CS898BD Assignment 2: ReLU vs Tanh Activation Function – A Comparative Study

Andrew Lisenby
15 October 2025
CS898BD – Deep Learning
Dr. Lokesh Das

Abstract. This report presents an experimental comparison of ReLU and Tanh activation functions using a custom 4-layer Convolutional Neural Network (CNN) on the CIFAR-10 dataset [1]. The study models the performance of both activation functions in their ability to achieve the target 25% training error. It was determined that ReLU significantly outperforms Tanh in terms of convergence speed, generalization ability, and final test performance.

1. Introduction and Motivation

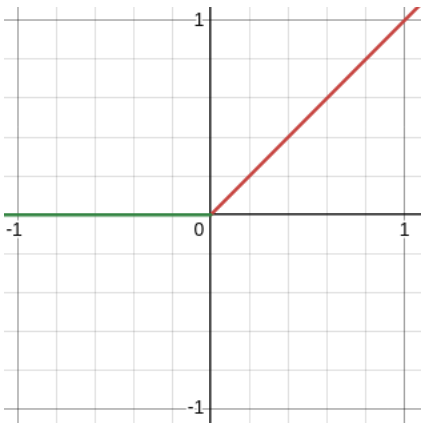
1.1 Introduction to Activation Functions

At the heart of any Neural Network is an activation function, without it our network is unable to transform non-linear input data into a probabilistic estimate. It is because of an activation function that we can perform back-propagation in order to adjust the neural network's weights and generalize data. The two well known activation functions at the focus of this experiment are Rectified Linear Unit (ReLU) and Hyperbolic Tangent (Tanh).

1.1.1 Rectified Linear Unit (ReLU)

ReLU is a piecewise function comprised of two linear functions that can be expressed as a simple *MAX* operation as follows:

$$f(x) = max(0, x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

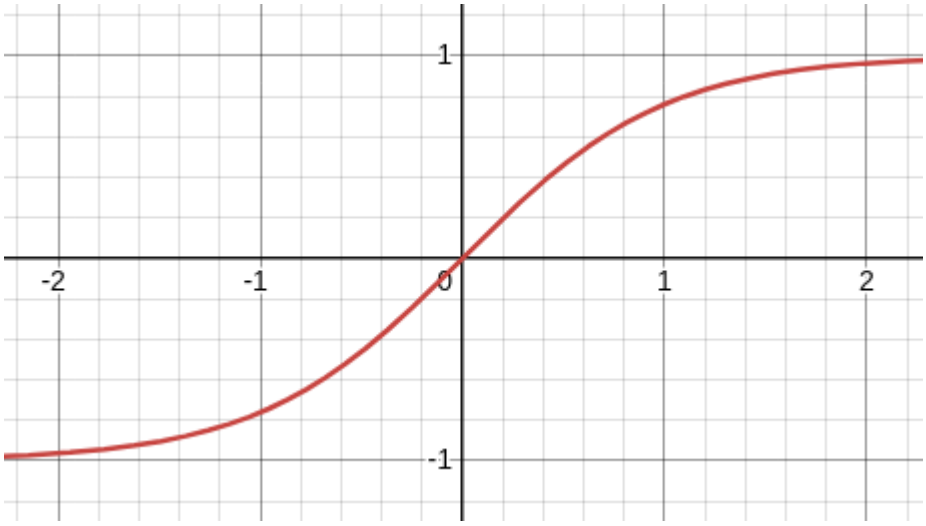


For positive values of x, the output is simply x, but if the input becomes negative, the output becomes zero. This leads to very quick convergence and efficient backpropagation, but is susceptible to the "Dying ReLU" problem, where the gradients get stuck in a zero state during backpropagation and fail to update the neuron's weights, effectively disabling the neuron.

1.1.2 Hyperbolic Tangent (Tanh)

Tanh on the other hand, is an asymptotic function with a limit of positive and negative one as it approaches positive and negative infinity respectively. It's simplest representation (without getting too far into the weeds) is as follows:

$$f(x) = \tanh(x)$$



This has a distinct advantage over ReLU in that it has a symmetric activation about zero. It is not prone to the same issues with negative inputs as ReLU; However, it suffers its own issue called "The Vanishing Gradient Problem". Very large or small values of x bring Tanh close to its asymptotes, resulting in a gradient very close to zero, and causing learning to slow or outright halt.

