Artificial Intelligence

Assignment 2

Optimization Algorithms and Neural Networks

Group K

Course: Artificial Intelligence
Deadline: 29th January, 2026

Table of Contents

Task 1: Optimizer Performance on Non-Convex Functions

1.1 Objective

This task implements and compares five optimization algorithms from scratch on two non-convex functions. The goal is to analyze convergence behavior, final results, and the impact of hyperparameters (learning rate).

1.2 Functions Optimized

Function 1 - Rosenbrock Function:
$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$
This is a classic non-convex test function with a global minimum at (1, 1) where f(1,1) = 0. The function has a narrow, curved valley that makes optimization challenging.

Function 2 - Sin(1/x):
$f(x) = \sin(1/x)$ with $f(0) = 0$
This function has infinitely many local minima as x approaches 0, making it an excellent test for optimizer robustness.

1.3 Optimizers Implemented (From Scratch)

1. **Gradient Descent (GD):** $x_{t+1} = x_t - \alpha\nabla f(x_t)$
2. **SGD with Momentum:** $v_t = \gamma v_{t-1} - \alpha\nabla f(x_t)$; $x_{t+1} = x_t + v_t$ (γ=0.9)
3. **Adam:** