Neural Network (Project Code: N)

Tajkia Nuri Ananna (001661229), Nowshin Tasnim (001649184),
Ali Salehi Darjani (001657808), Joy Saha (001657938)

May 5, 2025

1 Introduction

Neural networks are foundational models in machine learning, capable of learning complex mappings from data through layered architectures. This report presents the implementation and evaluation of a feedforward neural network from scratch, beginning with a two-layer architecture. The model is tested on common classification datasets to examine performance. The report further investigates how performance is influenced by architectural choices such as the number of layers, neurons per layer, and activation functions.

2 Motivation

Machine learning models often face trade-offs between complexity, generalization, and computational efficiency. Neural networks offer flexibility but require careful design to avoid underfitting or overfitting. Understanding the influence of model architecture and components like activation functions is critical for effective neural network deployment in practical classification tasks. This project aims to build intuition about these factors by constructing and analyzing a basic neural network from scratch. Also, we focus on a deeper understanding of how neural networks make decisions—how data is transformed layer by layer and how weights and activations interact to form final predictions. By experimenting with different configurations, we aim to develop insights into the internal mechanisms of neural networks, including how they learn representations and generalize from training data to unseen examples.

3 Problem Statement

A neural network (Fig. 1) is fundamentally structured around three core types of layers: the **input layer**, **hidden layers**, and the **output layer**. The input layer serves as the entry point for raw data, receiving the features of a dataset, and passing them into the network for further processing. Following this, the hidden layers, often consisting of multiple interconnected layers, perform the essential task of learning complex patterns and representations from the data through a series of transformations.

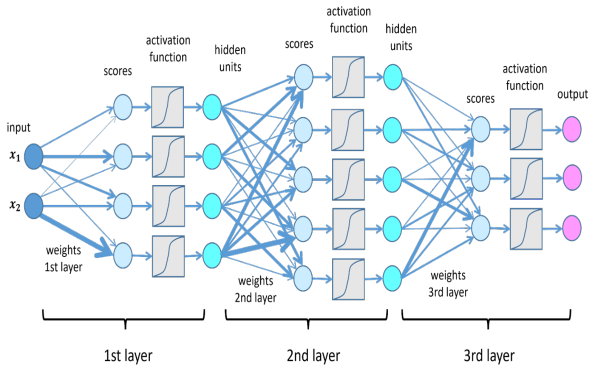
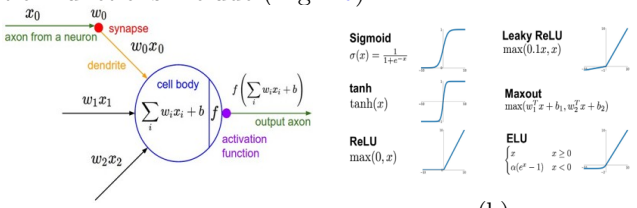


Figure 1: Neural Network.

Each of these layers is composed of units called **neurons**, which form the basic computational elements of the network. A neuron receives one or more inputs, computes a weighted sum (adding a bias term), and then applies an **activation function** to this sum (Fig. 2a). The activation function introduces non-linearity into the network, allowing it to model complex functions.Common activation functions include (Fig. 2b):



(a) (b)

Figure 2: (a) Signal flow inputs to outputs for a single neuron. (b) Different kinds of activation functions used in Neural Networks.

ReLU (Rectified Linear Unit): Outputs the input directly if it is positive; otherwise, it outputs zero.
Sigmoid: Maps the input to a range between 0 and 1, useful for binary classification.
Leaky ReLU: A variation of ReLU that allows a small, non-zero gradient when the input is negative, typically defined as $f(x) = \max(\alpha x, x)$ where α is a small constant. This helps mitigate the "dying ReLU" problem, where neurons can become inactive and stop learning.

