zero gradient for $x \leq 0$.
- **Leaky ReLU:** mitigates both vanishing gradients (in practice) and dead neurons by using a small non-zero slope for $x \leq 0$.

5.3 Question 3: ReLU Issue and Leaky ReLU Solution

Problem: Dying ReLU happens when a neuron receives mostly negative inputs, so ReLU outputs zero and the gradient becomes zero. With zero gradient, weights stop updating, so the neuron may never reactivate.

Leaky ReLU solution: Leaky ReLU provides a small gradient α for negative inputs. This keeps learning active even when inputs are negative, allowing neurons to recover.

Table 2: ReLU vs Leaky ReLU for negative inputs

Aspect	**ReLU**	**Leaky ReLU**
Output for $x < 0$	0	αx
Gradient for $x < 0$	0	α
Neuron recovery	unlikely	possible
Risk of dead neurons	higher	lower

5.4 Question 4: Why Softmax in Output Layers?

Softmax is preferred for multi-class classification because it converts raw model outputs (logits) into a probability distribution:

- Each class probability is in $[0, 1]$
- All probabilities sum to 1, which makes interpretation straightforward
- It models competition between classes naturally
- It is differentiable, enabling gradient-based learning

Given logits z_i, Softmax is:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

A numerically stable implementation typically subtracts $\max(z)$ before exponentiation to reduce overflow risk.

6 Results Summary

6.1 Task 1

- Implemented Sigmoid, ReLU, Tanh, and Leaky ReLU activation functions in Python.
- Tested on sample inputs and recorded outputs in a results table.
- Generated individual plots and a combined comparison plot for visual analysis.

6.2 Task 2

- Identified ReLU as a strong default for hidden layers and justified the choice.
- Explained vanishing gradients and listed which functions are most affected.
- Discussed dying ReLU and showed how Leaky ReLU mitigates it.
- Explained why Softmax is used for multi-class output layers.

7 Conclusion

This lab provided a practical and conceptual understanding of activation functions in neural networks. Sigmoid is useful for probability outputs in binary classification but often suffers from vanishing gradients in deep networks. Tanh improves zero-centering and gradients near the origin, but still saturates for large magnitudes. ReLU is widely used in hidden layers due to simplicity and effective gradient flow for positive inputs, although it can suffer from dead neurons. Leaky ReLU addresses this by maintaining a small gradient for negative inputs, improving robustness during training. For multi-class classification, Softmax is preferred in the output layer because it produces a valid probability distribution across classes.

8 References

1. Lab Manual: Lab 03, Activation Functions
2. NumPy Documentation: https://numpy.org/doc/stable/
3. Matplotlib Documentation: https://matplotlib.org/stable/contents.html