Comparative Analysis of Gradient-Based Optimization Algorithms on Non-Convex Functions

1 Introduction

This report investigates the performance of several gradient-based optimization algorithms on challenging non-convex objective functions. Two benchmark problems are considered: the Rosenbrock function, characterized by a narrow curved valley with steep gradients, and the oscillatory function $\sin(1/x)$, whose frequency increases rapidly near the origin.

The comparison is based on three quantitative metrics:

- Final function value
- Number of iterations required for convergence
- Computational time required for optimization

Including execution time enables a more realistic and comprehensive evaluation of optimizer efficiency.

2 Mathematical Background

2.1 Rosenbrock Function

The Rosenbrock function is defined as:

[f(x,y) = (1 - x)^2 + 100$(y - x^2)^2$]

The global minimum occurs at $(1, 1)$ with function value 0. Its curved valley structure makes convergence slow and unstable for many first-order optimization methods.

2.2 Sinusoidal Function

The one-dimensional sinusoidal function is given by:

[f(x) = $\sin(1/x)$]

As $x \to 0$, the oscillation frequency increases dramatically, producing infinitely many local extrema and unstable gradient directions. This makes the function useful for evaluating optimizer robustness.

3 Experimental Setup

Five optimization algorithms were implemented from scratch:

- Gradient Descent (GD)
- Momentum
- Adagrad
- RMSprop
- Adam

Each optimizer was evaluated using learning rates:
$[\ \eta \in 0.01, 0.05, 0.1]$
For every run, the following quantities were recorded:

- Estimated optimal parameters
- Final loss value
- Iteration count
- Execution time (seconds)

4 Results and Analysis

4.1 Rosenbrock Function

4.1.1 Convergence Quality

The numerical results show:

- Adam achieves extremely small loss values (approximately 10^{-9}) and converges within about 2000–2800 iterations.
- Gradient Descent and Momentum either diverge or remain at very large loss values (10^6–10^9) even after the maximum iteration limit.
- RMSprop and Adagrad provide partial improvement but fail to reach the true global minimum.

These observations indicate that Adam is the only optimizer that reliably converges on the Rosenbrock landscape under the tested settings.

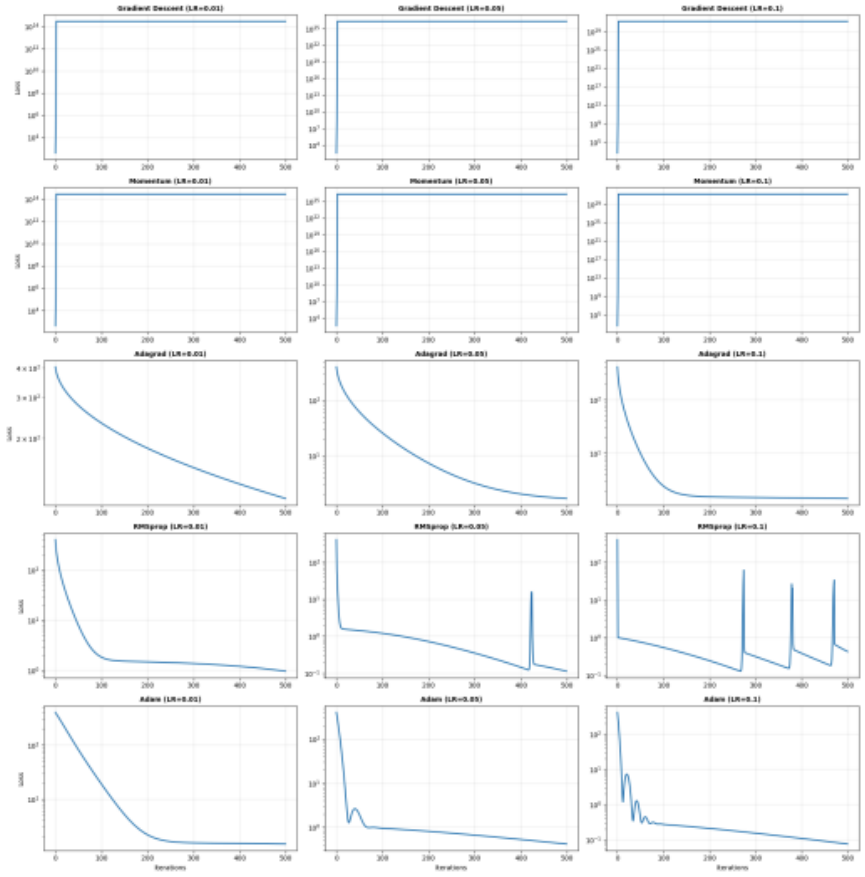


Figure 1: Rosenbrock Function

4.1.2 Time Efficiency

Execution-time measurements reveal:

- Adam converges faster in both iteration count and wall-clock time (approximately 0.20–0.34 seconds).
- GD, Momentum, RMSprop, and Adagrad typically run the full 5000 iterations, resulting in higher runtimes (0.33–0.90 seconds) despite poorer convergence.

This demonstrates an important principle: rapid convergence reduces total computation time more effectively than inexpensive individual iterations.

4.2 Sinusoidal Function $\sin(1/x)$

4.2.1 Convergence Behavior

Observed results include:

- Many optimizers reach function values close to -1 within a small number of iterations.

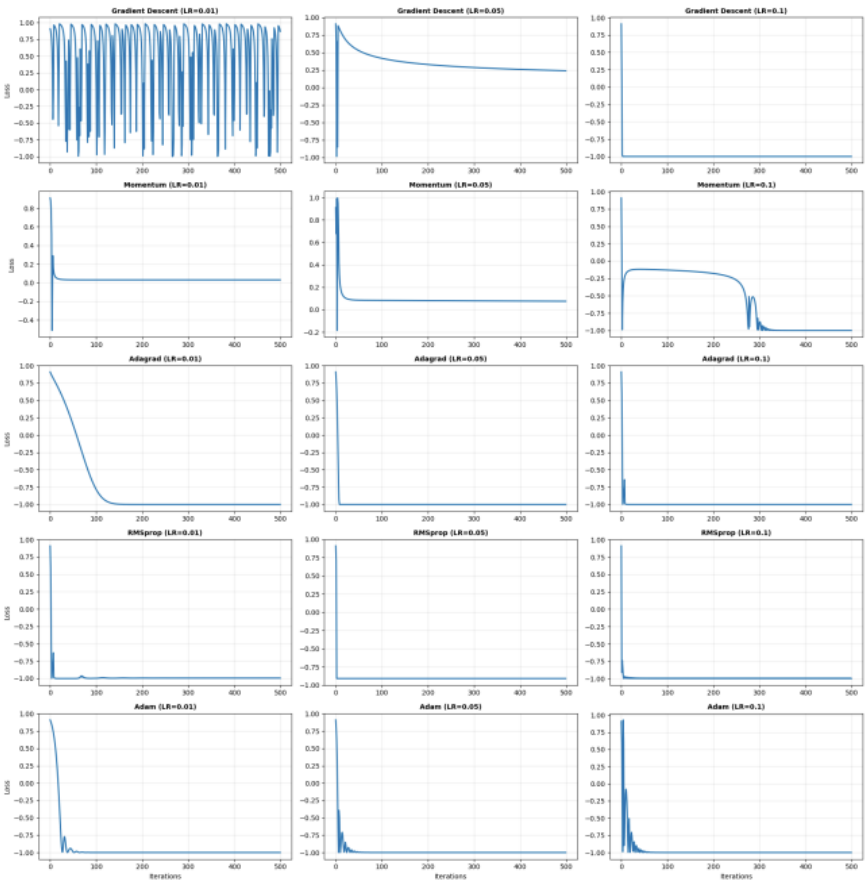


Figure 2: Sinusoidal Function (sin(1/x))

- Adaptive methods (Adagrad, RMSprop, Adam) converge in approximately 13–176 iterations.
- Gradient Descent frequently fails and instead runs for the full 5000 iterations.

