Scientific Research Publishing

Journal of Computer and Communications, 2025, 13(11), 128-162
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Vanishing Gradients in an Artificial Deep Neural Network: The Impact of Activation Functions and Optimization Techniques

Baron Ntambwe

Department of Computer Science, University of East London, London, UK
Email: baronbabutuke@gmail.com

How to cite this paper: Ntambwe, B. (2025) Vanishing Gradients in an Artificial Deep Neural Network: The Impact of Activation Functions and Optimization Techniques. *Journal of Computer and Communications*, **13**, 128-162.
https://doi.org/10.4236/jcc.2025.1311009

Received: August 27, 2025
Accepted: November 22, 2025
Published: November 25, 2025

Abstract

Artificial deep neural networks (ADNNs) have become a cornerstone of modern machine learning, but they are not immune to challenges. One of the most significant problems plaguing ADNNs is the vanishing gradient problem, which hinders their ability to learn and generalize. Activation functions and optimization techniques play a crucial role in mitigating or aggravating this issue. This study compares how two activation functions (Sigmoid, ReLU) and two optimizers (SGD, Adam) affect the vanishing-gradient phenomenon in a 12-layer neural network trained on MNIST. Gradient-norm ratios, mean gradients, loss, and accuracy are tracked across 10 epochs for each of four activation-optimizer combinations. Results indicate that ReLU with Adam maintains the most stable gradient flow and achieves the highest accuracy, whereas both Sigmoid and SGD combinations exhibit severe vanishing or exploding gradients. The author concludes that activation-optimizer synergy is critical and recommends ReLU with Adam for deep networks.

Keywords

Vanishing Gradient, Sigmoid, Optimization, ADNN, ReLU, SGD, ADAM

1. Introduction

The vanishing gradient problem remains a significant challenge in training deep artificial neural networks, often leading to slow convergence and suboptimal performance, especially in very deep architectures. Activation functions and optimization techniques play a critical role in influencing how gradients are propagated during backpropagation. While sigmoid and ReLU are widely used activation functions, they differ significantly in how they handle gradient flow. Similarly,

stochastic gradient descent (SGD) and Adam are popular optimization algorithms that affect how gradients are updated and can interact differently with activation functions in deep networks.

Despite extensive use, there is limited empirical evidence comparing the combined effects of these activation functions and optimization techniques on the vanishing gradient problem. Most existing studies evaluate these components in isolation, leaving a gap in understanding their interaction and collective impact on training deep networks.

Even though it is widely acknowledged that ReLU helps mitigate the vanishing gradient problem compared to sigmoid, there remains a notable lack of systematic, controlled experiments directly comparing ReLU and sigmoid under identical architectures, initializations, and training protocols [1]. Reviews such as [2] remark on "limited empirical and theoretical evidence for Sigmoid's competitors." They explicitly state that comparisons lack theoretical rigor and empirical depth across activation functions. Although this survey evaluates multiple activation functions including sigmoid and ReLU, it primarily offers a descriptive comparison, not controlled evaluations isolating activation choices. Empirical results often differ in architectures, datasets, or settings—hindering direct conclusions about activation-specific effects. Furthermore, smaller-scale studies like [3] are also constrained to specific datasets like MNIST and limited depth, with no systematic control of architecture and optimizer across conditions to isolate activation effects in deeper or diverse tasks.

This research seeks to address this gap by quantitatively analyzing how the choice of activation function (sigmoid vs. ReLU) and optimization algorithm (SGD vs. Adam) affects the magnitude and flow of gradients in a controlled architecture and optimization of deep neural networks. By systematically measuring gradient behavior, convergence rates, and performance outcomes, the study aims to provide practical insights into mitigating vanishing gradients through architectural and algorithmic choices.

2. Research Questions

This study aims to investigate the impact of two widely used activation functions (Sigmoid and ReLU) and two popular optimization techniques (Stochastic Gradient Descent (SGD) and Adam) on the vanishing gradient problem in DNNs. Therefore, as a result of the experiment, we will attempt to answer the following five questions at the end of this research project:

1. How does the choice of activation function (Sigmoid vs. ReLU) affect the occurrence and severity of vanishing gradients in deep artificial neural networks?

2. What are the impacts of optimization algorithms (SGD vs. Adam) on gradient flow and the mitigation of vanishing gradients during training?

3. What is the interaction effect between activation functions (Sigmoid and ReLU) and optimization techniques (SGD vs. Adam) that influences the vanishing gradient problem in deep neural networks?

4. How do different combinations of activation functions and optimization algorithms affect training convergence speed and final model performance (e.g., accuracy, loss)?

5. Which combination of activation function and optimization algorithm is most effective in minimizing vanishing gradients in deep network architectures?

3. Foundational Knowledge

3.1. Previous Work

3.1.1. Causes of the Vanishing Gradient Problem

The vanishing gradient problem remains a fundamental challenge in training deep neural networks, particularly when using activation functions that cause gradients to diminish exponentially as they are propagated backward through layers. This issue has significant implications for the learning capacity of deep models and is heavily influenced by both the activation function and the optimization algorithm used.

Early neural networks predominantly used the sigmoid activation function due to its smooth, differentiable curve, mapping inputs to an output between 0 and 1. However, it was soon discovered that the sigmoid function suffers from saturation in both positive and negative regions of its domain, resulting in gradients that tend toward zero as illustrated in **Figure 1**, and making it difficult for weights in early layers to update effectively [4]. This phenomenon is now well-known as the vanishing gradient problem. The gradient of the sigmoid function becomes very small for large input values, thereby impeding effective learning in deep networks [5].

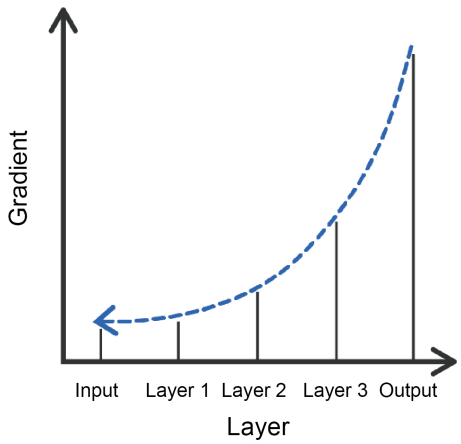

Figure 1. Vanishing gradient.

During backpropagation, the gradients of the loss function with respect to the weights are calculated using the chain rule. When the gradients flow backward through the network, the derivatives of the activation functions are multiplied together. If the activation functions are sigmoid or tanh, the small derivatives can cause the gradients to vanish. **Figure 2** illustrates this phenomenon.

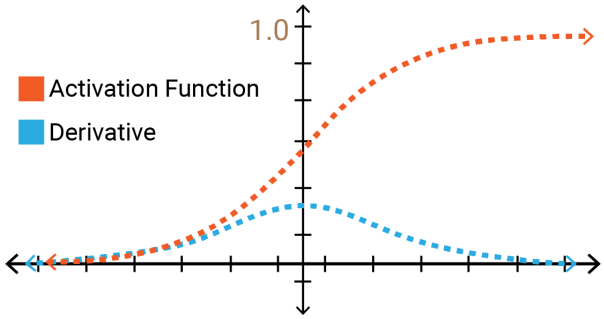

Figure 2. Vanishing gradient—activation function derivative.

Suppose we have a neural network with multiple layers using sigmoid activation functions. The derivative of the sigmoid function is typically small, around 0.25. If we have 5 layers, the gradient of the loss function with respect to the weights in the first layer would be multiplied by (0.25)^5, resulting in a very small value. This can cause the weights in the earlier layers to be updated very slowly, leading to the vanishing gradient problem.

To address this issue, Rectified Linear Units (ReLU) were introduced and quickly became the default activation function in modern deep learning due to their simplicity and effectiveness in mitigating vanishing gradients [6]. The ReLU function is defined as f(x) = max(0, x), which allows gradients to remain large and active for positive input values. While ReLU introduces its own challenge—the "dying ReLU" problem where neurons can become inactive—it generally maintains gradient flow better than sigmoid in deep architectures.

Simultaneously, advances in optimization algorithms have also played a role in addressing vanishing gradients. The Adam optimizer, introduced by [5], combines the advantages of RMSProp and momentum by using adaptive learning rates for each parameter, which improves convergence speed and stability, especially in the presence of sparse or noisy gradients. Although Adam was not designed explicitly to counter vanishing gradients, its adaptive nature often results in better performance in deep networks where vanishing or exploding gradients are present.

Despite these developments, there is a notable research gap in systematic studies that compare the impact of ReLU and sigmoid activation functions under identical experimental conditions, particularly in combination with different optimizers such as SGD and Adam. Reviews by [2] [7] highlight that while ReLU's advantages are empirically supported, most comparisons between activation functions and optimization strategies lack rigorous control over confounding factors such as network depth, initialization, and batch normalization. These limitations make it difficult to isolate the precise contribution of activation and optimization choices to the vanishing gradient problem.

Furthermore, theoretical work by [8] shows that sigmoid networks can maintain gradient flow if initialized using orthogonal methods, suggesting that the choice of activation function may not be solely responsible for vanishing gradi-

ents. However, this finding has not been extensively validated in practical, large-scale models using modern optimizers like Adam.

In conclusion, while ReLU and Adam have become standard components of modern neural networks, the interaction between activation functions and optimizers—and their joint impact on the vanishing gradient problem—remains insufficiently explored. A more comprehensive, controlled evaluation is necessary to understand these effects and to provide evidence-based guidelines for model design in deep learning.

3.1.2. Mitigating the Vanishing Gradient Problem

1. Use of Non-Saturating Activation Functions

Several techniques have been proposed to address the vanishing gradient problem. One of the earliest and most effective techniques involves replacing saturating functions like sigmoid and tanh with non-saturating alternatives, such as the Rectified Linear Unit (ReLU). Unlike sigmoid, whose gradients vanish for large positive or negative inputs, ReLU maintains a constant gradient of 1 for positive values, thus allowing gradients to propagate more effectively in deep architectures [1]. Variants like Leaky ReLU and ELU further address the issue of dying neurons while maintaining non-saturation characteristics [9].

2. Proper Weight Initialization

Weight initialization plays a crucial role in controlling the magnitude of gradients during training. Techniques such as Xavier initialization [10] and He initialization [11] are designed to maintain the variance of activations and gradients across layers. These methods set initial weights based on the number of input and/or output units in each layer, which prevents early layer gradients from vanishing or exploding.

3. Batch Normalization

Batch normalization, introduced by Ioffe and Szegedy [12], normalizes the inputs of each layer to have zero mean and unit variance. This helps to stabilize the distribution of activations throughout training, reducing internal covariate shift and allowing for faster convergence. Batch normalization has been shown to maintain more consistent gradient flow, thus alleviating the vanishing gradient problem even when using activation functions like sigmoid or tanh.

4. Residual Connections (Skip Connections)

The introduction of residual networks (ResNets) [13] marked a significant breakthrough in deep learning. Residual connections allow gradients to bypass several layers by directly adding the input of a layer to its output. This ensures that gradients can flow back more easily, even through very deep networks, effectively mitigating vanishing gradients and enabling the training of networks with hundreds of layers.