Finally, the output layer generates the network's final prediction, converting internal representations into interpretable results such as class labels or continuous values, depending on the task's nature.
In our project, we constructed a neural network from scratch, allowing us to explore and understand each component in depth. We experimented with different activation functions and datasets to investigate the impact on learning dynamics and overall performance. Our primary objective was to build a functioning model to deeply understand how neural networks learn from data, adjust internal parameters, and optimize decision-making processes.

4 Challenges

Building neural networks from scratch involves addressing several non-trivial challenges that significantly im-

pact performance and training stability:

- **Architecture Design:** Choosing the right number of layers, neurons, and activation functions is non-trivial and problem-dependent. A poorly chosen architecture can either underfit the training or make the model unnecessarily complex and prone to overfitting.
- **Training Settings:** Effective learning depends heavily on hyperparameters such as the learning rate, batch size, and the number of epochs. Poor choices can lead to slow learning, oscillations, or failure to converge.
- **Vanishing Gradients:** In deep networks, especially those using sigmoid activation functions, gradients can become extremely small or large during backpropagation. This can lead to slow convergence or numerical instability.
- **Overfitting:** As the network's capacity increases, with more layers, neurons, or epochs, it may begin to memorize the training data rather than generalize to unseen data.
- **Computational Cost:** Deep and wide networks come with significant memory and time requirements, especially when training on large datasets.

5 Methods

This section outlines the methodology used to design, implement, and evaluate the neural network models developed for this project.

5.1 Implementation

The neural network was implemented in **Python**, utilizing libraries such as `NumPy`, `pandas`, `scikit-learn`, and `matplotlib` for numerical computation, data handling, model evaluation, and visualization. The architecture supports:

- **Support for multiple layers with configurable depth:** The architecture supports networks with two (128,10), three (128,64,10), and four (128,64,32,10) layers.
- **Customizable number of neurons in each hidden layer:** The architecture enables precise configuration of neuron counts per layer, allowing users to define the number of neurons in each hidden and output layer through a simple list input.
- Flexible choice of activation functions (e.g., ReLU, Sigmoid, Leaky ReLU).
- **Optimization—Gradient Descent and the Adam Optimizer:** The project implements two optimization methods within a custom Optimizer class. Batch Gradient Descent updates weights using basic gradient steps, making it simple but sometimes slow to converge. The Adam optimizer, implemented from scratch, improves training by adaptively adjusting learning rates using momentum-based estimates of past gradients. Users can switch between optimizers by passing the desired method to the training function.
- Support for multi-class classification using softmax activation and categorical cross-entropy loss.

Training process: We used a learning rate of 0.001 for the Adam optimizer and 0.01 for batch gradient descent. The model was trained with a batch size of 64 for a total of 100 epochs, except for the Iris dataset, which was trained for 300 epochs. Weights were initialized using a normal distribution. During training, we saved the model with the best training accuracy, which was later used to generate the final output.

5.2 Datasets

We have used six publicly available datasets. The first three F-MNIST, MNIST, and Iris—are standard classification datasets. While F-MNIST and MNIST are image datasets flattened into tabular format, Iris is a classic low-dimensional dataset often used for evaluating basic pattern recognition tasks. Among the remaining datasets, Madelon is artificially generated, while MLL and Leukemia are high-dimensional datasets, with Leukemia having the highest number of features. This diverse selection allows us to evaluate the performance of our network across various types of data. The dataset characteristics are provided in Table 1.

Dataset	#Instances	#Features	#Class
F-MNIST	70000	784	10
MNIST	70000	784	10
IRIS	150	4	3
MLL	72	5848	3
Madelon	2600	500	2
Leukemia	72	7129	2

Table 1: Dataset Used in the Experiments

5.3 Evaluation Metrics

For evaluation, we monitored three key metrics during training: cross-entropy loss, training accuracy, and validation accuracy.

6 Experiment and Results

Among all the datasets, MNIST, F-MNIST, and Iris demonstrated notable accuracy; therefore, this section focuses exclusively on these three.