However, because the function contains infinitely many oscillations, reaching -1 does not guarantee global optimality, making this benchmark less reliable for strict ranking

4.2.2 Time Analysis

Execution time highlights significant differences:

- Adaptive optimizers reminder finish within milliseconds ($\approx$ 0.001–0.01 seconds) due to very low iteration counts.
- Gradient Descent consumes substantially more time ($\approx$ 0.20–0.23 seconds) because it fails to converge early.

Thus, adaptive scaling of learning rates improves both convergence speed and computational efficiency.

5 Overall Comparison

5.1 Rosenbrock (Reliable Benchmark)

Ranking by combined accuracy and runtime:

1. Adam — best convergence and fastest effective runtime
2. RMSprop — moderate accuracy with slower convergence
3. Adagrad — stable but less accurate
4. Momentum — divergence observed
5. Gradient Descent — worst performance

5.2 Sinusoidal Function (Less Reliable Benchmark)

Several optimizers appear fast due to oscillatory stopping behavior. Adaptive methods still achieve the lowest runtime, but accuracy-based ranking alone is uncertain.

6 Conclusion

Key conclusions based on convergence behavior, stability, and execution time:

- Adaptive optimizers dominate non-convex optimization, with Adam providing the best balance of precision, stability, and speed.
- Iteration count alone is insufficient; true efficiency must consider final loss and computational time together.
- Benchmark selection is critical. The Rosenbrock function enables meaningful comparison, whereas $\sin(1/x)$ primarily evaluates robustness to oscillations.

Overall, Adam is the recommended optimizer for complex loss landscapes due to its reliable convergence and minimal runtime.

Task 02: Linear Regression Using a Multi-Layer Neural Network

1. Introduction

The objective of **Task 02** is to implement a **multi-layer neural network from scratch** to perform **linear regression** and to study the impact of different **optimizers, learning rates, network depth, and regularization** on model performance. Unlike closed-form linear regression, neural networks rely on gradient-based optimization, making optimizer choice and hyperparameter tuning crucial.

This task focuses on understanding:

- Optimizer behavior (**Gradient Descent, Momentum, Adam**)
- Effect of learning rate on convergence and generalization(**0.01, 0.001, 0.0001**)
- Impact of network depth(**2 hidden layers vs 3 hidden layers**)
- Role of **L2 regularization i**n reducing overfitting

---

2. Dataset Description

- Dataset: Boston Housing Dataset
- Target Variable: Median value of homes (MEDV)
- Input Features Used:
    - RM (Average number of rooms per dwelling)
    - CRIM (Per capita crime rate)

Data Preprocessing

- Feature normalization was applied to ensure numerical stability
- Dataset split:
    - **Training set: 80%**
    - **Test set: 20%**

🔹 Training Process

The model learns by:

1. Predicting output
2. Comparing with real value
3. Calculating error (**MSE**)
4. Updating itself to reduce error

Lower MSE = Better model

🔹 ReLU Activation Function

ReLU (Rectified Linear Unit) in **hidden layers** means:

-If the value is negative → make it 0
- If the value is positive → keep it

Why ReLU?

- Makes learning faster
- Helps the network learn complex patterns

🔹 Optimizers (How learning improves)

- **Gradient Descent: slow learning (worst)**
- **Momentum: slightly faster**
- **Adam: smart and fast (best)**

Adam gave the lowest MSE

🔹 Learning Rate

1. Controls **how fast the model learns**
2. 0.01 worked best
3. Too small → learning is very slow

🔹 Overfitting

Overfitting means:

> The model learns training data too well
> But performs badly on new data

In our task:

- **3 hidden layers** caused overfitting
- Model became too complex for small data

🔹 L2 Regularization(Overfitting Control)

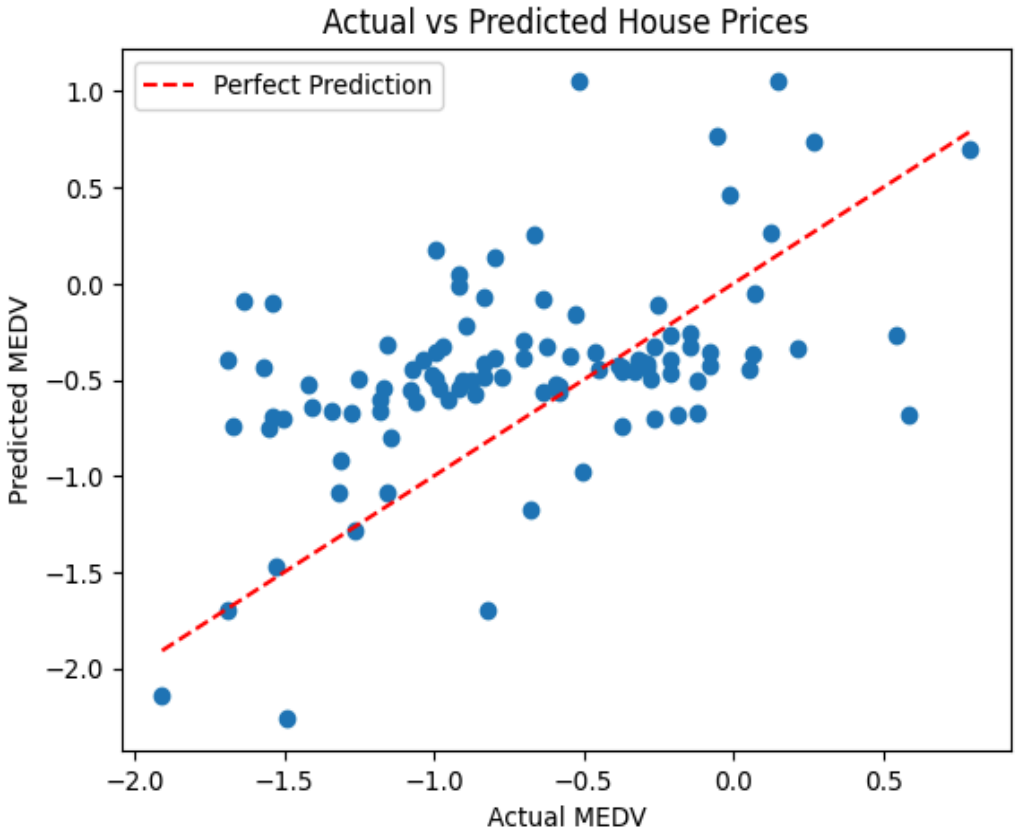
L2 Regularization means:

> Add a penalty to large weights. Forces the model to stay simple

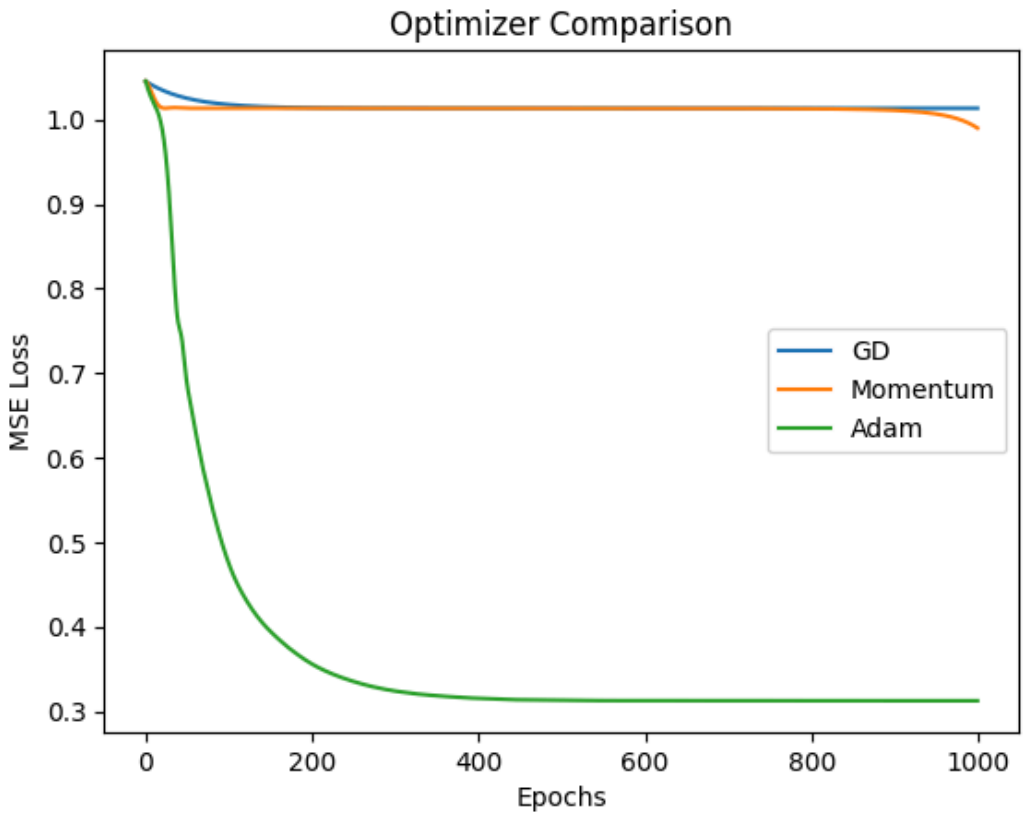
Result:

- Reduced overfitting
- Improved Test MSE

3. Visualized the regression results by plotting the Predicted vs Actual values



4. Optimizer Comparison

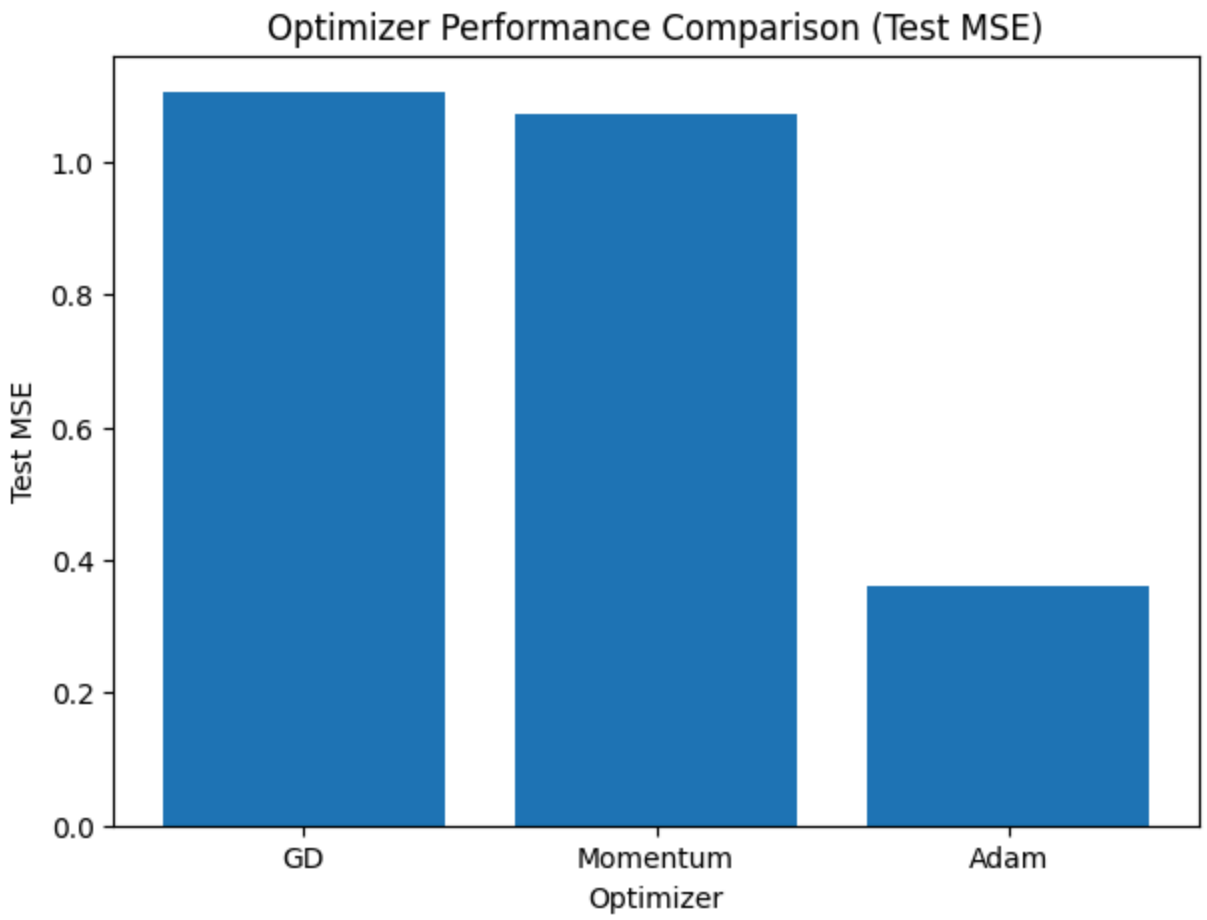


Three optimizers were evaluated using the same architecture and learning rate:

- Gradient Descent (GD)
- Gradient Descent with Momentum
- Adam

Test Mean Squared Error (MSE)

Optimizer	Test MSE
Gradient Descent	1.1054
Momentum	1.0711
Adam	**0.3595**



Observation

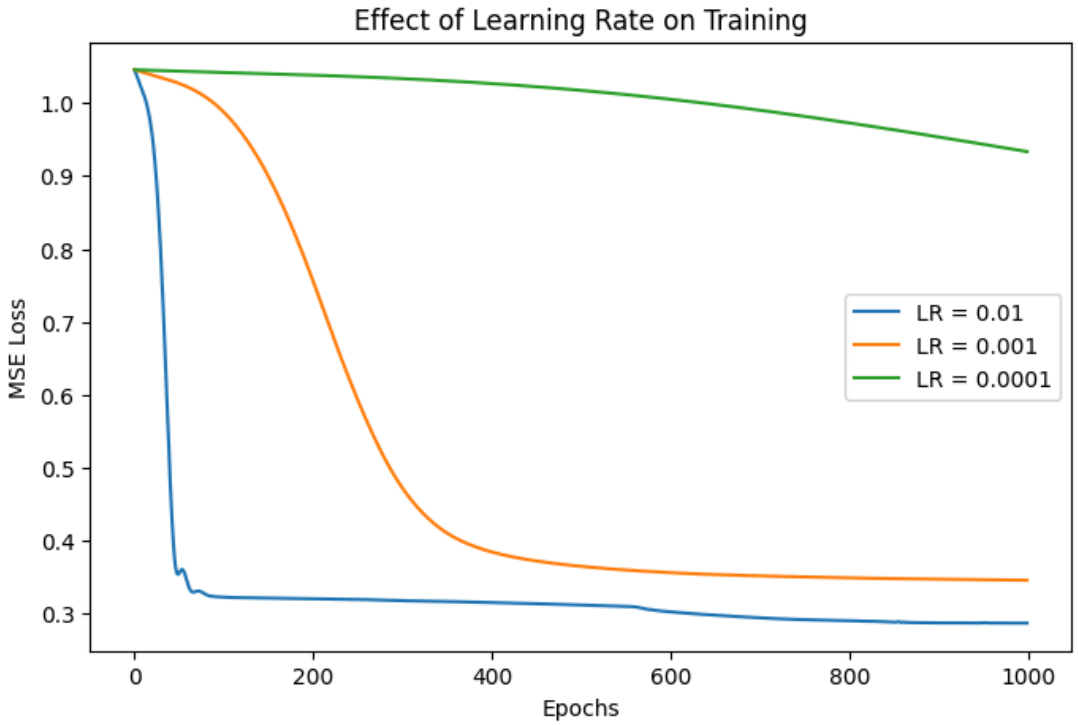
- Adam significantly outperformed GD and Momentum
- Momentum provided marginal improvement over basic GD
- Adaptive learning rates in Adam enabled faster and more stable convergence

Best Optimizer: Adam
Worst Optimizer: Gradient Descent

---

5. Learning Rate Analysis

The effect of learning rate (η) was studied using the Adam optimizer.



Training Performance Loss Comparison

Learning Rate	Final Training Loss
0.01	**0.2872**
0.001	0.3459
0.0001	0.9334

Testing Mean Sum Error Comparison

Learning Rate	Test MSE
0.01	0.3441
0.001	0.4271
0.0001	0.6330

Observation

- A higher learning rate (0.01) led to faster convergence and lower error
- Too small a learning rate resulted in slow learning and underfitting

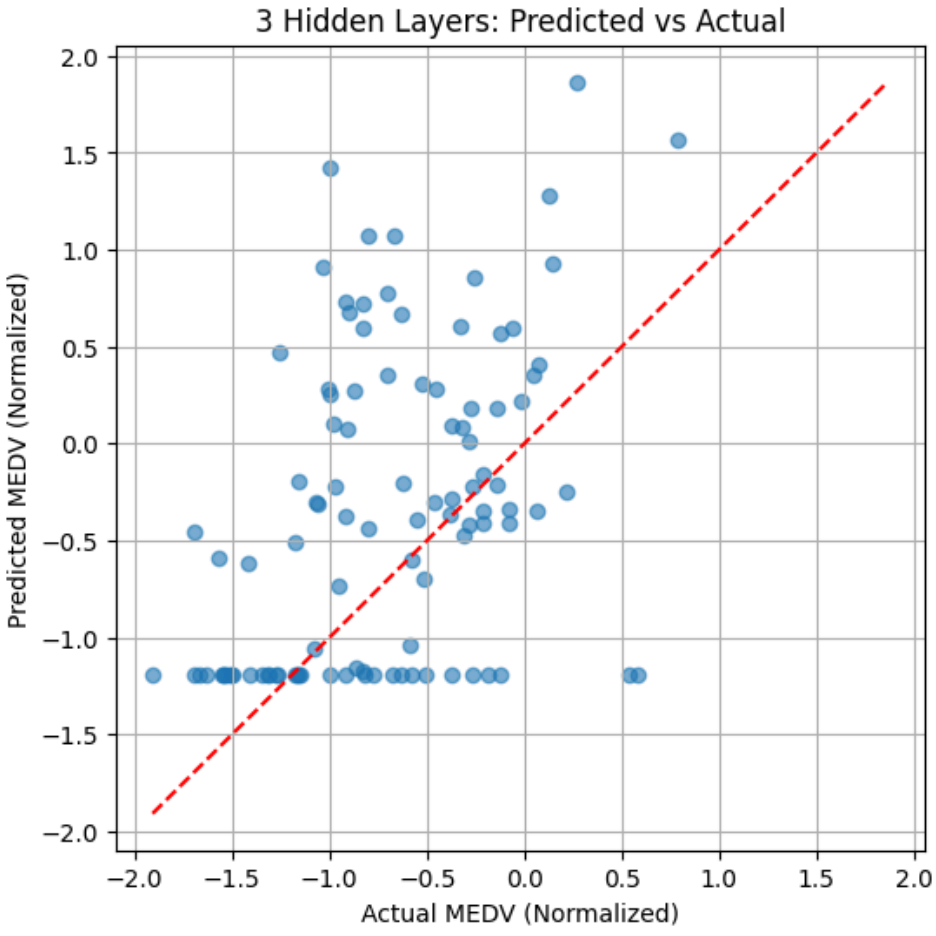
Best Learning Rate: 0.01
Worst Learning Rate: 0.0001

BONUS

1.Base Neural Network

Component	Description
Input Layer	2 neurons (RM, CRIM)
Hidden Layers	3
Layer Structure	2 → 5 → 3 → 2 → 1
Output Layer	1 neuron (continuous value)
Activation	ReLU (hidden layers)
Loss Function	Mean Squared Error (MSE)

This architecture was chosen to allow sufficient representational capacity while maintaining simplicity.

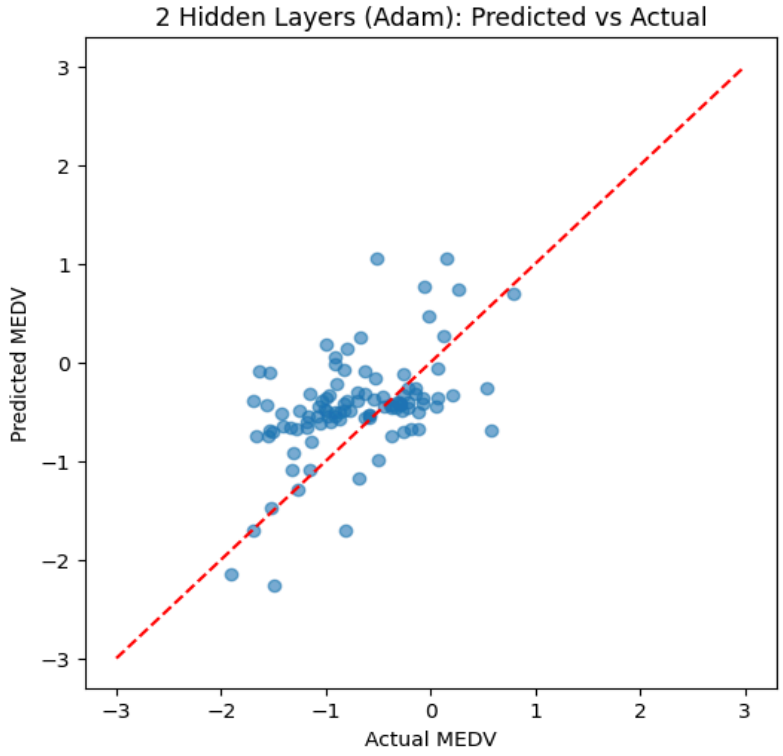


L2 Regularization Study

To mitigate overfitting, **L2 regularization (weight decay)** was applied.

2. Regularized Model Details

Parameter	Value
Architecture	2 → 5 → 3 → 1
Hidden Layers	2
Regularization	L2
λ (Lambda)	0.01
Test MSE	0.5627



Observation

- L2 regularization improved performance compared to deeper unregularized models
- However, it did not outperform the simpler 2-hidden-layer Adam model

---

6. Effect of Network Depth

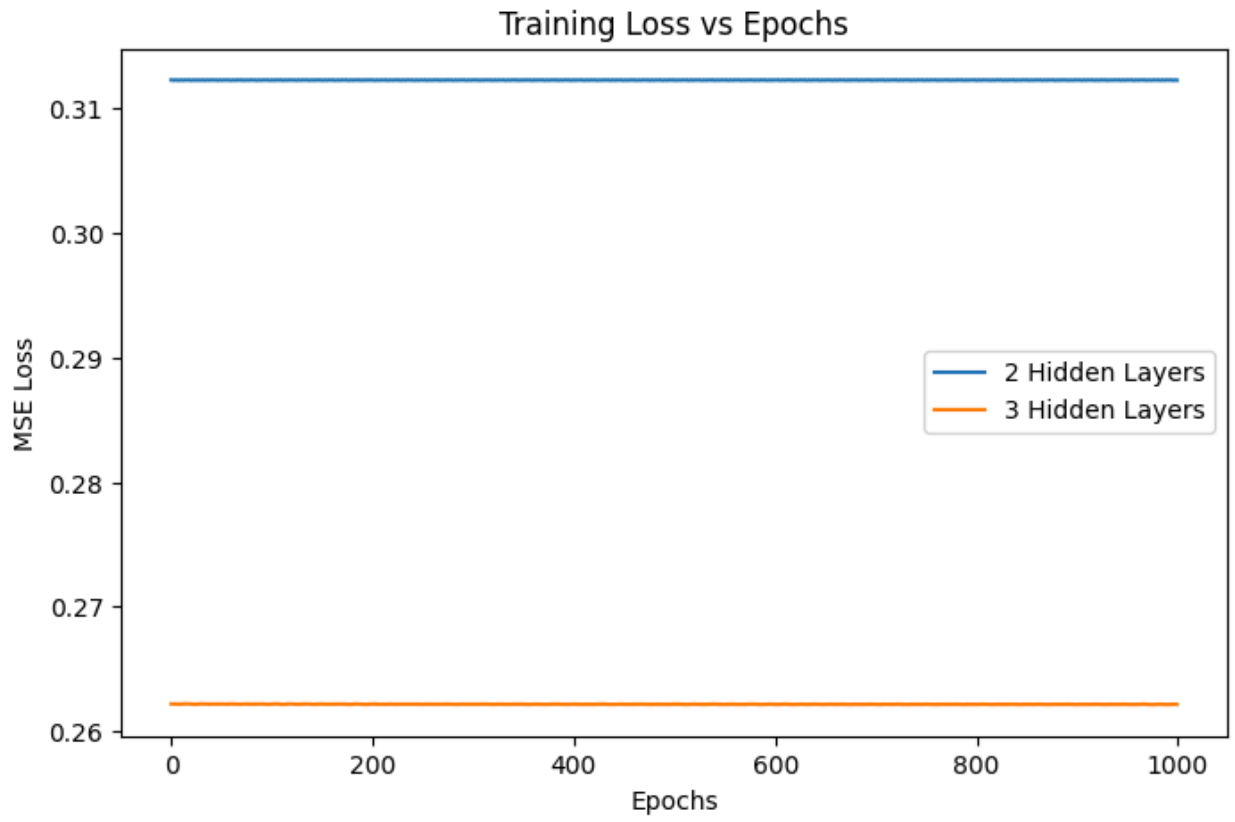
Multiple network depths were compared to analyze the impact of model complexity.