5. Gradient Clipping and Normalization Techniques

While more common in recurrent neural networks (RNNs), gradient clipping is also used to stabilize training in deep feedforward networks. By limiting the maximum gradient magnitude, this technique can prevent exploding gradients,

which often co-occur with vanishing ones [14]. Additionally, layer normalization and weight normalization have been explored as alternatives to batch normalization, offering similar gradient-stabilizing effects, especially in recurrent architectures.

6. Adaptive Optimization Algorithms

Optimizers such as Adam and RMSProp adjust learning rates dynamically for each parameter, often helping maintain effective updates even when gradients are small [6]. Although not designed to directly solve vanishing gradients, their adaptive mechanisms improve training stability and convergence in deep models that might otherwise struggle with poor gradient flow.

3.2. Sigmoid vs. ReLU Activation Functions

Sigmoid and ReLU are two widely used activation functions in deep learning models. Therefore, in the course of this research, we will focus solely on them. We will keep the scope of this comparative section within the boundaries of these two activation functions while emphasizing performance and the vanishing gradient problem, which are critical aspects in training deep neural networks. **Figure 3** below provides their mathematical representation.

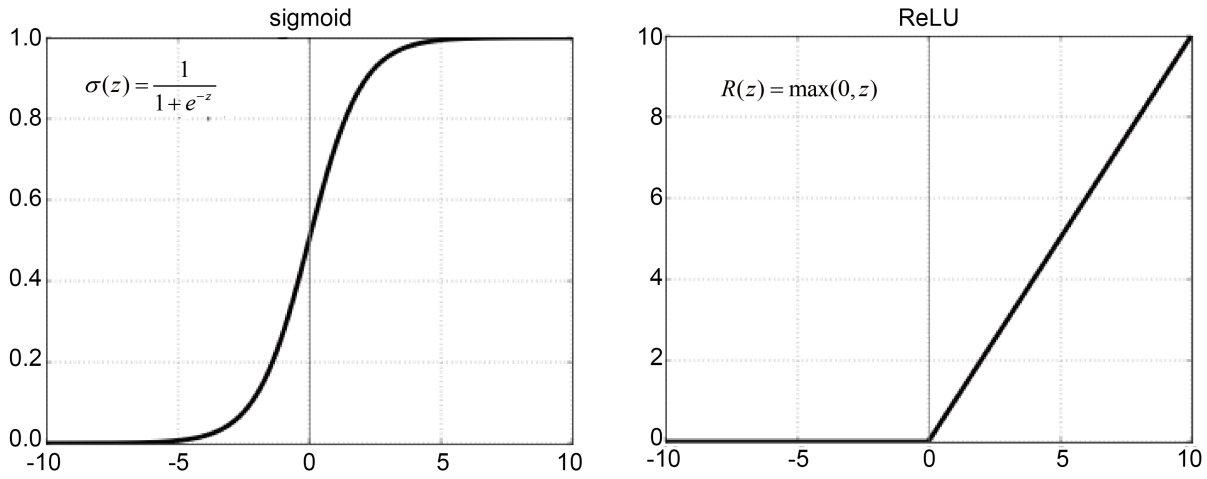

Figure 3. Sigmoid vs. ReLU activation functions.

Several scholarly studies have compared the performance of sigmoid versus ReLU activation functions in DANNs under different contexts. However, most of them converged to the supremacy of ReLU over sigmoid. For example, in a study conducted by Xu using a deep neural network trained on the MNIST dataset, it was proven that ReLU performed better than sigmoid [15]. In another experiment, it was discovered that ReLU outperformed sigmoid in a neural network trained on a natural language processing task [16].

In the sigmoid function, as the input becomes very positive or very negative, the output saturates near 1 or 0, respectively. This causes the gradient (derivative) to be very small and contributes to the vanishing gradient problem. This slows

down or stalls learning in deep networks, especially for earlier layers.

ReLU, on the other hand, has a gradient of 1 for $x > 0$, causing it to avoid vanishing gradients for positive inputs. However, it can suffer from the "dying ReLU" problem: if many neurons output $x \leq 0$, they produce 0 gradients (this can be mitigated using Leaky ReLU or PReLU).

As many researchers suggest, ReLU is generally a better choice than sigmoid for deep learning models due to its ability to avoid vanishing gradients and its computational efficiency. However, sigmoid can still be useful in certain applications, such as binary classification problems. The choice of activation function ultimately depends on the specific problem and the characteristics of the data.

3.3. SGD vs. Adam Optimization

While both Adam and Stochastic Gradient Descent (SGD) are foundational optimization algorithms in deep learning, their effectiveness in mitigating the vanishing gradient problem differs in practice.

SGD updates weights using stochastic samples, which introduces noise and may help escape flat regions of the loss surface; however, it is often sensitive to vanishing gradients, particularly when used with saturating activation functions like sigmoid or tanh [5]. Unless carefully tuned with momentum or learning rate schedules, SGD alone struggles to propagate gradients effectively in very deep networks.

In contrast, Adam uses adaptive learning rates and momentum estimates, allowing it to maintain relatively strong parameter updates even when gradients are small. This makes Adam more robust in networks affected by vanishing gradients, especially in the early stages of training. Additionally, empirical studies show that Adam often converges faster and with greater stability compared to SGD in complex or deep architectures [17]. However, while Adam mitigates the effects of vanishing gradients better in many cases, it can lead to poorer generalization compared to SGD in some tasks [18].

Thus, Adam provides more reliable optimization in the presence of weak gradients, whereas SGD may offer better long-term performance when combined with appropriate regularization and architectural choices.

4. Methodology

This study adopts a quantitative research approach, utilizing an experimental design to compare the effects of sigmoid and ReLU activation functions on vanishing gradients in deep artificial neural networks.

To compare the effects of sigmoid and ReLU activation functions on vanishing gradients with regard to different optimization techniques using a quantitative research approach and experimental design, we will follow a structured methodology grounded in empirical data collection and statistical analysis.

Using a quantitative research approach and experimental design allows for a rigorous, repeatable, and statistically valid comparison of sigmoid vs. ReLU activation functions with respect to the SGD and Adam optimizers in the context of

vanishing gradients. This methodology has the potential to yield concrete evidence to support or refute theoretical expectations, providing valuable insights into neural network training dynamics.

4.1. Research Problem and Hypothesis

4.1.1. Research Problem

Vanishing gradients hinder the training of deep neural networks. This study aims to quantitatively compare the extent to which sigmoid and ReLU activation functions, with respect to Stochastic Gradient Descent (SGD) and Adam optimization techniques, contribute to or mitigate vanishing gradients.

4.1.2. Hypotheses

- H_0 (Null Hypothesis): There is no significant difference in the magnitude of gradients when using sigmoid vs. ReLU activation functions.
- H_1 (Alternative Hypothesis): ReLU leads to significantly fewer vanishing gradients compared to sigmoid.

4.2. Quantitative Research Approach

The quantitative research approach emphasizes numerical data, statistical inference, and objective measurement. In this study, we will apply it as follows.

4.2.1. Data Collection

- Run multiple training sessions of deep neural networks using different activation functions (sigmoid and ReLU) combined with different optimization techniques (SGD and Adam).
- Collect numerical gradient values (e.g., the mean absolute value of gradients per layer per epoch).
- Track training loss, accuracy, and convergence time.

4.2.2. Variables

- Independent Variable: Type of activation function (sigmoid or ReLU).
- Dependent Variables: Magnitude of gradients, convergence speed, training loss, and final model accuracy.
- Control Variables: Network architecture, optimizer, learning rate, batch size, and dataset used.

4.3. Experimental Design

For this experimental design, we will focus on ensuring the control and repeatability of the experiments. Therefore, we will use the following framework.

4.3.1. Network Architecture

For this experiment, we will use a deep feedforward or convolutional neural network for images. The network will have 128 input neurons, 5 layers, a Sigmoid or ReLU activation function for the hidden layers, a SoftMax activation for the output layer, and SGD (Stochastic Gradient Descent) or Adam as the optimizer. All

hyperparameters and initial conditions (except the activation function) are kept constant.

The proposed Artificial Neural Network (ANN) architecture in Figure 4 comprises 128 input neurons, five hidden layers using sigmoid activation functions, and a SoftMax-activated output layer, and is well-suited for investigating the vanishing gradient problem in the context of different activation and optimization strategies. Deep networks with multiple hidden layers are particularly susceptible to vanishing gradients, especially when using sigmoid activations. This makes the architecture ideal for empirically observing the limitations imposed by vanishing gradients on weight updates and convergence.

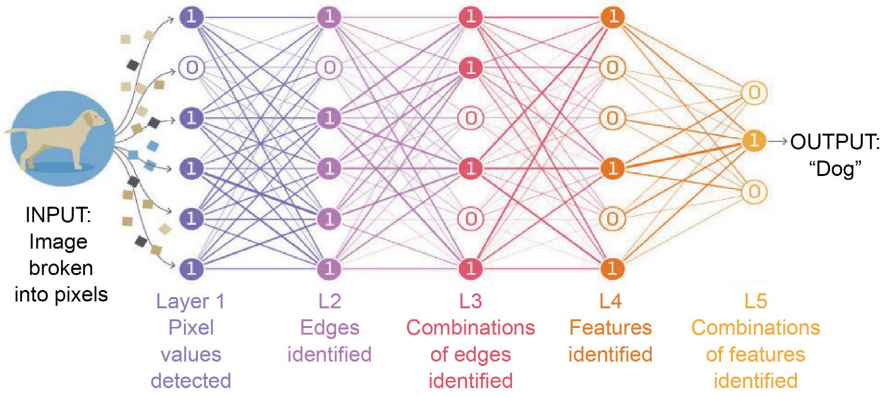

Figure 4. Neural network architecture.

The use of five hidden layers provides sufficient depth to expose these issues, as deeper architectures increase the likelihood of gradients shrinking during backpropagation [10]. Incorporating 128 input neurons allows the model to handle moderately complex input features, making it suitable for experimentation across various datasets while maintaining computational feasibility. Additionally, using Softmax activation in the output layer aligns with standard multi-class classification practices, enabling the use of cross-entropy loss, which is sensitive to small gradients and thus accentuates vanishing gradient effects [19].

Finally, running this architecture under both SGD and Adam optimizers enables a direct comparison of how different optimization techniques influence training stability and gradient propagation, making this setup both theoretically grounded and experimentally informative.

1. Implementation

For this research project, we have decided to implement the entire Artificial Neural Network (ANN) from scratch instead of using existing frameworks such as TensorFlow, PyTorch, or Keras. This decision was motivated by the need to have granular and low-level control over the learning process so that we could collect gradient data with more precision at each layer and epoch. As proposed in Figure 5, the components of this implementation are structured to facilitate the flow of data and transformations through various computational layers. The ar-

chitecture begins with the main module, which orchestrates the remaining components of the architecture.

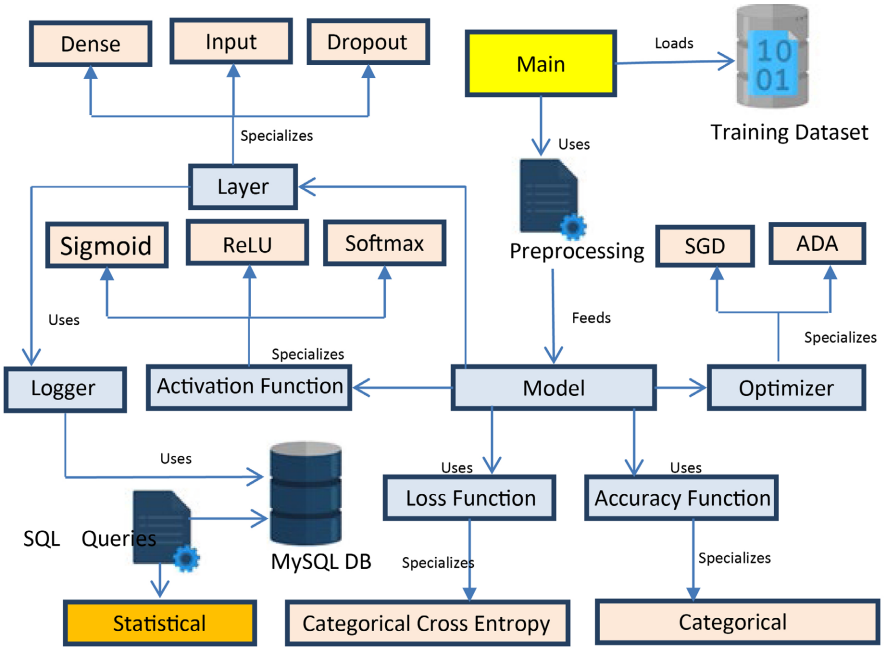

Figure 5. Neural network architecture: components diagram.

4.3.2. Between-Subjects Design

We will create four groups of neural networks.

1. Group 1 uses sigmoid activation with an SGD optimizer

Here are the characteristics as detailed in Table 1:

- 128 input neurons;
- 5 hidden layers + 1 input layer + 6 activation layers;
- Sigmoid activation functions for hidden layers;
- SoftMax activation for the output layer;
- SGD (Stochastic Gradient Descent) is the optimization algorithm.

Table 1. Group 1 neural network architecture—sigmoid and SGD.

Layer	Layer Type	Number of Neurons	Activation Function	Optimizer
Input Layer	Input	128	-	-
Hidden Layer 1	Dense (Fully Connected)	128	Sigmoid	SGD
Hidden Layer 2	Dense	128	Sigmoid	SGD
Hidden Layer 3	Dense	64	Sigmoid	SGD
Hidden Layer 4	Dense	32	Sigmoid	SGD
Hidden Layer 5	Dense	16	Sigmoid	SGD
Output Layer	Dense	10	SoftMax	SGD

2. Group 2 uses sigmoid activation with an Adam optimizer

Here are the characteristics as detailed in Table 2:

- 128 input neurons;
- 5 hidden layers + 1 input layer + 6 activation layers;
- Sigmoid activation functions for hidden layers;
- SoftMax activation for the output layer;
- Adam as the optimization algorithm.

Table 2. Group 2 neural network architecture—sigmoid and Adam.

Layer	Layer Type	Number of Neurons	Activation Function	Optimizer
Input Layer	Input	128	-	-
Hidden Layer 1	Dense (Fully Connected)	128	Sigmoid	Adam
Hidden Layer 2	Dense	128	Sigmoid	Adam
Hidden Layer 3	Dense	64	Sigmoid	Adam
Hidden Layer 4	Dense	32	Sigmoid	Adam
Hidden Layer 5	Dense	16	Sigmoid	Adam
Output Layer	Dense	10	SoftMax	Adam

3. Group 3 uses ReLU activation with an SGD optimizer

Here are the characteristics as detailed in Table 3:

- 128 input neurons;
- 5 hidden layers + 1 input layer + 6 activation layers;
- ReLU activation functions for hidden layers;
- SoftMax activation for the output layer;
- SGD (Stochastic Gradient Descent) is the optimization algorithm.

Table 3. Group 3 neural network architecture—ReLU and SGD.

Layer	Layer Type	Number of Neurons	Activation Function	Optimizer
Input Layer	Input	128	-	-
Hidden Layer 1	Dense (Fully Connected)	128	ReLU	SGD
Hidden Layer 2	Dense	128	ReLU	SGD
Hidden Layer 3	Dense	64	ReLU	SGD
Hidden Layer 4	Dense	32	ReLU	SGD
Hidden Layer 5	Dense	16	ReLU	SGD
Output Layer	Dense	10	SoftMax	SGD

4. Group 4 uses ReLU activation with the Adam optimizer

Here are the characteristics as detailed in Table 4:

- 128 input neurons;
- 5 hidden layers + 1 input layer + 6 activation layers;
- ReLU activation functions for hidden layers;
- SoftMax activation for the output layer;
- Adam as the optimization algorithm.

Table 4. Group 4 neural network architecture—ReLU and Adam.

Layer	Layer Type	Number of Neurons	Activation Function	Optimizer
Input Layer	Input	128	-	-
Hidden Layer 1	Dense (Fully Connected)	128	ReLU	Adam
Hidden Layer 2	Dense	128	ReLU	Adam
Hidden Layer 3	Dense	64	ReLU	Adam
Hidden Layer 4	Dense	32	ReLU	Adam
Hidden Layer 5	Dense	16	ReLU	Adam
Output Layer	Dense	10	SoftMax	Adam

4.3.3. Hyper-Parameters

Each group of neural networks is trained multiple times with different random seeds for statistical robustness. The above uses a progressively decreasing architecture, which is common in classification tasks. Dropout and Batch Normalization are also used between layers as regularization is needed. Table 5 provides a complete list of hyperparameters.

Table 5. Training hyperparameters.

Hyper-Parameter	Value
Initial Adam learning rate	0.001
Initial SDG learning rate	1
Initial Momentum	0
Weight initialization scheme	Random
Batch size	128
Epochs	10
Optimizer Decay	1e-3
Epsilon	1e-7
beta_1	0.9
beta_2	0.999
Initial weight_regularizer_l1	0
Initial weight_regularizer_l2	0
Initial bias_regularizer_l2	0

To give a clear idea of how this architecture was implemented in Python, Figure 6 and Figure 7 provide the code snippets of the dependencies as well as the body of the main module of the deep neural network.

```
from utilities.model import Model
from utilities.layer import Layer_Dense
from utilities.activation import Activation_ReLU
from utilities.activation import Activation_Softmax
from utilities.optimizer import Optimizer_SGD
from utilities.loss import Loss_CategoricalCrossentropy
from utilities.accuracy import Accuracy_Categorical
```

Figure 6. Python code of the neural network: dependency modules.

```
# Instantiate the model
model = Model()

# Add layers
model.add(Layer_Dense(X.shape[1], 128, "layer_1_input"))
model.add(Activation_ReLU("layer_2_relu"))
model.add(Layer_Dense(128, 128, "layer_3_hidden"))
model.add(Activation_ReLU("layer_4_relu"))
model.add(Layer_Dense(128, 64, "layer_5_hidden"))
model.add(Activation_ReLU("layer_6_relu"))
model.add(Layer_Dense(64, 32, "layer_7_hidden"))
model.add(Activation_ReLU("layer_8_relu"))
model.add(Layer_Dense(32, 16, "layer_9_hidden"))
model.add(Activation_ReLU("layer_10_relu"))
model.add(Layer_Dense(16, 10, "layer_11_hidden"))
model.add(Activation_Softmax("layer_12_softmax"))

# Set loss, optimizer and accuracy objects
model.set(
        loss=Loss_CategoricalCrossentropy(),
        optimizer=Optimizer_SGD(decay=1e-3),
        accuracy=Accuracy_Categorical()
)

# Finalize the model
model.finalize()
```

Figure 7. Python code of the neural network: main module.

4.3.4. Dataset

For this experiment, we have opted to use the MNIST dataset. MNIST is an ideal benchmark for experimentation on our research topic due to its simplicity, balanced structure, and wide adoption in deep learning research. Comprising 70,000 grayscale images (60,000 for training and 10,000 for testing), each of 28×28 pixels and labeled from 0 to 9, MNIST presents a low-dimensional yet sufficiently complex classification task that allows for effective observation of learning behavior in neural networks [20]. Its moderate input size (784 features per image) aligns well with the proposed ANN architecture featuring 128 input neurons and multiple hidden layers. Importantly, MNIST's structure makes it easier to isolate and observe the effects of architectural and optimization choices—such as the use of sigmoid vs. ReLU and SGD vs. Adam—on issues like vanishing gradients.

Even though MNIST does not require 12 layers to achieve >99% accuracy, using such a network can be justified as our goal is research and testing deep architectures. It also serves as a stepping stone for more complex datasets (e.g., Fashion-

MNIST, CIFAR-10) where deeper networks are beneficial.

Additionally, this dataset is well-understood, computationally lightweight, and does not introduce confounding domain-specific complexity, making it a practical and scientifically valid choice for foundational experimentation [19]. The widespread use of MNIST in past research also facilitates reproducibility and comparison with existing studies addressing gradient flow and neural optimization behavior.

Thus, while a 12-layer fully connected network would be overkill for MNIST, it can be justified for experimentation, studying deep network behavior, or testing regularization and optimization techniques. With proper precautions, overparameterization can even aid generalization rather than hurt it.

Metrics to Measure the Vanishing Gradient

- Mean or median of gradients for each layer during training.
- Frequency of near-zero gradients (e.g., < 1e−3 (0.001) or gradients < 1e−5).
- Gradient norm ratios between layers.

4.3.5. Data Collection

For this experiment, we will collect gradient data from each layer through different epochs. Each hidden layer of the architecture is made of:

- 128 inputs;
- 128 neurons;
- 128 sets of weights (one set for each neuron);
- 128 weights per set (one weight for each input);
- 128 gradients (each gradient belongs to a single neuron);
- 128 partial derivatives per gradient (each partial derivative belongs to a single weight);
- 7 layers (1 input, 5 hidden, and 1 output).

This gives us a total of 16,384 partial derivatives to be generated per hidden layer.

Data size: 45 GB per experiment.

Rows per layer: 3,676,960.

4.4. Statistical Analysis

For this study, we will use descriptive statistics to measure the gradient norm ratios between layers as well as the mean of gradient values per layer.

4.4.1. Gradient Norm

The gradient norm is a quantitative measure of the overall magnitude of the gradients in a neural network during training. It reflects how strongly the loss function responds to changes in the model parameters and is commonly expressed as the L_2 norm of the gradient vector, $\left\|\nabla_\theta \mathcal{L}\right\|_2$. Monitoring the gradient norm provides critical insight into training dynamics, helping to identify issues such as vanishing or exploding gradients, which can hinder convergence or destabilize learning [4] [14]. Large gradient norms indicate that parameter updates are excessively

large, potentially causing divergence, while very small norms suggest weak learning signals and slow progress. Thus, the gradient norm serves as a foundational diagnostic tool in deep learning optimization, revealing how efficiently information and learning signals propagate through the network.

1. Gradient Norm Formula Equation

A concise equation for the gradient norm of a gradient vector $\mathcal{G}$ is:

$$\|g\|_p = \left(\sum_i |g_i|^p\right)^{1/p}. \tag{1}$$

Here, g_i are the components of the gradient vector, and p specifies the norm type.

In our case, we will use the common case (L2 norm), with $p = 2$, which gives:

$$\|g\|_2 = \sqrt{\sum_i g_i^2}. \tag{2}$$

2. Pseudo Code

1. Square each element of the gradient vector.
2. Sum all the squared values.
3. Calculate the square toot of the sum.

For this study, we use the average gradient norm instead of the per-step gradient norm to have a stable, long-term view of training dynamics rather than a snapshot from a single batch or step. This also helps with visualization as the average aggregation will reduce the number of items in the dataset while maintaining the statistical significance.

3. Average Gradient Norm Formal Equation

$$AvgGradNorm_l = \frac{1}{T}\sum_{t=1}^{T}\left\|\nabla_{\theta_l}\mathcal{L}^{(t)}\right\|_p. \tag{3}$$

where:

1. l = layer index;
2. T = total number of training steps or batches;
3. θ_l = parameters of layer l;
4. $\nabla_{\theta_l}\mathcal{L}^{(t)}$ = gradient of the loss at step t with respect to layer l;
5. $\|\cdot\|_p$ = chosen norm (commonly $p = 2$).

4. Average Gradient Norm Pseudocode

6. Initialize sum norm = 0;
7. For each training step t from 1 to T:

- Compute the norm of the gradient for layer l at step t.
- Add this value to the sum norm.

8. Divide the sum norm by T to get the average gradient norm for layer l.

5. Average Gradient Norm SQL Code

Because I used a MySQL database server to collect all the gradient metrics for easy manipulation using a structured query language, below in **Figure 8** is the SQL code used to calculate the average gradient norm.

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `layer_10_norm_avg` AS
    SELECT
        `sigmoid_sgd_layer_10_sigmoid`.`Epoch` AS `Epoch`,
        AVG(
            SQRT(
                COALESCE(POW(`sigmoid_sgd_layer_10_sigmoid`.Gradient1, 2), 0) +
                COALESCE(POW(`sigmoid_sgd_layer_10_sigmoid`.Gradient2, 2), 0) + ... +
                COALESCE(POW(`sigmoid_sgd_layer_10_sigmoid`.Gradient127, 2), 0) +
                COALESCE(POW(`sigmoid_sgd_layer_10_sigmoid`.Gradient128, 2), 0))
    AS norm
    FROM `sigmoid_sgd_layer_10_sigmoid`
    GROUP BY `sigmoid_sgd_layer_10_sigmoid`.`Epoch`;
```

Figure 8. Average gradient norm SQL query for layer 10.

In our study, we are more interested in capturing the changes in gradients by layer. To achieve this, we divide the gradient norm of layer n by that of layer $n-1$. In this way, we measure how gradients change as they flow through layers. This ratio helps identify gradient flow issues such as detecting vanishing or exploding gradients. We use SQL code to compute this as shown in **Figure 9** and provide an interpretation in **Table 6**.

Therefore, the gradient norm ratio between layer and layer +1 is:

$$Ratio_{l,l+1} = \frac{\left\|\nabla W^{(l)}\right\|}{\left\|\nabla w^{(l+1)}\right\|}. \tag{4}$$

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `ratio_2_1_avg` AS
    SELECT
        `a`.`Epoch` AS `Epoch`,
        (`b`.`norm` / NULLIF(`a`.`norm`, 0)) AS `ratio_2_1`
    FROM
        (`layer_1_norm_avg` `a`
        JOIN `layer_2_norm_avg` `b` ON ((`a`.`Epoch` = `b`.`Epoch`)));
```

Figure 9. Average gradient norm ratio SQL query for layers 2 and 1.

The gradient norm ratio can be interpreted as follows.

Table 6. Training hyperparameters.

Average Gradient Norm Ratio $\nabla W^{(l)} / \nabla w^{(l-1)}$	Interpretation
>5	Possible exploding gradient
<=1e-3	Possible vanishing gradient
±1	Healthy gradient flow
It varies significantly per layer.	Indicates possible instability or poor initialization

4.4.2. The Gradient Mean

In empirical research investigating the effect of activation functions (ReLU and Sigmoid) and optimization techniques (Adam and SGD) on the vanishing gradient problem, the use of statistical summarization methods such as the mean is crucial. In a deep neural network with 12 layers, where each layer can have hundreds or thousands of gradient values during backpropagation, raw gradients are typically high-dimensional, noisy, and non-uniform. Computing the mean of gradient values per layer or per epoch provides a compact, interpretable, and scalable metric for evaluating training dynamics.

5. Experiments

5.1. ReLU Activation + Adam Optimizer

5.1.1. Gradient Norm Ratio

In Figure 10 below, we provide a holistic view of how gradient norms changed layer by layer.

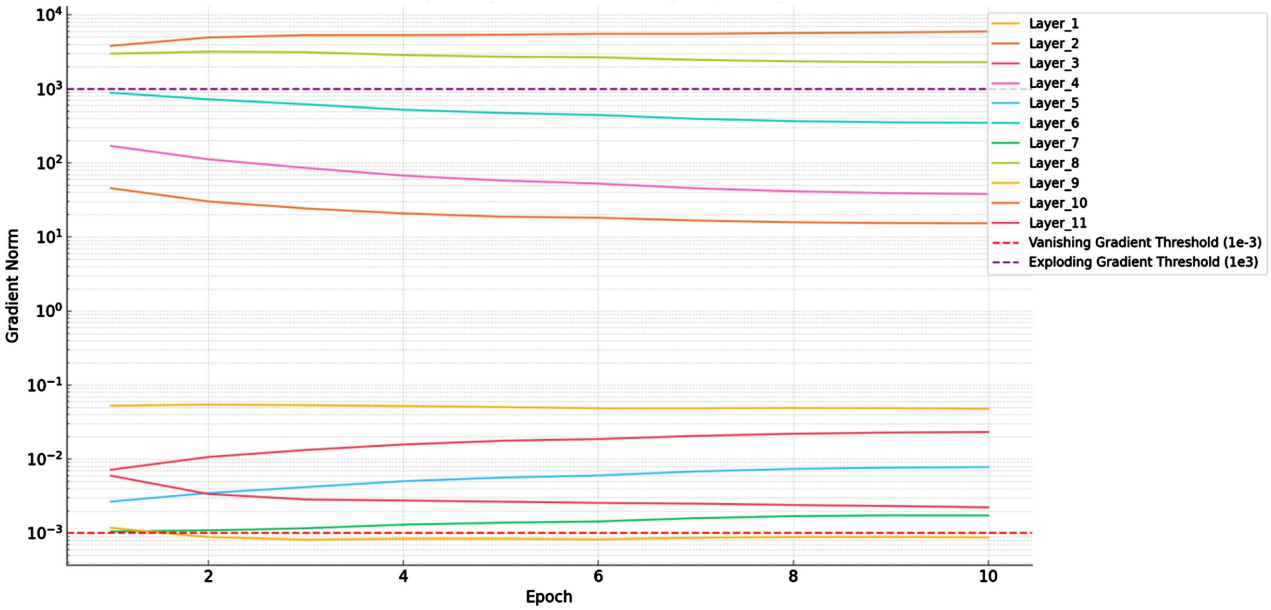

Figure 10. ReLU-Adam gradient norm ratio per Layer.

1. Threshold Annotations

- Red dashed line (1e−3): Vanishing gradient threshold.
- Purple dashed line (1e3): Exploding gradient threshold.

2. Observation

- The plot uses a logarithmic y-axis, which helps clearly display values across several magnitudes (from 1e−4 to 1e4).
- Vanishing gradients are evident in layers with norms below 1e−3 and are visible only in one trainable layer, Layer 9.
- No training layer had exploding gradients (gradient norms above 1e3). The only explosion observed was in activation layers 8 and 10.

- Table 7 provides a tabular representation of the gradient's dynamics.

Table 7. ReLU-Adam gradient norm ratio classification.

Epoch	Layer1	Layer2	Layer3	Layer4	Layer5	Layer6	Layer7	Layer8	Layer9	Layer10	Layer11
1	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Stable	Exploding	Stable
2	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
3	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
4	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
5	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
6	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
7	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
8	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
9	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable
10	Stable	Stable	Stable	Stable	Stable	Stable	Stable	Exploding	Vanishing	Exploding	Stable

5.1.2. Mean Gradient for Layer 9

Because Layer 9 is the only trainable layer affected by the vanishing gradient problem, it is important to dive deep into it and understand how the mean gradient fluctuated over the epochs. Figure 11 provides this focused view.

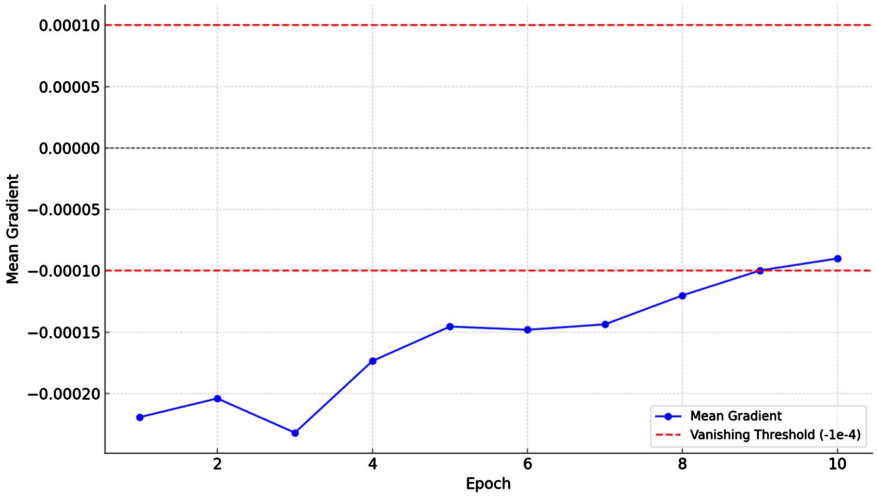

Figure 11. ReLU-Adam mean gradient per epoch in layer 9.

Observation

- The red dashed lines at ±1e−3 highlight the vanishing region.
- The values show a gradual decline, with later epochs falling into the vanishing zones, a strong indicator of learning degradation.

5.1.3. Accuracy

As depicted in Figure 12, the trend confirms effective training progress and indicates stable gradient flow and model convergence over time.

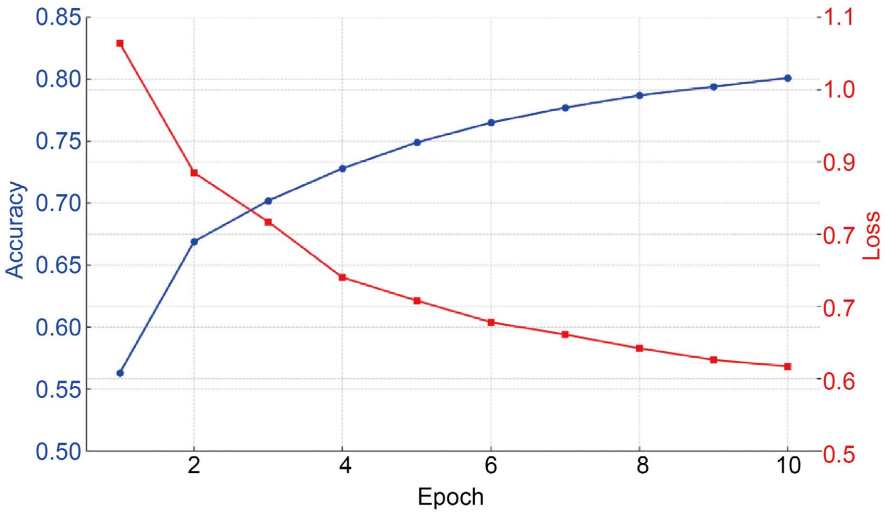

Figure 12. ReLU-Adam accuracy and loss over epochs.