6.1 Three-layer Neural Network

Three-layer neural networks with hidden units configured as $(128, 64, 10)$ for three different activations were trained on each dataset. The results are summarized in Table 2 for the MNIST, F-MNIST, and Iris datasets.

Dataset	Train Acc (%)	Val Acc (%)
MNIST	99.69	96.81
F-MNIST	92.81	92.81
Iris	100	98.33

Table 2: Performance of baseline three-layer network

6.2 Effect of Number of Layers

To evaluate how network depth influences performance, we trained and compared neural networks with 2-layer, 3-layer, and 4-layer architectures across three datasets: MNIST, F-MNIST, and Iris. Figure 3 provides a visual representation of the results.

The results on MNIST (Fig: 3a) show a clear trend: increasing the number of layers improves classification accuracy. The **2-layer** model provided a solid baseline but struggled to capture more nuanced variations. The **3-layer** network improved generalization and reduced validation loss, with better stability across training epochs. The **4-layer** network achieved the highest accuracy among all, particularly when using LeakyReLU and the Adam optimizer, but showed signs of slower training convergence due to increased complexity.

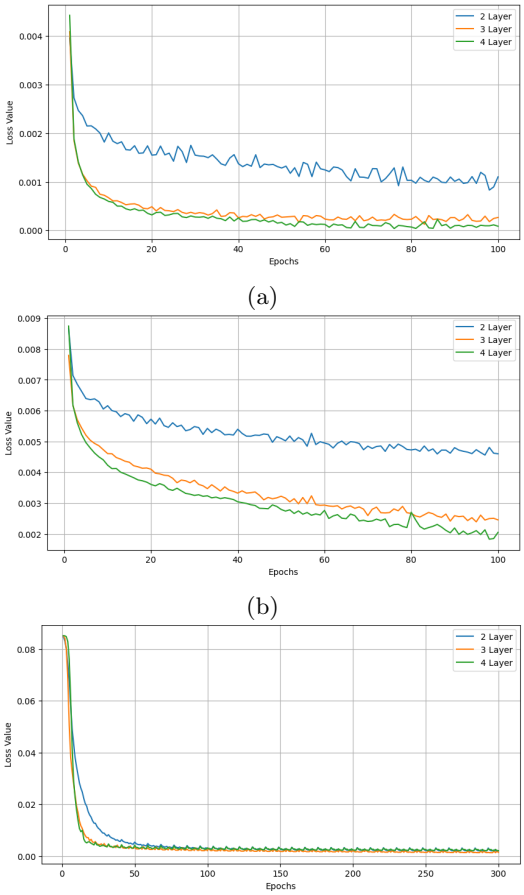


Figure 3: Loss Curves for Different Number of Layers (a) MNIST, (b) F-MNIST, (c) Iris

Similar performance trends were observed for F-MNIST (Fig. 3b): The **2-layer** model underperformed, especially on visually similar classes. The **3-layer** model struck a good balance between complexity and accuracy. The **4-layer** model showed marginal gains over the 3-layer setup, but with longer training time and increased sensitivity to initialization and learning rate.

Due to its simplicity, the Iris dataset did not benefit significantly from deeper networks (Fig. 3c). All three models reached high accuracy quickly. The **2-layer** model was sufficient for this dataset. Additionally, the **3-layer** and **4-layer** models did not yield meaningful improvements and occasionally led to slight overfitting.

Summary Deeper models (especially 3-layer and 4-layer architectures) perform better on complex datasets like MNIST and F-MNIST but offer little to no advantage for simpler data like Iris. Model depth should thus be chosen based on dataset complexity and size.

6.3 Effect of Optimizers

To understand how optimization strategies affect model performance, we compared two optimizers—Batch Gradient Descent (BGD) and Adam—across identical neural network architectures for MNIST, F-MNIST, and Iris datasets. These experiments highlight the strengths and limitations of both approaches in terms of convergence behavior, stability, and accuracy.