1.2 Motivation

The primary objective was to empirically compare ReLU and Tanh activation functions in a customized CNN architecture, analyzing their impact on:

- Convergence speed
- Generalization performance
- Architecture design
- Optimization stability

2. Methodology

2.1 Dataset

CIFAR-10 Dataset [1]:

- Training samples: 45,000
- Validation samples: 5,000
- Test samples: 10,000
- Image dimensions: 32×32×3
- Classes: 10

2.2 Model Architecture

The model architecture was highly experimental, it took several variations and trials to find an architecture that wasn't prone to vanishing gradient with Tanh. The final optimized CNN architecture used was constructed as follows:

Conv1

- Input channels: 3
- Output channels: 32
- Kernel Size: 3x3
- Stride=1
- Padding=1
- Normalization: BatchNorm [2]
- Activation: True

Conv2:

- Input channels: 32
- Output channels: 64
- Kernel Size: 3x3
- Stride=1
- Padding=1
- Normalization: BatchNorm [2]
- Activation: True

Conv3:

- Input channels: 64
- Output channels: 128
- Kernel Size: 3x3
- Stride=1
- Padding=1
- Normalization: BatchNorm [2]
- Activation: True

Conv4

- Input channels: 128
- Output channels: 128
- Kernel Size: 3x3
- Stride=1
- Padding=1

- Normalization: BatchNorm [2]
- Activation: True

MaxPool
- Kernel: 2x2
- Stride=2
- Output channels: 128
- Output dimensions: 16x16
- Activation=False

Flatten
- Features: 128x16x16 → 32,768

FC1
- Input channels: 32,768
- Output channels: 512
- Activation: True

FC2
- Input channels: 512
- Output channels: 128
- Activation: True

FC3
- Input channels: 128
- Output classes: 10
- Activation: False

Trials studied the impact of various design concepts:
- Large/Small kernel sizes
- Downsampling rate
- Combinations of maxpooling at varying layers
- With and without batch normalization [2]
- Feature size output to the dense layers

2.3 Training Configuration

- Optimizer: Adam (lr=0.001) [3]
- Loss Function: Cross-Entropy Loss [4]
- Batch Size: 64
- Target: Stop when training error ≤ 25%
- Maximum Epochs: 100
- Hardware: Nvidia RTX 5070 TI 16GB

2.4 Experimental Design

This is the architecture found to be successful after approximately 20 experimental training cycles. Most of these attempts were early stopped when Vanishing Gradient became apparent.

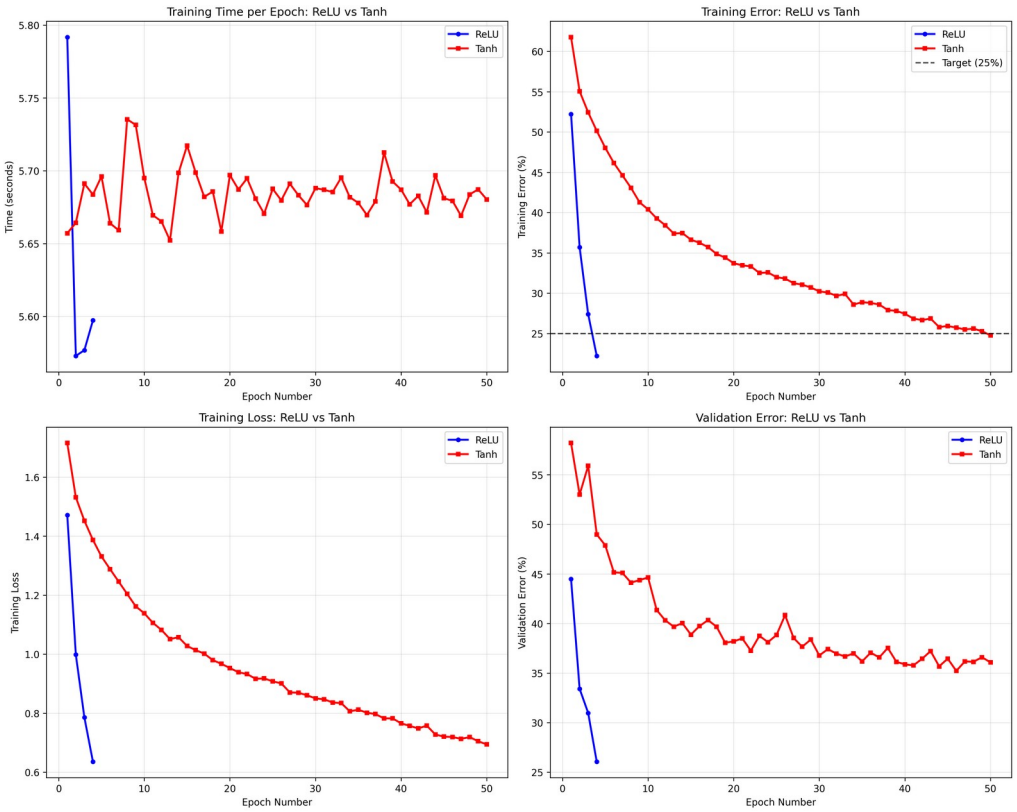
Architecture Evolution Process:
- Baseline: Simple CNN with small kernels, multiple pooling operations
- Batch Normalization [2]: Testing impact on Tanh training
- Pooling Optimization: Single vs multiple pooling operations
- Spatial Resolution Tuning: Testing 4x4, 8x8, 16x16, and 32x32 final feature maps