Observation

- As the number of epochs increases, accuracy improves steadily from 56.3% to 80.1%.
- Simultaneously, the loss consistently decreases from 1.064 to 0.617.
- This inverse correlation is expected: as the model learns better (higher accuracy), it makes fewer prediction errors (lower loss).

5.2. ReLU Activation + SGD Optimizer

5.2.1. Gradient Norm Ratio

This architecture experiences complete vanishing gradients from epoch 2 in all layers because the gradients are consistently 0. Hence, the gradient norm ratio tends toward infinity because of the division by 0. Table 8 provides more insight.

Table 8. ReLU-SGD gradient norm ratio.

Epoch	Layer1	Layer2	Layer3	Layer4	Layer5	Layer6	Layer7	Layer8	Layer9	Layer10	Layer11
1	Infinity	Infinity	Infinity	Infinity	7.31016	0	Stable	0.000786	32368.85	0.127616	13472.95
2	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
3	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
4	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
5	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
6	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
7	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
8	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
9	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity
10	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity

1. Epoch 1 Observation

From Figure 13, we can confirm the following:

- Layer 6 experiences complete vanishing (ratio = 0).
- Layer 8 experiences vanishing (a very small ratio).
- Layers 1 - 4, 7 are possibly inactive or cut off, since all upstream layers show ∞, meaning they were divided by zero gradient norms.

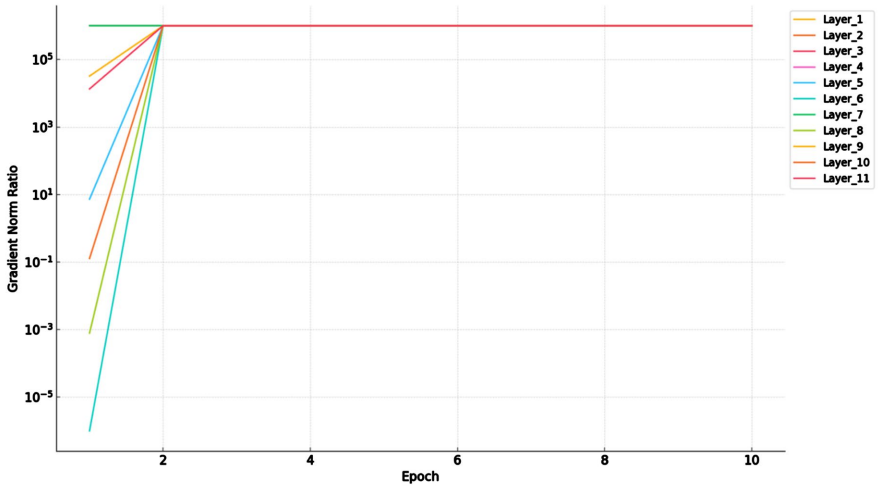

Figure 13. ReLU-SGD gradient norm ratio per layer.

2. Observation

- Log scale is used to visualize both very large and very small ratios clearly.
- Infinity values are plotted as 1e6 and 0 values as 1e-6, for visual continuity.
- Only Epoch 1 shows real computed values; all other epochs have degenerated into extremes due to vanishing, where gradients are equal to 0.
- Backpropagation is breaking down before reaching Layer 6.
- Signals from the output layer (Layer 12) are not effectively flowing back to the earlier layers.
- This is a classic vanishing gradient problem, especially affecting deeper (earlier) layers.

5.2.2. Norm Ratio SoftMax Layer 12 over Hidden Layer 11

To gain insight into where gradients are equal to zero, let us inspect the ratio norm from the SoftMax layer 12 to hidden layer 11 during backpropagation based on the information provided in Table 9.

Table 9. ReLU-SGD gradient norm ratio layers 12-11.

Epoch	Layer_11_Norm	Layer_12_Norm	Gradient_Norm_Ratio_L12_Over_L11
1	5.50477e-07	0.007416549	13472.949823516696
2	0.0	0.007416314	Infinity
3	0.0	0.007416149	Infinity

Continued

4	0.0	0.00741604	Infinity
5	0.0	0.007415966	Infinity
6	0.0	0.007415922	Infinity
7	0.0	0.007415892	Infinity
8	0.0	0.007415867	Infinity
9	0.0	0.007415825	Infinity
10	0.0	0.007415816	Infinity

Observation

- Epoch 1: Ratio ≈ 13,472.95, layer 11's gradient norm is extremely small compared to layer 12.
- Epochs 2 - 10: Ratio = ∞, because layer 11's gradient norm is zero.
- This is a classic and severe vanishing gradient problem.

5.2.3. Mean Gradient for Layer 3

To gain further insight into where gradients are equal to zero, let us inspect Figure 14 and Figure 15 showing the gradients received by layer 3 during backpropagation.

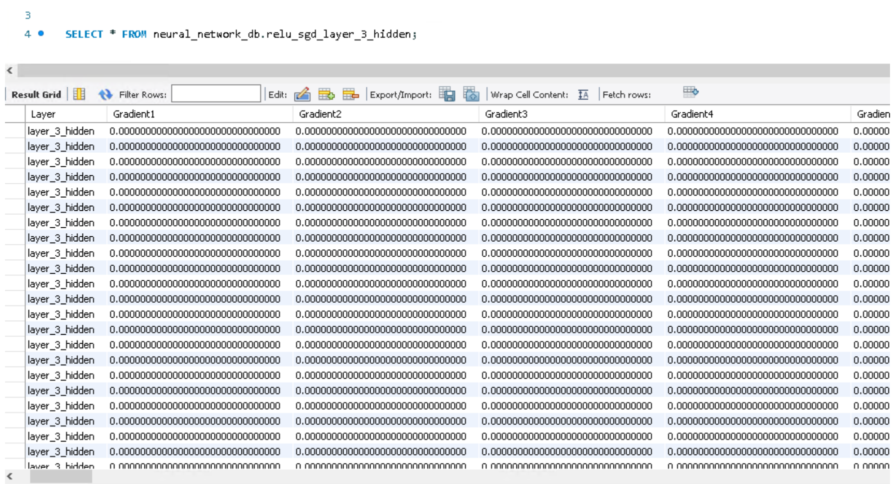

Figure 14. ReLU-SGD layer 3 MySQL gradient snapshot.

1. Observation

- The gradient is exactly zero across all epochs.
- This indicates complete vanishing. Layer 3 is not receiving any learning signal at all.

5.2.4. Accuracy

This flat trend shown in Figure 16 is a strong indicator of vanishing gradients, dead neurons, and optimizer failure in deep networks using Sigmoid activation with SGD, as confirmed by earlier statistical analysis of the experiment.

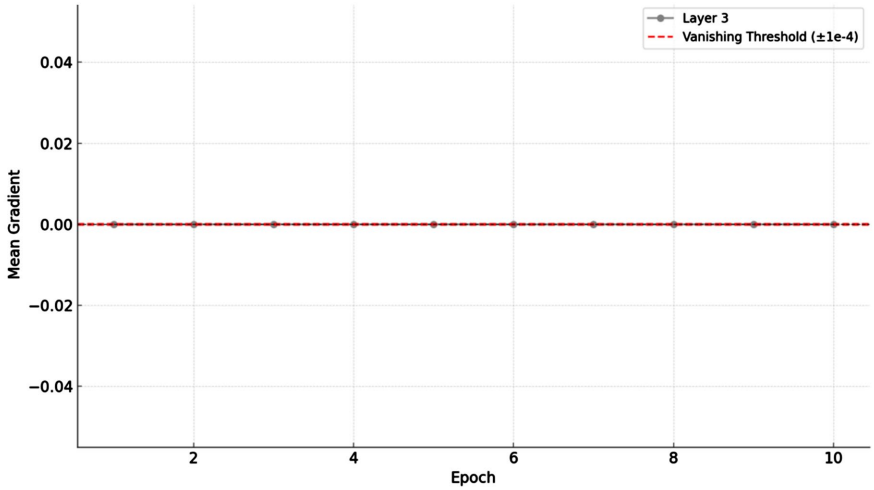

Figure 15. ReLU-SGD layer 3 mean gradient.

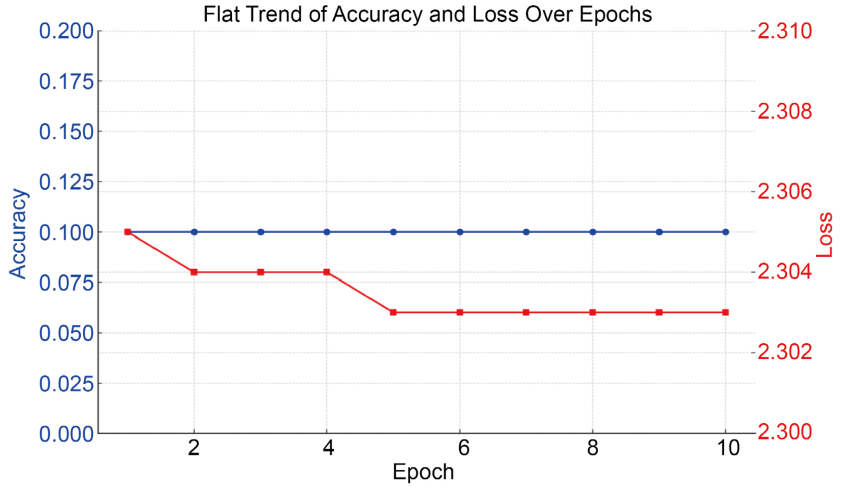

Figure 16. ReLU-SGD accuracy and loss.

Observation

- Accuracy remains constant at 10%, which corresponds to random guessing in a 10-class classification task.
- Loss stays around 2.303 - 2.305, which is the initial entropy-based loss when predictions are uninformative.
- This indicates that the model has failed to learn. No gradient flow or weight updates have led to improved performance.

5.3. Sigmoid Activation + Adam Optimizer

5.3.1. Gradient Norm Ratio

In this experiment, the earliest layers (1 - 2) have introduced ineffective learning, causing the model to fail to converge and underfit. As shown in Table 10, layers 2 and 4 are over-amplifying gradients. This causes weight explosions and loss divergence, a situation in training where the loss value increases uncontrollably instead of decreasing.

Table 10. Sigmoid-Adam average gradient norm ratio.

Epoch	Layer 1	Layer 2	Layer 3	Layer4	Layer 5	Layer6	Layer 7	Layer 8	Layer 9	Layer10	Layer 11
1	0.004914894	860.8627154	0.197353164	2,789,491,251,041.32	1	1.224744871	1	1.08012345	1	1.026436276	1
2	0.025071687	13.46757526	0.157446181	40,865,315,077.87	1	1.224744871	1	1.08012345	1	1.026436276	1
3	0.027430868	17.9886311	0.145178895	11,514,162,046.97	1	1.224744871	1	1.08012345	1	1.026436276	1
4	0.027741735	17.95290062	0.150808395	7,200,863,889.55	1	1.224744871	1	1.08012345	1	1.026436276	1
5	0.027974071	18.46727612	0.157702335	5,421,996,695.95	1	1.224744871	1	1.08012345	1	1.026436276	1
6	0.028951381	19.31739176	0.167144921	4,413,243,396.25	1	1.224744871	1	1.08012345	1	1.026436276	1
7	0.03003981	20.12576233	0.178742639	3,673,728,615.87	1	1.224744871	1	1.08012345	1	1.026436276	1
8	0.03265061	21.39081129	0.196764134	3,046,388,292.97	1	1.224744871	1	1.08012345	1	1.026436276	1
9	0.032769058	22.79576997	0.208060335	2,853,593,963.16	1	1.224744871	1	1.08012345	1	1.026436276	1
10	0.032564155	23.04775107	0.211539078	2,796,188,183.10	1	1.224744871	1	1.08012345	1	1.026436276	1

The charts in Figure 17 and Figure 18 show the Gradient Norm Ratio per Epoch for all 11 layers of the deep neural network, with a red dashed line indicating the vanishing gradient threshold at 1e−3.