Model Performance Summary

Model Configuration	Test MSE
2 Hidden Layers (Adam)	**0.3595**
3 Hidden Layers	0.7110

Observation

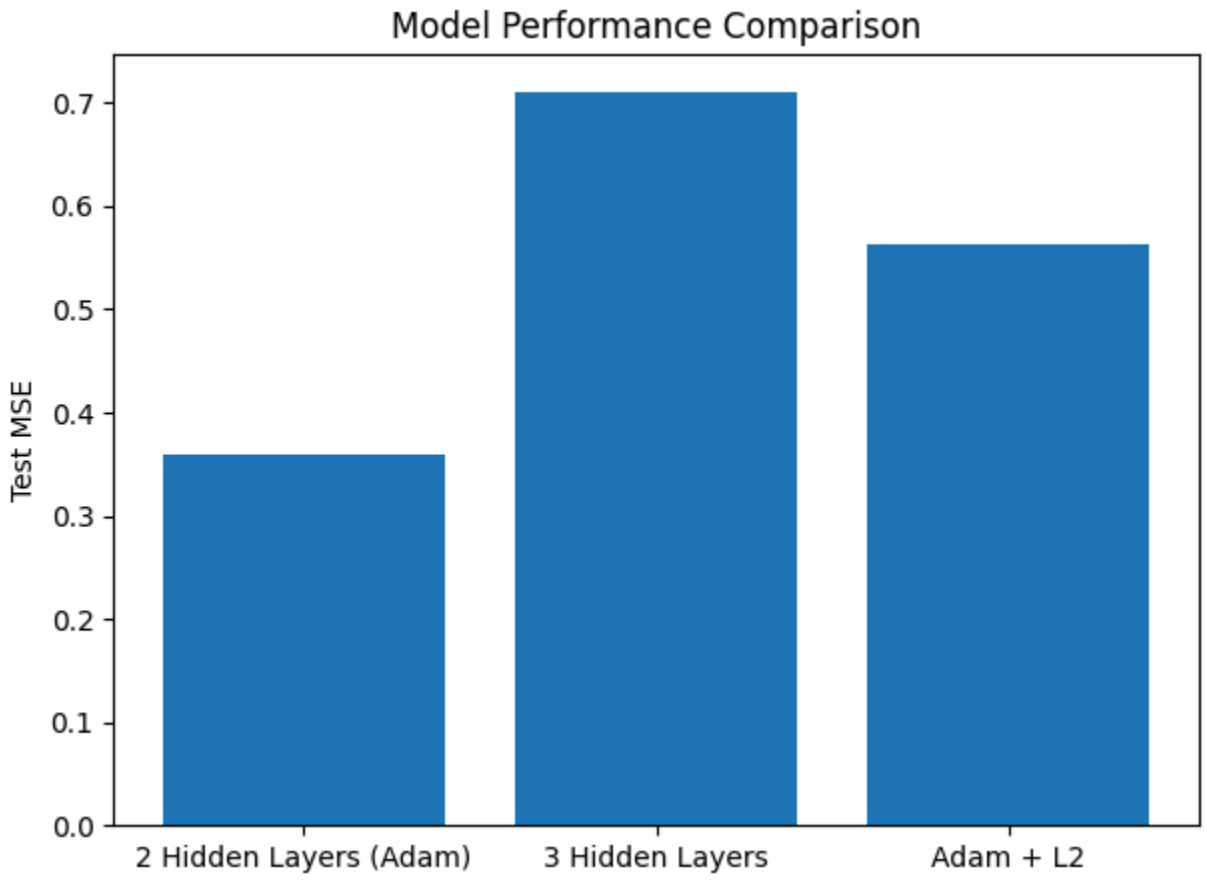
- Increasing network depth degraded performance
- Deeper network likely overfitted the limited feature space
- Simpler architecture generalized better



7. Final Model Comparison

Model	Test MSE
2 Hidden Layers (Adam)	**0.3595**
Adam + L2 Regularization	0.5627
3 Hidden Layers	0.7110

Best Model: 2 Hidden Layers with Adam optimizer
Worst Model: 3 Hidden Layers without regularization



8. Key Observations

1. **Adam optimizer** performed best with the lowest test MSE (0.3595), showing faster and more stable learning.
2. A **learning rate of 0.01** gave the best results; very small learning rates slowed learning and increased error.
3. Increasing hidden layers from 2 to 3 **reduced performance** due to overfitting.
4. **L2 regularization** helped improve generalization by controlling large weights.

---

9. Conclusion

A **simpler neural network with a good optimizer and proper learning rate** performs better than a deeper, more complex model on limited data.

Task 03 Report:
Multi-Class and Non-Linear Classification using Fully Connected Neural Networks

Bhargav

1 Introduction

This task investigates the behavior of a Fully Connected Neural Network (FCNN) on both linearly separable and non-linearly separable datasets. The goal is to analyze convergence, classification accuracy, and representational power of neural networks compared to a single-layer perceptron.

2 Dataset Description

Two synthetic datasets were generated:

- **Dataset 1 (Linear)**: Three Gaussian clusters corresponding to three classes.
- **Dataset 2 (Non-Linear)**: Two concentric circular classes.

Each dataset was split class-wise into:

- Training set: 60%
- Validation set: 20%
- Test set: 20%

3 Network Architecture

The Fully Connected Neural Network uses:

- Input layer: 2 neurons
- Hidden layers:
  - Dataset 1: One hidden layer with 8 neurons
  - Dataset 2: Two hidden layers with 16 and 8 neurons
- Output layer: Softmax-style sigmoid outputs
- Activation function: Sigmoid

4 Training Configuration

- Learning rate: 0.1
- Loss function: Mean Squared Error
- Optimization: Gradient Descent with Backpropagation
- Epochs:
  - Dataset 1: 100
  - Dataset 2: 150

5 Results: Dataset 1 (Linear Classification)

5.1 Loss Convergence

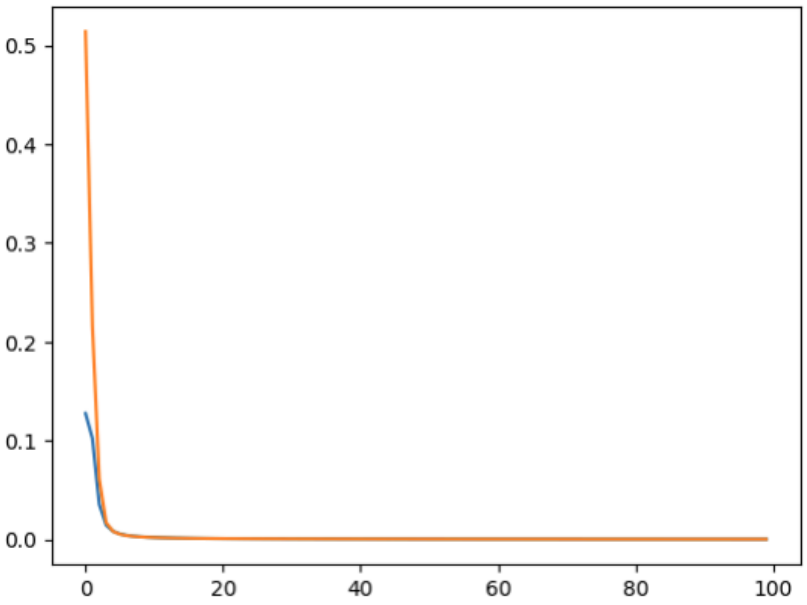


Figure 1: Enter Caption

Figure 2: Training and Validation Loss vs Epochs (Linear Dataset)

5.2 Confusion Matrix and Accuracy

Table 1: Confusion Matrix (Validation Set)

	Class 0	Class 1	Class 2
Class 0	100	0	0
Class 1	0	100	0
Class 2	0	0	100

Validation Accuracy: 1.00

5.3 Decision Boundary

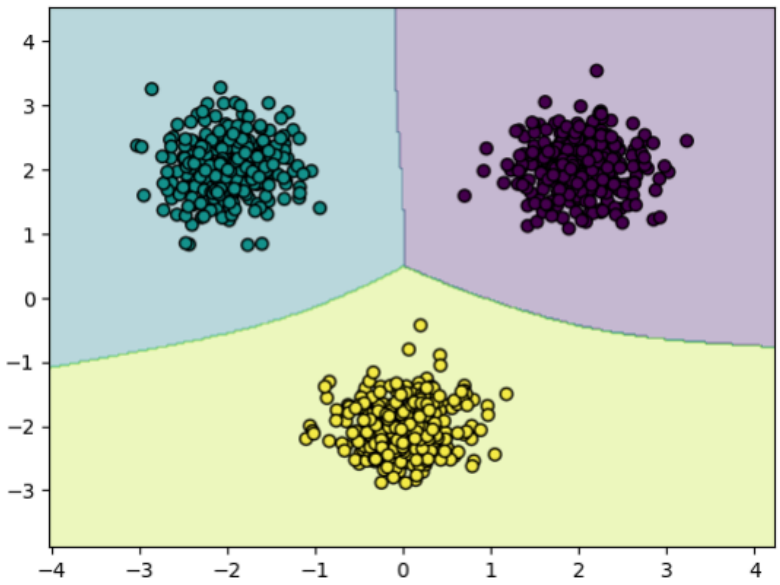


Figure 3: Decision Regions Learned by FCNN (Linear Dataset)

6 Results: Dataset 2 (Non-Linear Classification)

6.1 Loss Convergence

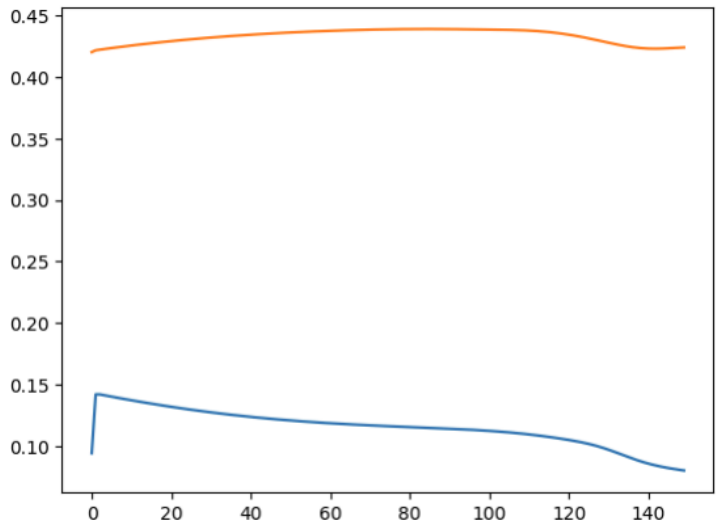


Figure 4: Training and Validation Loss vs Epochs (Non-Linear Dataset)

6.2 Confusion Matrix and Accuracy

Table 2: Confusion Matrix (Validation Set)

	Class 0	Class 1
Class 0	0	100
Class 1	0	100

Validation Accuracy: 0.50

6.3 Decision Boundary

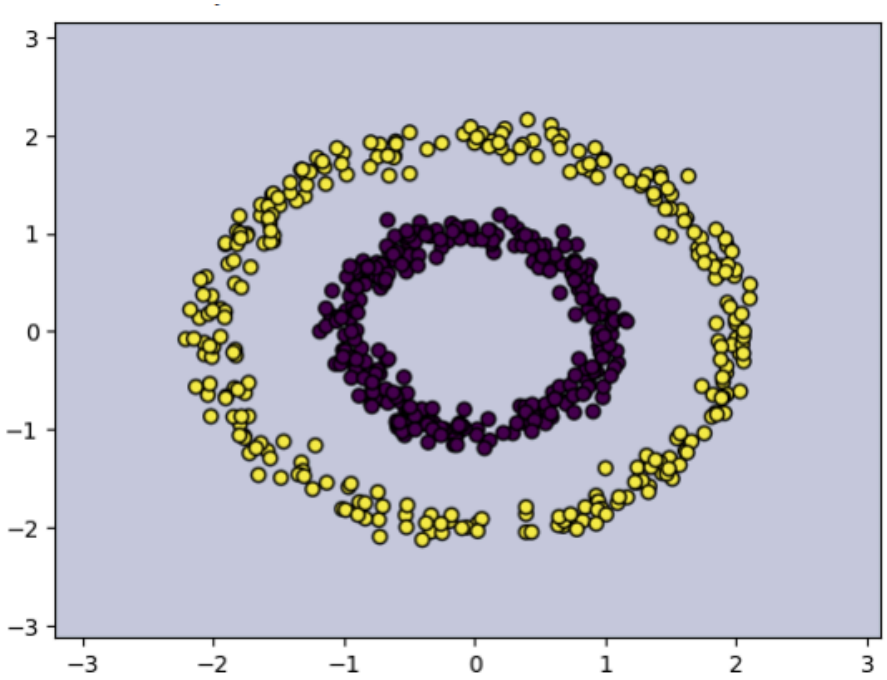


Figure 5: Enter Caption

Figure 6: Decision Regions Learned by FCNN (Non-Linear Dataset)

7 Hidden Neuron Activations

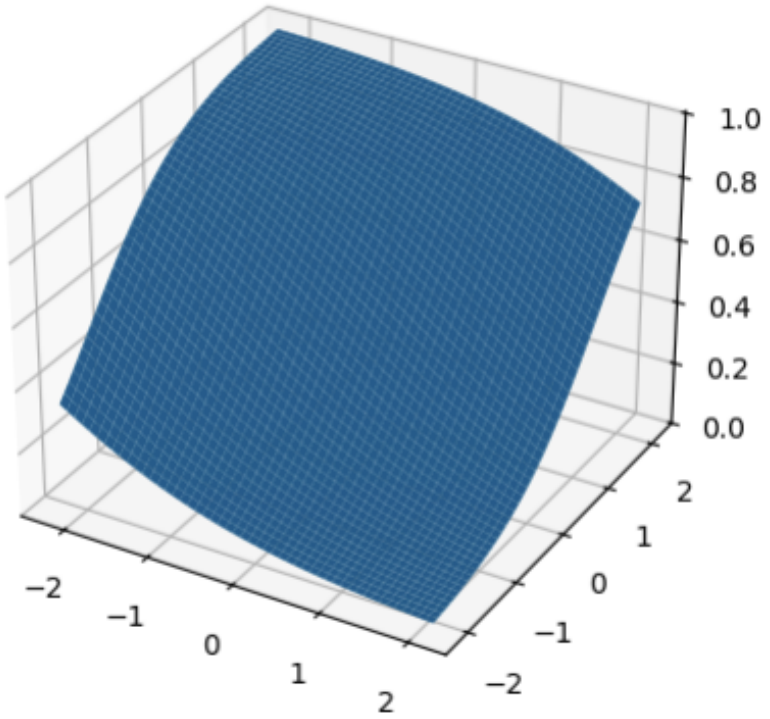


Figure 7: Enter Caption

Figure 8: Sample Hidden Neuron Activation Surfaces

8 Comparison with Perceptron

Table 3: Perceptron vs FCNN Performance

Model	Dataset 1	Dataset 2
Perceptron Accuracy	–	0.50
FCNN Accuracy	1.00	0.50

9 Key Observations

- FCNN perfectly classifies linearly separable data.
- Non-linear data requires deeper architectures and careful tuning.
- Hidden layers learn meaningful non-linear features.
- Perceptron fails on non-linearly separable data.