MNIST: Adam significantly outperformed BGD in convergence speed and training stability, reaching high accuracy within fewer epochs, and the loss curve was smooth and consistent, whereas BGD, while stable, exhibited slower convergence and required more epochs to achieve performance comparable to Adam. It was more sensitive to the learning rate and prone to stagnation without careful tuning.

F-MNIST: The differences were even more pronounced here. Adam handled the complex, high-variance input better and showed faster convergence with higher final accuracy, while BGD plateaued early in some runs and was more prone to local minima. Performance gains were modest, even with increased training time.

Iris: Both optimizers achieved near-perfect accuracy, but Adam still converged faster and required fewer updates. BGD exhibited minor instability in early epochs, especially with small neuron configurations.

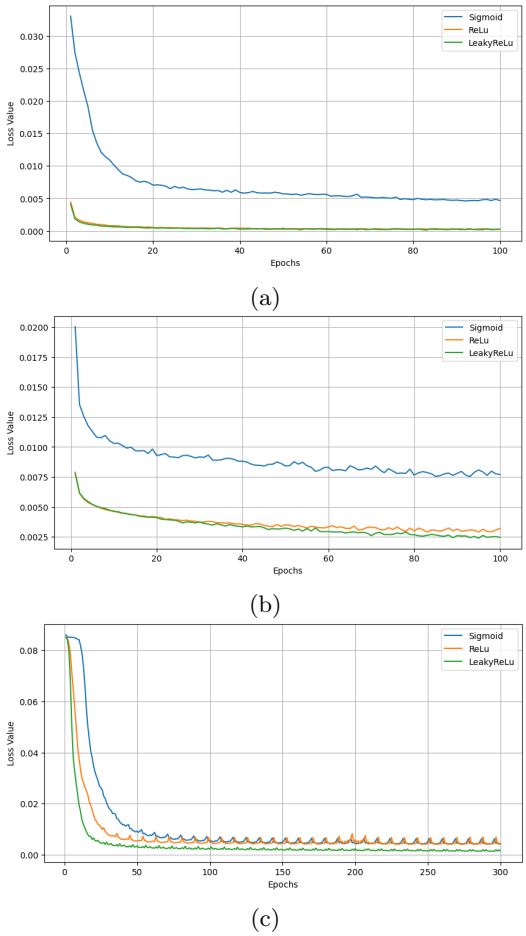


Figure 4: Performance comparison across activation functions on (a) MNIST, (b) F-MNIST, and (c) Iris datasets.

Summary: Adam consistently outperformed BGD in terms of convergence speed, training efficiency, and accuracy, particularly on more complex datasets like MNIST and F-MNIST. BGD remained effective on simpler tasks but demanded more careful learning rate tuning and longer training durations. These results suggest that

adaptive optimizers like Adam offer significant advantages when working with diverse datasets and deeper architectures.

6.4 Effect of Activation Functions

We evaluated the impact of three activation functions—**Sigmoid**, **ReLU**, and **Leaky ReLU**—using a consistent three-layer architecture ($[128, 64, \text{output}]$) and the Adam optimizer across MNIST, F-MNIST, and Iris datasets. The training loss curves and the performance for different activation functions on the MNIST, Fashion MNIST, and Iris datasets are shown in Fig. 4 and Table 3. These plots illustrate how ReLU and Leaky ReLU lead to significantly faster convergence compared to Sigmoid, particularly on deeper networks. Among the