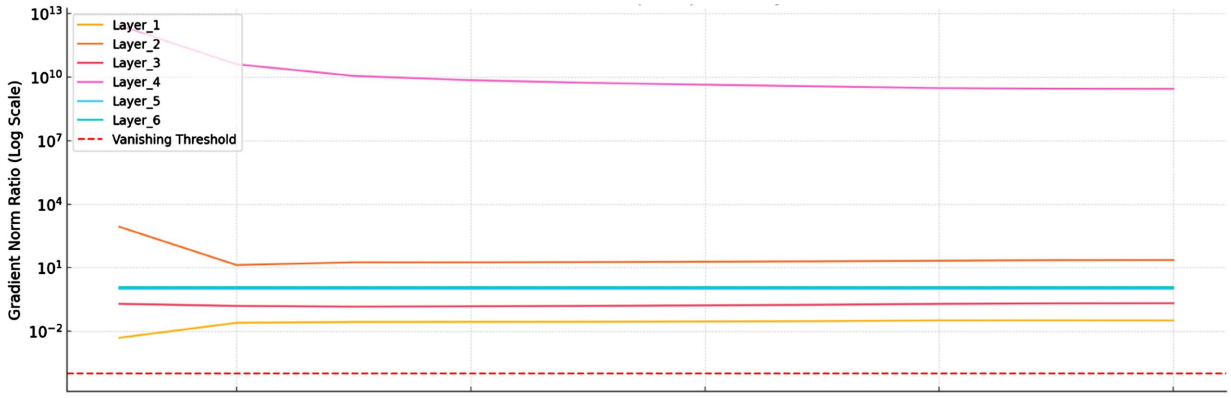

Figure 17. Sigmoid-Adam gradient norm ratio for layers 1 - 6.

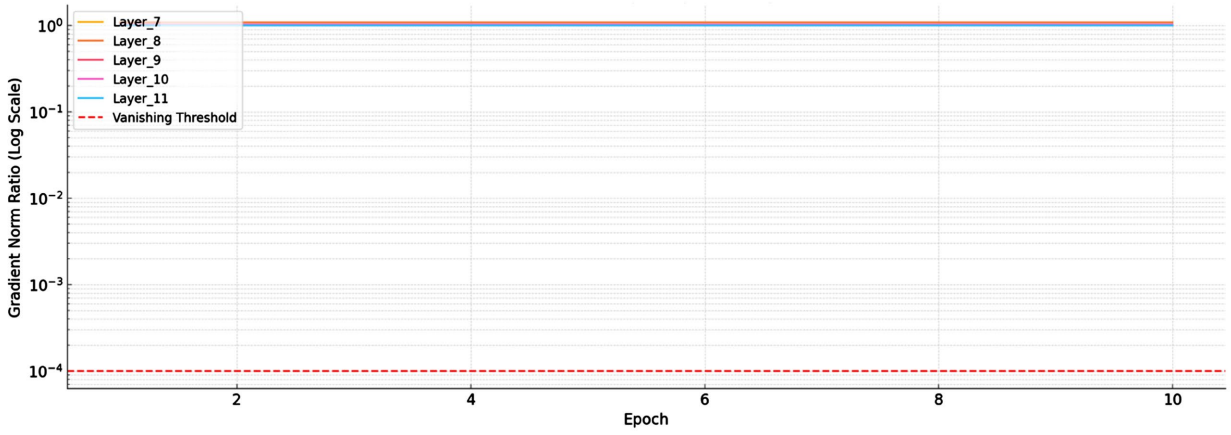

Figure 18. Sigmoid-Adam gradient norm ratio for layers 7 - 11.

Observation:

- Layer 1: Gradients start relatively well slightly above the vanishing gradient threshold (≈0.005) and remain above it. The gradient norm ratio is consistently > 1e−3.
- Layer 2: Exploding gradients start very high (~860), then drop sharply and stabilize around still high values between 13 - 23.
- Layer 4: Severe exploding starts extremely high (2.79 trillion) and decreases gradually. It is still far above the threshold but exploding.
- Layers 5 - 11: Stable gradients all around ~1.0, a sign of healthy gradient flow.

5.3.2. Accuracy

In this pattern displayed in Figure 19, the model is partially learning, likely improving confidence on already correct predictions.

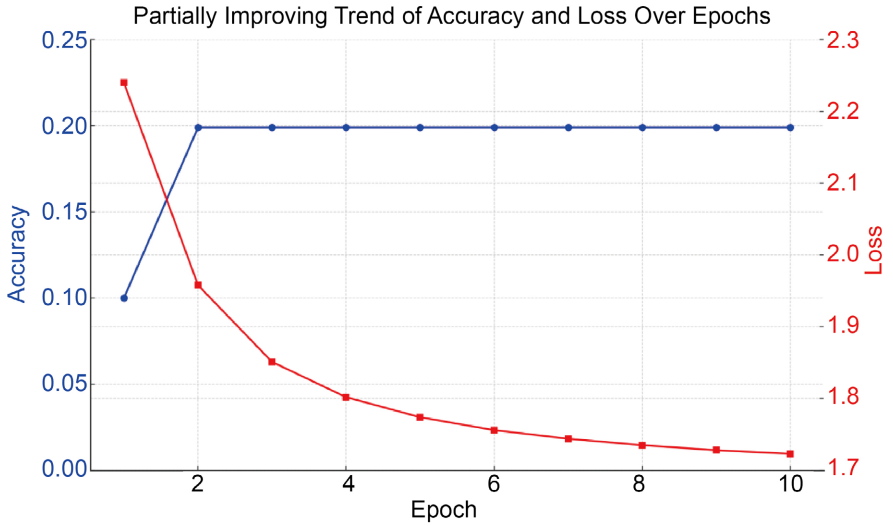

Figure 19. Sigmoid-Adam gradient norm ratio for Layers 7 - 11.

Observation:

- Accuracy improves slightly from 0.100 - 0.199 by Epoch 2 and plateaus.
- Loss continuously decreases from 2.240 to 1.723, indicating that the model is optimizing the loss function but not achieving better classification results.

5.4. Sigmoid Activation + SGD Optimizer

5.4.1. Gradient Norm Ratio

From the data presented in Table 11 and Figure 20, we understand that in this architecture the backpropagation breaks down in the early layers (Layers 1\u20134). This impairs the ability of the network to learn useful low-level features.

Observation:

Stable Layers (Above the vanishing threshold):

- Layers 5 - 11 all maintain a gradient norm ratio of approximately 1.0 or slightly

above.

- This means the gradient is consistently preserved across these layers, with no vanishing.

Table 11. Sigmoid-SGD average gradient norm ratio.

Epoch	Layer1	Layer2	Layer3	Layer4	Layer5	Layer6	Layer7	Layer8	Layer9	Layer10	Layer11
1	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
2	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
3	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
4	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
5	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
6	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
7	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
8	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
9	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1
10	infinity	infinity	infinity	infinity	1	1.224744871	1	1.08012345	1	1.026436276	1

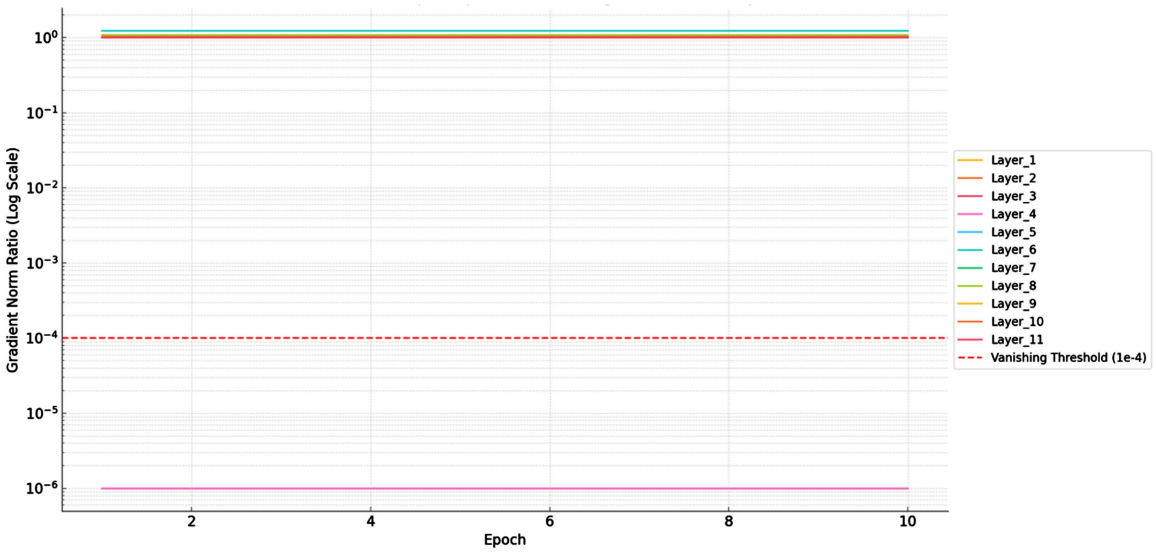

Figure 20. Sigmoid-SGD gradient norm.

Vanishing Gradient (Replaced with 1e−6 for plotting):

- Layers 1 - 4 show a ratio of infinity in your dataset, indicating:
- The denominator (lower layer) had a zero-gradient norm
- These layers are receiving no gradient signal at all.
- This is a case of severe vanishing gradient. Layers are effectively disconnected from learning.

5.4.2. Accuracy

Based on Figure 21, the pattern in this experiment is characteristic of training

failure due to vanishing gradients preventing signal propagation.

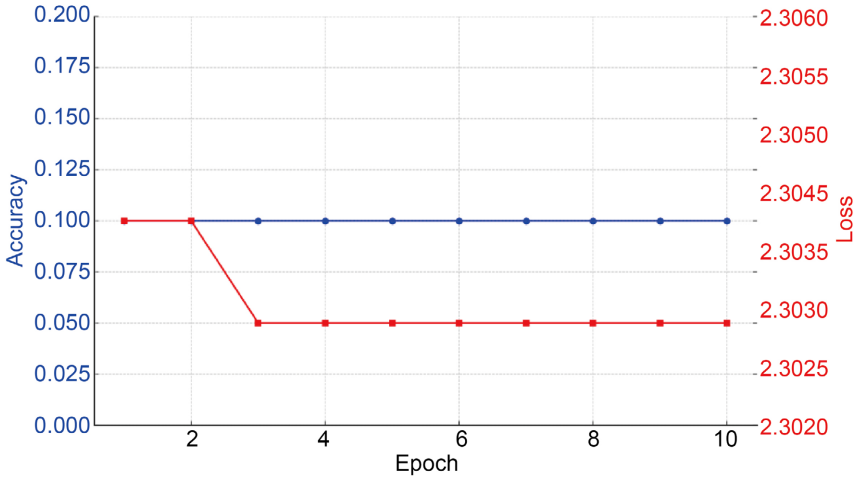

Figure 21. Sigmoid-SGD accuracy and loss.