10 Conclusion

This experiment demonstrates the superiority of Fully Connected Neural Networks over single-layer models. While FCNN achieves perfect performance on linear data, non-linear classification remains challenging and highlights the importance of architecture depth and optimization strategy.

Task 04: Multi-class Classification on MNIST using Fully Connected Neural Networks

1. Introduction

The objective of Task 04 is to study the **behavior of different optimization algorithms** used in backpropagation while training a Fully Connected Neural Network (FCNN) on a subset of the MNIST dataset. Instead of focusing only on classification accuracy, this task emphasizes **convergence behavior, training time, and stability** of optimizers under controlled experimental conditions.

All experiments were conducted under identical settings, keeping the dataset, network architectures, learning rate, and stopping criterion fixed, while varying only the **optimizer**. This allows a fair and meaningful comparison of optimization strategies.

---

2. Dataset Description and Preprocessing

- Dataset: MNIST handwritten digit dataset
- Selected Classes: Any 5 classes (subset used consistently across experiments)
- Image Size: 28 × 28 pixels
- Input Representation: Flattened into a 784-dimensional vector
- Data Split:
    - Training set: 80%
    - Validation set: 20%

Pixel values were normalized to the range [0, 1].

---

3. Network Architectures

Three Fully Connected Neural Network architectures were evaluated:

Architecture	Layer Configuration
3 Hidden Layers	784 → 256 → 128 → 64 → 5
4 Hidden Layers	784 → 256 → 128 → 64 → 32 → 5
5 Hidden Layers	784 → 256 → 128 → 64 → 32 → 16 → 5

- Activation Function (Hidden Layers): ReLU
- Output Layer: Softmax (implicitly handled by CrossEntropyLoss)

---

4. Optimizers and Hyperparameters

The following optimizers were evaluated:

1. Stochastic Gradient Descent (SGD)
2. Batch Gradient Descent
3. SGD with Momentum
4. SGD with Nesterov Accelerated Gradient (NAG)
5. RMSProp
6. Adam

Fixed Hyperparameters

- Learning Rate (η): 0.001
- Momentum: 0.9 (Momentum, NAG)
- RMSProp: $\beta = 0.99$, $\varepsilon = 1e\text{-}8$
- Adam: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e\text{-}8$
- Batch Size:
  - Batch GD: Full dataset
  - Other optimizers: Mini-batch size = 64

Stopping Criterion

Training was stopped when:

$|L\square - L\square_{-1}| < 10^{-4}$

A maximum epoch cap was used as a safety mechanism to avoid indefinite training.