- Kernel Size Experiments: Comparing 3x3, 5x5, and 7x7 kernels

3 Experimental Results

3.1 Activation Performance

The final performance comparison between ReLU and Tanh activation functions revealed a clear difference in all recorded metrics.



Metric	ReLu	Tanh	ReLU Advantage
Final Training Error	22.88%	24.88%	2.00%
Final Validation Error	26.12%	36.36%	10.2%
Test Accuracy	74.44%	62.99%	11.45%
Test Error	25.56%	37.01%	-11.45%
Test Loss	0.7509	1.1057	32.09%
Training Epochs to Target	4	51	47
Total Training Time (seconds)	22.5	289.1	266.6
Average Time Per Epoch (seconds)	5.63	5.67	0.04

This demonstrates while both activation functions were able to reach the target training error of 25%, ReLU dominated Tanh in every recorded metric. ReLU converged in just 9.2% of the time

of Tanh, it had a 11.45% better Test Accuracy and it was 32.09% more confident in its inferences.

3.2 Key Experimental Findings

3.2.1 Impact of Batch Normalization [2]

Batch normalization proved to be critical for Tanh to converge to the target error. Early trials without batch normalization resulted in failure, with Tanh convergence stalling at a training error greater than 46%. This strongly suggests that the activation function became saturated and the gradients approached zero during back-propagation.

3.2.2 Spatial Resolution Optimization

Experimentation with feature map sizes at different stages revealed a delicate balancing act between spatial compression and preservation. Early aggressive downsampling led to excessive loss of spacial data, while preserving spacial data without downsampling resulted in a complex model that struggled to converge using Tanh. Downsampling to a 16x16 feature map using maxpooling just before the fully connected layers proved to offer the optimal balance for both activation functions.

3.2.3 Pooling Strategy

Implementing pooling was only successful after convolution 4. All other attempts led to Tanh failing to converge.

3.2.4 Kernel Size

Attempts to optimize the kernel size through progressively smaller kernel sizes from a larger kernel, such as 7x7 or 5x5, didn’t prove fruitful. Tanh appears to perform better with smaller kernel sizes regardless of the stride used in the operation.

4 Conclusion

The results of this experiment suggest through empirical evidence that the ReLU activation function is superior for this style of CNN architecture. ReLU achieved a >12x reduction in convergence time and 11.45% better test accuracy while proving to be much more robust in many different architectures – something that cannot be said about Tanh. While both activations were eventually able to reach the target error, the architectural sensitivity of Tanh and slow convergence would make it impractical and difficult to tune for in many applications.

References

[1] A. Krizhevsky, “Learning multiple layers of features from tiny images,” M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[2] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, Lille, France, Jul. 2015, pp. 448–456.

[3] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.