Observation:

- Accuracy remains fixed at 10%, which is equivalent to random guessing in a 10-class classification task.
- Loss hovers around 2.303 - 2.304, matching the initial SoftMax cross-entropy loss when the model outputs uniform probabilities.

6. Findings

This research investigated how different combinations of activation functions (ReLU, Sigmoid) and optimization techniques (Adam, Stochastic Gradient Descent [SGD]) influence the vanishing gradient problem in a 12-layer deep neural network. The evaluation was based on the gradient norm ratio between layers across 10 training epochs.

The sections below provide a comparative interpretation of the empirical observations as well as delve into the underlying theoretical reasons for the observed differences. We will provide a more in-depth analysis connecting the findings to the mathematical properties of activation functions and optimizers under examination.

6.1. ReLU + Adam

This configuration showed the best overall gradient stability. Although mild vanishing was observed in hidden layer 9 (with gradient norm ratios around 0.0008), the earlier and deeper layers (Layers 1, 3, 5, 7, and 11) maintained stable ratios close to 1.0, indicating strong signal preservation during backpropagation. No signs of exploding gradients were detected in the input, hidden, and output layers, and the gradient values gradually stabilized over the epochs, causing the model to score 0.801 in accuracy. This makes ReLU + Adam the most effective combination in mitigating vanishing gradients while preserving training stability across the

network.

6.1.1. Mathematical Behavior of ReLU

The Rectified Linear Unit (ReLU) activation function is defined as:

$$f(x) = \max(0, x). \tag{5}$$

And its derivative is:

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \le 0 \end{cases}. \tag{6}$$

This implies that, unlike the sigmoid function, whose derivative is bounded between 0 and 0.25 (and decays exponentially for large $|x|$), ReLU's derivative is either 1 or 0. Thus, whenever a neuron in our architecture is active ($x > 0$), the gradient passes through unchanged and no shrinking occurs.

Mathematically, for a deep network with L layers ($L = 12$ in our case), the back-propagated gradient is approximately:

$$\frac{\partial \mathcal{L}}{\partial w_l} \approx \prod_{i=l}^{L} f'(z_i) w_i. \tag{7}$$

When $f'(z_i) \approx 1$ for most layers (as with ReLU), the gradient magnitude stays stable.

6.1.2. Adaptive Nature of the Adam Optimizer

The Adam optimizer combines momentum and adaptive learning rate scaling. It maintains first and second moment estimates of the gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t. \tag{8}$$

And

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{9}$$

Then, it performs parameter updates as:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{10}$$

This implies that during our experiment, Adam was capable of automatically rescaling gradients based on their variance and therefore preventing:

Exploding gradients from dominating updates (via $\sqrt{v_t}$).

Vanishing gradients are being neglected (via bias correction in m_t and v_t).

As a result, even though some layers of our architecture such as hidden layer 9 experienced slight decay in gradient magnitude with gradient norm ratios dropping around 0.0008, Adam compensated adaptively to maintain consistent learning across layers. This explains why in our data ReLU-Adam yielded steady gradient norm ratios across epochs with layers 1, 3, 5, 7 and 11 maintaining stable ratios close to 1.0.

Thus, ReLU + Adam performed better because ReLU ensured non-saturating gradients in any active neuron of our network and Adam adjusted learning rates dynamically per parameter. Combined, they preserved gradient magnitude and

directionality across our 12 layers.

As for the mathematical link to the Vanishing Gradient Threshold in our experiment, the vanishing threshold was defined as $\|\nabla\| < 10^{-3}$. For the ReLU-Adam combination, all layers stayed well above this threshold (10^{-2} - 10^{0} range), indicating healthy signal flow, causing ReLU-Adam to sit at the optimal balance point where gradient magnitudes neither vanish nor explode.

The mathematical robustness of ReLU and the adaptive nature of Adam jointly form a gradient-stabilizing mechanism. They maintain training signal integrity across multiple layers, preventing both vanishing and exploding gradients - two sides of the same instability spectrum. Our experiment empirically confirms this theoretical expectation: only ReLU-Adam led to a convergent and accurate model, validating foundational deep learning theory (Glorot & Bengio, 2010; Kingma & Ba, 2015).

6.2. ReLU + SGD

The ReLU + SGD experiment experienced severe gradient pathologies. In the earliest layers, all gradient norm ratios were either ∞ or 0, indicating division by zero or complete gradient disappearance, a hallmark of the vanishing gradient problem. Simultaneously, several later layers (e.g., Layer 9 and 11) had exploding gradients, with values exceeding 32,000 and 13,000, respectively. This combination proved unstable, with no effective learning occurring in either shallow or deep parts of the network, amounting to a poor performance of 0.100 in accuracy.

6.2.1. Behavior of ReLU with SGD Optimization

In our experiment, the combination of ReLU + SGD resulted in gradient explosions and divergence, with gradient norm ratios reaching infinity after the first epoch. This can be mathematically understood through the interaction between the ReLU gradient and the SGD update rule:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t). \tag{11}$$

Here, η is the learning rate and $\nabla_\theta L(\theta_t)$ is the gradient of the loss with respect to the model parameters.

Without adaptive scaling (as in Adam), SGD applied the same global learning rate η to all parameters. Because the initial gradients in our experiment were large - which often happens with ReLU when activations are not normalized - the update $\eta \nabla_\theta L$ became excessively large, causing the weights to overshoot the minimum (from layer 5 we were already at 7.31 as an average gradient norm). This led to unstable updates, where successive gradients such as in layers 9 and 11 grew exponentially, reaching respectively an average of 32,368 and 13,472, resulting in loss divergence and infinite gradient norms.

Moreover, because ReLU sets negative activations to zero, parts of our network received zero gradients more predominantly in layer 6, while other parts (especially those with large positive pre-activations) received large gradients. This imbalance exacerbated instability, leading to the exploding gradient phenomenon

observed in our experimental dataset.

6.2.2. Mathematical Indicators of Divergence

The results in Table 12 show heterogeneous gradient magnitudes across layers—some vanish while others explode—a strong signature of gradient instability. This instability prevented SGD from converging, as evidenced by the consistently infinite gradient norms in later epochs and the stagnant accuracy (0.1) observed in our model.

Table 12. ReLU-SGD mathematical divergence.

Layer	Layer1	Layer2
Layer_5	7.31	Exploding gradient zone
Layer_6	0	Dead neurons (no gradient flow)
Layer_8	0.00078	Near vanishing threshold
Layer_9	32,368	Severe gradient explosion
Layer_10	0.1276	Mild damping
Layer_11	13,472	Extreme explosion

ReLU + SGD is prone to exploding gradients due to unbounded activations and a lack of adaptive gradient control. Vanishing gradients also occur simultaneously in "dead" ReLU neurons, where $f'(x)=0$. This instability of ReLU + SGD highlights the mathematical necessity of either careful weight initialization (e.g., He initialization) or adaptive optimization algorithms for deep networks like the one we used for this experiment.

6.3. Sigmoid + Adam

While Sigmoid + Adam provided stable gradient norm ratios in deeper layers, it struggled in early layers. Layers 1 and 2 exhibited very small gradient norm ratios (~0.02 - 0.03), approaching the vanishing gradient threshold, and Layer 4 had exploding values reaching over 2.7 trillion, leading to a poor accuracy of 0.199. This shows that even with an adaptive optimizer like Adam, the Sigmoid activation's saturation in deep networks remains a major issue. Nevertheless, the stability in later layers suggests that some backpropagation was preserved.

6.3.1. Mathematical Behavior of Sigmoid

The Sigmoid function is defined as:

$$\sigma(x)=\frac{1}{1+\mathrm{e}^{-x}}. \tag{12}$$

Its derivative is:

$$\sigma'(x)=\sigma(x)\left(1-\sigma(x)\right). \tag{13}$$

This derivative reaches a maximum of 0.25 at $x=0$ and quickly approaches zero as the absolute value $|x|$ increases. In the context of our 12-layer deep neural net-

work used for the experiment, this means that during backpropagation, the chain rule multiplies many small derivatives together:

$$\frac{\partial \mathcal{L}}{\partial w_l} = \frac{\partial \mathcal{L}}{\partial a_n} \prod_{k=l}^{n} \sigma'(z_k). \tag{14}$$

In our network, because $\sigma'(z_k) < 1$, these products exponentially shrank and induced the vanishing gradient problem, causing earlier layers in the network, respectively layers 1 and 2, to effectively stop learning. This can be perceived in how these two early layers exhibited very small average gradient norm ratios (~0.02 - 0.03), thus leading to stagnation in both accuracy and loss after a few epochs, as reflected in our dataset where, for example, layer 4 had exploding gradient values reaching an average of over 2.7 trillion, leading to a poor accuracy of 0.199.

6.3.2. Role of the Adam Optimizer

The Adam optimizer (Adaptive Moment Estimation) modifies standard gradient descent by adapting the learning rate for each parameter:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t. \tag{15}$$

And

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2. \tag{16}$$

In our network, m_t and v_t track the first and second moments of the gradient, stabilizing updates by scaling the learning rate inversely with the estimated gradient variance.

This helped mitigate (but not eliminate) vanishing gradients by ensuring that even small gradients are rescaled to maintain a minimal learning signal. In our experiment, this explains why accuracy rises modestly (to around 0.199) and the loss decreases gradually from 2.24 to 1.723. This is a hallmark of slow convergence with partial vanishing rather than complete failure. The Adam optimizer preserved learning momentum despite the Sigmoid's saturation effect.

Mathematically, the Sigmoid's limited derivative range creates a narrow gradient propagation window, while Adam's adaptive scaling partially compensates for this. The result is nonlinear convergence. Progress is visible but plateaus quickly, confirming that Adam's moment-based correction improves stability but cannot fully overcome Sigmoid's intrinsic vanishing tendency.

6.4. Sigmoid + SGD

This configuration suffered from complete vanishing gradients in Layers 1 through 4. All values were marked as infinity, a common artifact of division by zero when upstream gradients disappear entirely, resulting in a poor accuracy of 0.100 as shown in **Figure 22** and **Figure 23**. However, learnable layers between 5 - 11 showed constant, unchanging gradient norm ratios (exactly 1.0), suggesting numerical stability there. Nonetheless, the failure to propagate learning signals through the earlier layers renders this configuration unsuitable for deep networks.

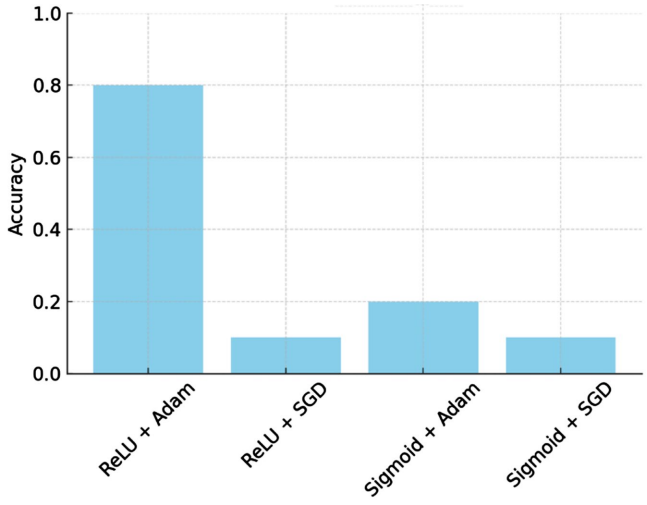

Figure 22. Model accuracy by experiment.