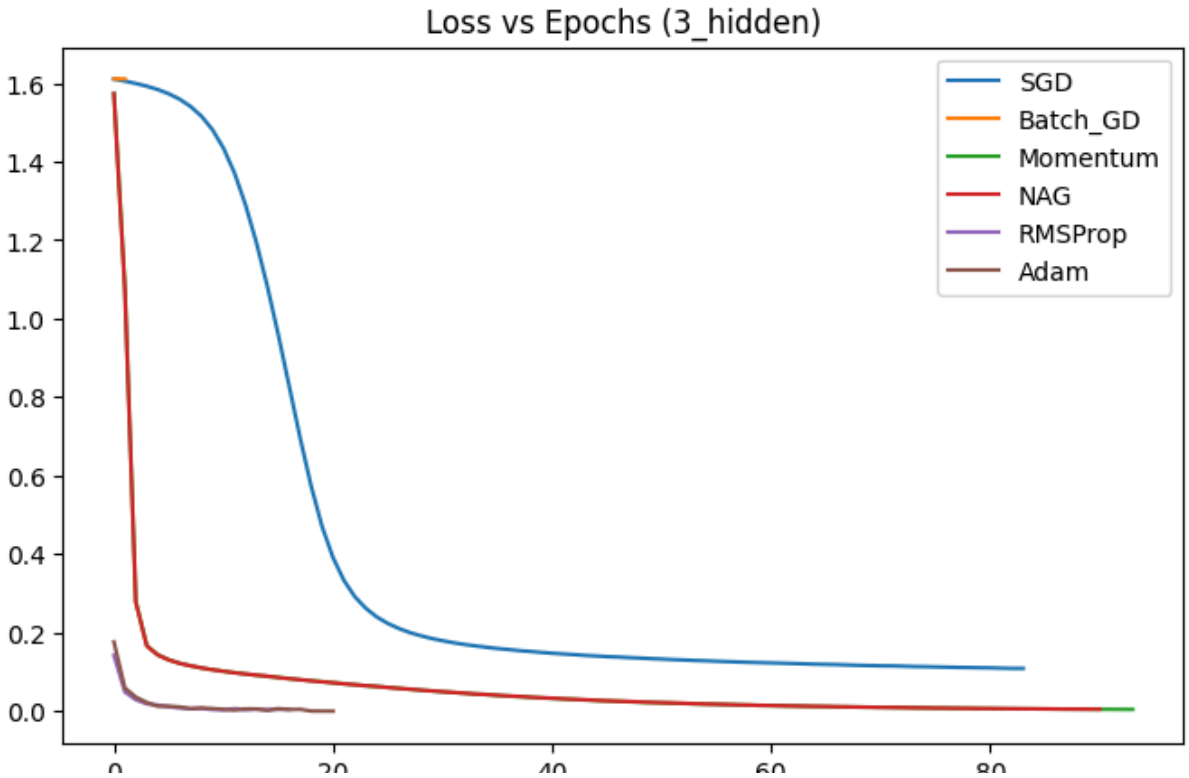
---

5. Experimental Results

5.1 Convergence Epochs, Validation Accuracy, and Training Time

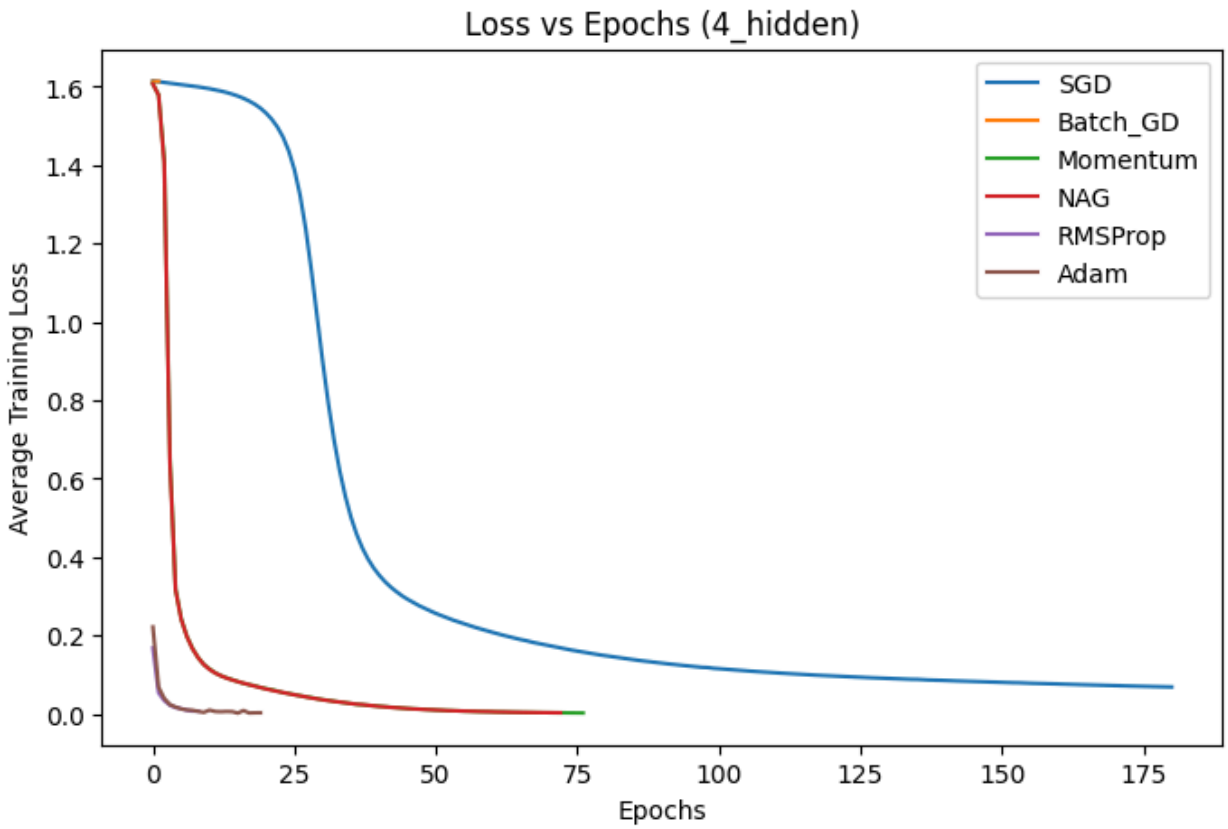
Architecture: 3 Hidden Layers

Optimizer	Epochs	Validation Accuracy	Time (s)
SGD	83	0.9663	49.88
Batch GD	1	0.1908	0.13
Momentum	93	0.9882	62.72
NAG	90	0.9881	58.70
RMSProp	16	0.9874	11.64
Adam	20	**0.9940**	15.98



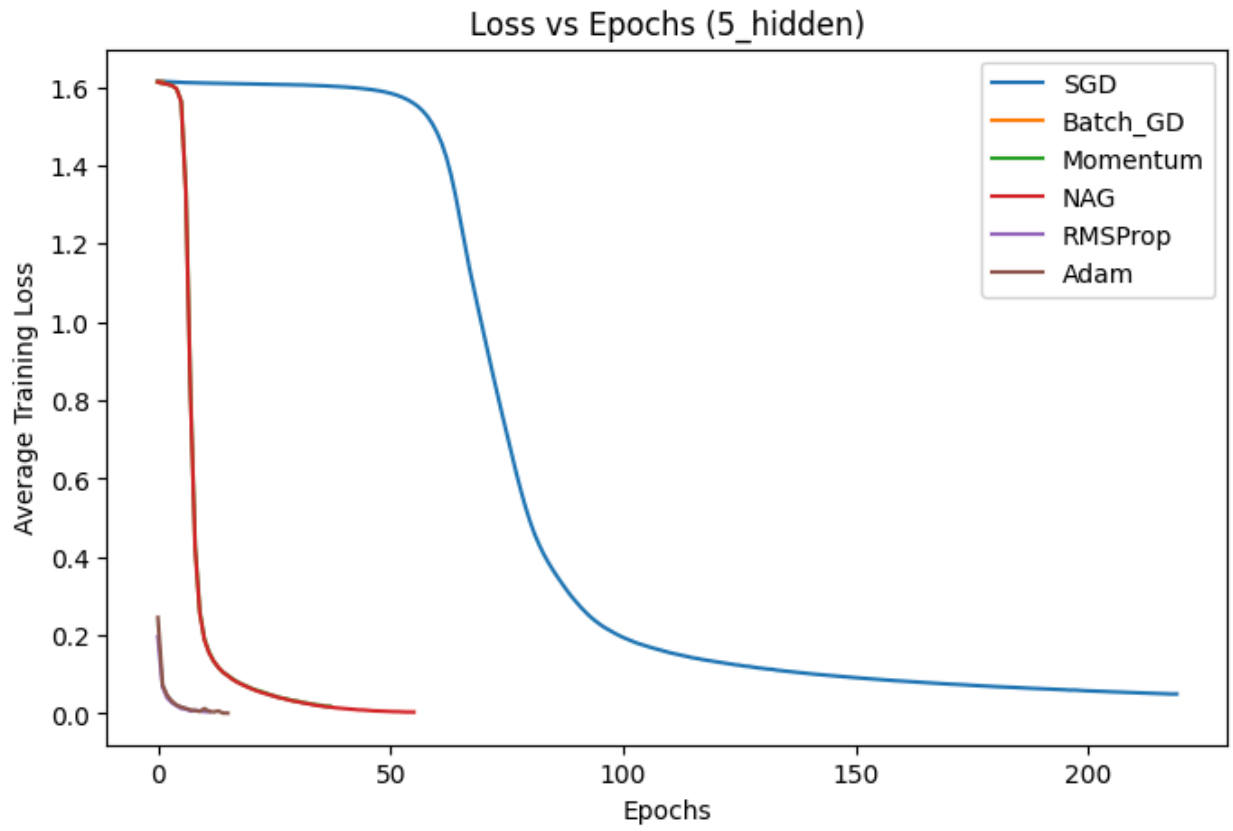
Architecture: 4 Hidden Layers

Optimizer	Epochs	Validation Accuracy	Time (s)
SGD	180	0.9753	125.97
Batch GD	1	0.1908	0.12
Momentum	76	0.9868	58.37
NAG	72	0.9869	54.14
RMSProp	8	0.9887	**7.41**
Adam	19	0.9902	18.82



Architecture: 5 Hidden Layers

Optimizer	Epochs	Validation Accuracy	Time (s)
SGD	219	0.9771	174.23
Batch GD	1	0.2003	0.09
Momentum	37	0.9809	34.84
NAG	55	0.9822	49.22
RMSProp	11	**0.9910**	10.55
Adam	15	0.9855	14.92



6. Loss vs Epoch Analysis

The following figures show **average training loss vs epochs** for each architecture. All optimizer curves are superimposed for comparison.

- Figure 1: Loss vs Epochs (3 Hidden Layers)
- Figure 2: Loss vs Epochs (4 Hidden Layers)
- Figure 3: Loss vs Epochs (5 Hidden Layers)

Observation:

- SGD exhibits slow and smooth convergence with long tails.
- Momentum and NAG reduce oscillations and converge faster than SGD.
- RMSProp and Adam show steep initial loss reduction, indicating faster convergence.
- Batch GD converges in one epoch but to a poor solution.

---

7. Architecture Comparison and Selection

Architecture	Best Optimizer	Best Validation Accuracy
3 Hidden	Adam	**0.9940**
4 Hidden	Adam	0.9902
5 Hidden	RMSProp	0.9910

Although deeper architectures reduce the number of epochs for some optimizers, the **3-hidden-layer network achieved the highest validation accuracy** with lower computational cost.

Chosen Best Architecture: 3 Hidden Layers - Adam Optimizer.

---

8. Key Observations

1. Faster convergence (fewer epochs) does not always imply better accuracy.
2. Batch Gradient Descent converges quickly but performs poorly due to premature stopping.
3. Adaptive optimizers (Adam, RMSProp) significantly outperform classical methods.
4. Increasing network depth increases representational capacity but also computational cost.
5. Optimal depth depends on task complexity; MNIST does not require very deep FCNNs.

---

9. Conclusion

This study demonstrates that optimizer choice has a substantial impact on both convergence speed and classification performance. Adam consistently provided the best trade-off between speed and accuracy, while RMSProp showed excellent convergence speed for deeper networks. The experiments also highlight that deeper architectures do not always lead to better performance, especially for relatively simple datasets like MNIST.

The final selected model is a **3-hidden-layer FCNN trained using the Adam optimizer**, which achieved the highest validation accuracy with efficient training time.