Table 3: Activation Function Performance Across Datasets

Data	Act.	Train	Val.	Remarks
MNIST	Sigmoid	91.3	90.8	Slow; vanishing grad.
	ReLU	99.7	96.8	Fast; accurate
	LeakyR	99.8	97.8	Best; no dead neurons
F-MNIST	Sigmoid	83.1	83.1	Weak features
	ReLU	92.8	92.8	Good generalization
	LeakyR	93.5	93.5	Slightly better
IRIS	Sigmoid	98.3	100	Needs more epochs
	ReLU	97.5	100	Stable, efficient
	LeakyR	100	100	Perfect; overfit risk

activation functions evaluated, Leaky ReLU consistently achieved the highest or near-highest accuracy across all datasets, highlighting its robustness and ability to prevent issues such as dead neurons. ReLU also performed strongly, offering efficient training and solid accuracy, particularly on more complex datasets like MNIST and F-MNIST. In contrast, Sigmoid lagged behind in deeper models due to gradient saturation, though it remained effective on simpler tasks such as the Iris dataset. In addition, Fig. 5 presents a comparison of training and validation accuracy across all six datasets.

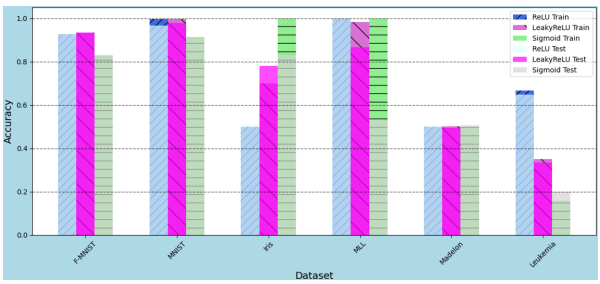


Figure 5: Training vs Validation Accuracy for Different Activation Functions

6.5 Final Result

Figure 6 illustrates the best results achieved using our neural network model. As shown, the model performs exceptionally well across a range of datasets, including MNIST, Iris, and MLL, achieving over 98% validation accuracy. This indicates that our model can effectively handle both high-instance datasets and those with varying numbers of features. Notably, the F-MNIST dataset, which has 10 classes, yields a validation accuracy of 94%, demonstrating the model's robust performance even with more complex classification tasks. However,

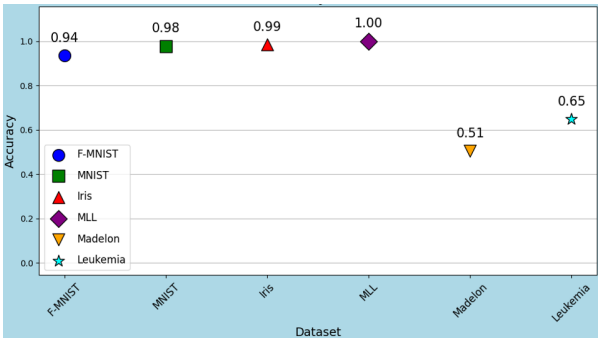


Figure 6: Training vs Validation Accuracy for Different Activation Functions

the model's performance significantly decreases on the leukemia dataset, with its accuracy dropping to 65%. This suggests that the model struggles when the number of instances is lower but the feature set is more extensive, reflecting the challenge of working with datasets that have a high dimensionality but fewer samples. Additionally, the accuracy further declines to 51% when tested on the Madelon dataset, an artificially generated dataset. This indicates that the model's performance can be heavily influenced by the nature of the dataset, with artificially generated data presenting additional complexities that are harder to navigate. These findings emphasize the importance of dataset characteristics, such as instance count, feature dimensions, and data generation methods, on the neural network's effectiveness.

7 Conclusion

The neural network exhibits strong performance on structured datasets such as MNIST, but its accuracy declines on complex, high-dimensional datasets like Leukemia with limited samples. Simpler tasks benefit from shallow architectures, whereas deeper networks yield incremental improvements on more complex data. The Adam optimizer consistently surpasses batch gradient descent in convergence efficiency, particularly on simpler datasets. Increasing the number of hidden neurons enhances model capacity and accuracy. These results highlight the critical role of aligning network architecture, activation functions, optimizers, and normalization strategies with dataset characteristics to ensure optimal model performance.