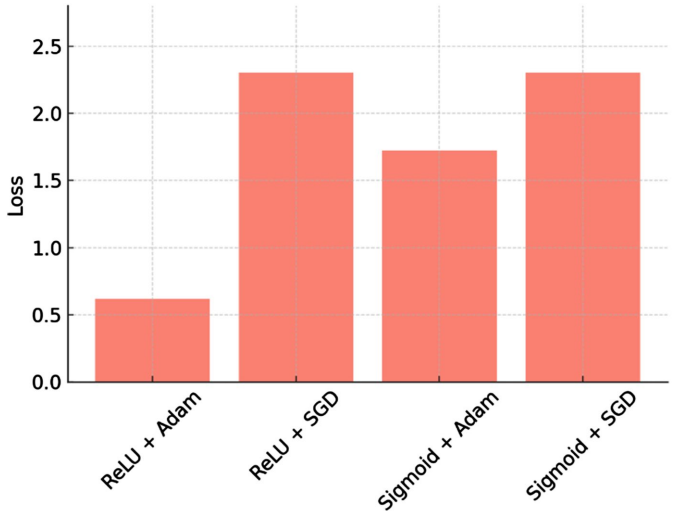

Figure 23. Model loss by experiment.

6.4.1. Sigmoid Activation Function: Mathematical Behavior and Gradient Attenuation

The derivative of a Sigmoid has a maximum value of 0.25, achieved when $\sigma(x) = 0.5$, and it rapidly approaches zero when x is far from zero (*i.e.*, when the neuron is strongly activated or deactivated). This leads to two major mathematical consequences in our 12-layer deep neural network.

1. Gradient Shrinkage Across Layers

In our 12-layer deep network, the total backpropagated gradient at layer l is approximately proportional to the product of derivatives across subsequent layers:

$$\frac{\partial \mathcal{L}}{\partial w_l} \propto \frac{\partial \mathcal{L}}{\partial a_n} \prod_{i=l}^{n} \sigma'(z_i). \tag{17}$$

Since $\sigma'(z_i) \leq 0.25$, this product rapidly tends toward zero as the depth n increases (from 1 to 12). This means the gradient magnitude at early layers can approach machine-level precision limits, causing training stagnation (vanishing gra-

dient problem).

2. Saturation Effects

During the learning process, at any epoch when the inputs to the Sigmoid function were large (positive or negative), the output saturated near 1 or 0, and $\sigma'(x)$ became extremely small. This effectively killed the gradient in those neurons, preventing meaningful weight updates.

6.4.2. Stochastic Gradient Descent (SGD): Lack of Adaptive Scaling

SGD updated the parameters in our network according to:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t). \tag{18}$$

where η is the learning rate, and $\nabla_\theta L(\theta_t)$ is the gradient of the loss.

In our case of the Sigmoid activation:

The gradients were already small due to the derivative's bounded nature.

SGD applied a fixed learning rate to these small gradients.

Without adaptive scaling (unlike Adam), the effective weight update becomes negligible, especially in earlier layers such as layers 1 through 4 where the average gradient is 0.

This means that even when the loss decreased marginally in later layers, the initial layers remained frozen, leading to poor representational learning and a plateau in accuracy, as seen in our experimental data where accuracy stagnates around 0.1 (random chance) while the loss remains high (≈2.303).

6.4.3. The Combined Effect: Severe Vanishing Gradient

As we combined Sigmoid activation with SGD, the multiplicative attenuation from Sigmoid derivatives and the non-adaptive update rule of SGD interacted to create the worst-case scenario for our deep learning optimization:

- Early layers from layer 1 to 4 received zero or near-zero gradient signals.
- Weight updates approached zero, halting learning in the lower network.
- The model became biased toward the initial weights, unable to extract hierarchical features.

This explains why our experiment showed:

- Constant accuracy (0.1) across epochs,
- No meaningful loss reduction occurred after initialization,
- Stable but ineffective gradient propagation, consistent with total vanishing.

Sigmoid's derivative structure inherently causes gradients to diminish exponentially in deep networks. SGD's fixed learning rate cannot recover lost gradient signal strength. Together, they form a mathematically unstable pairing for deep architectures (12 layers or more). This validates our experimental results as the network fails to learn beyond random performance due to vanishing gradients propagating through the Sigmoid layers.

6.5. Summary

Across all combinations, ReLU with Adam was the most effective in minimizing vanishing gradients, showing steady gradient norm ratios without numerical in-

stability. In contrast, SGD, when combined with either activation function, resulted in significant vanishing or exploding gradients. This highlights the importance of activation-optimizer synergy in deep network design and reinforces the preference for ReLU and adaptive optimizers like Adam in deep learning applications.

7. Conclusions

This study examined the impact of activation functions (ReLU and Sigmoid) and optimization techniques (Adam and Stochastic Gradient Descent [SGD]) on the vanishing gradient problem within a 12-layer deep neural network architecture. By analyzing gradient norm ratios across training epochs and layers, the research aimed to uncover which combinations best preserve gradient flow and support effective learning in deep models.

The experiments revealed substantial differences in gradient behavior depending on the chosen activation–optimizer pair. Notably, the combination of ReLU activation with Adam optimization exhibited the most favorable outcomes. It consistently maintained stable gradient norm ratios in the deeper layers (Layers 5 - 11) and minimized vanishing gradients in the early layers. This pairing proved resilient across all epochs, with no evidence of exploding gradients, highlighting its reliability in deep learning applications.

In contrast, ReLU + SGD suffered from extreme instability, showing both complete vanishing and exploding gradients. Layers 1 - 4 experienced gradient norm ratios of 0 or infinity, implying a total breakdown in signal propagation during backpropagation. Exploding gradients in the deeper layers further destabilized learning, making this combination unsuitable for deep networks.

The use of the Sigmoid activation function, regardless of optimizer, consistently led to vanishing gradients in early layers. Even when paired with Adam, which offered some adaptive gradient control, Sigmoid resulted in highly unstable gradients, especially in Layer 4, which reached explosive values in the trillions. While gradient stability was maintained in deeper layers, the inability of Sigmoid to effectively propagate gradients through shallow layers remains a critical limitation.

Overall, the findings confirm that both activation choice and optimizer selection significantly affect the gradient flow in deep networks. The ReLU + Adam combination offers the best protection against vanishing gradients and ensures robust training dynamics. Conversely, SGD and Sigmoid, especially when used together, amplify the risk of vanishing or exploding gradients, impeding learning in deep architectures.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Pedamonti, D. (2018) Comparison of Non-Linear Activation Functions for Deep

Neural Networks on MNIST Classification Task. arXiv.
https://arxiv.org/abs/1804.02763

[2] Glorot, X. and Bengio, Y. (2010) Understanding the Difficulty of Training Deep Feed Forward Neural Networks. *Proceedings of the* 13*th International Conference on Artificial Intelligence and Statistics*, **9**, 249-256.
http://proceedings.mlr.press/v9/glorot10a.html

[3] Ramachandran, P., Zoph, B. and Le, Q.V. (2017) Swish: A Self-Gated Activation Function for Deep Neural Networks. arXiv:1710.05941.
https://arxiv.org/abs/1710.05941

[4] Xu, B., Wang, N., Chen, T. and Li, M. (2015) Empirical Evaluation of Rectified Activations in Convolutional Network. *arXiv:1505.00853.*
https://arxiv.org/abs/1505.00853

[5] Bengio, Y., Simard, P. and Frasconi, P. (1994) Learning Long-Term Dependencies with Gradient Descent Is Difficult. *IEEE Transactions on Neural Networks*, **5**, 157-166. https://doi.org/10.1109/72.279181

[6] Glorot, X., Bordes, A. and Bengio, Y. (2011) Deep Sparse Rectifier Neural Networks. In: Gordon, G., Dunson, D. and Dudík, M., Eds., *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (*AISTATS*), Fort Lauderdale, 11-13 April 2011, 315-323.
https://proceedings.mlr.press/v15/glorot11a.html

[7] Kingma, D.P. and Ba, J. (2015) Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (*ICLR*). arXiv:1412.6980.
https://arxiv.org/abs/1412.6980

[8] Mostafanejad, M. (2023) Unification of Popular Artificial Neural Network Activation Functions. arXiv:2302.11007.

[9] Pennington, J., Schoenholz, S.S. and Ganguli, S. (2017) Resurrecting the Sigmoid in Deep Learning Through Dynamical Isometry: Theory and Practice. *Advances in Neural Information Processing Systems*, 30, 4785-4795.
https://arxiv.org/abs/1711.04735

[10] Szandała, L. (2021) Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. *Bio-Algorithms and Med-Systems*, **17**, 1-12.

[11] He, K., Zhang, X., Ren, S. and Sun, J. (2015) Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proceedings of the IEEE International Conference on Computer Vision* (*ICCV*), Santiago, 7-13 December 2015, 1026-1034. https://doi.org/10.1109/ICCV.2015.123

[12] He, K., Zhang, X., Ren, S. and Sun, J. (2016) Deep Residual Learning for Image Recognition. 2016 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, 27-30 June 2016, 770-778. https://doi.org/10.1109/cvpr.2016.90

[13] Ioffe, S. and Szegedy, C. (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the* 32*nd International Conference on Machine Learning* (*ICML*), Lille, 7-9 July 2015, 448-456.
https://arxiv.org/abs/1502.03167

[14] Maas, A.L., Hannun, A.Y. and Ng, A.Y. (2013) Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Proceedings of the* 30*th International Conference on Machine Learning* (*ICML*), Atlanta, 16-21 June 2013, 3-6.
https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

[15] Pascanu, R., Mikolov, T. and Bengio, Y. (2013) On the Difficulty of Training Recurrent Neural Networks. In: Pereira, F., Burges, C.J.C., Bottou, L. and Weinberger, K.Q.,

Eds., *Proceedings of the 30th International Conference on Machine Learning* (*ICML* 2013), Atlanta, 16-21 June 2013, 1310-1318.
https://proceedings.mlr.press/v28/pascanu13.html

[16] Hochreiter, S. (1991) Untersuchungen Zu Dynamischen Neuronalen Netzen. Doctoral Dissertation, Technische Universität München.

[17] Wilson, A.C., Roelofs, R., Stern, M., Srebro, N. and Recht, B. (2017) The Marginal Value of Adaptive Gradient Methods in Machine Learning. *Advances in Neural Information Processing Systems* (*Neurips*), **30**, 4151-4161.
https://arxiv.org/abs/1705.08292

[18] Ruder, S. (2016) An Overview of Gradient Descent Optimization Algorithms. arXiv Preprint. https://arxiv.org/abs/1609.04747

[19] Goodfellow, I., Bengio, Y. and Courville, A. (2016) Deep Learning. MIT Press.
https://www.deeplearningbook.org/

[20] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, **86**, 2278-2324.
https://doi.org/10.1109/5.726791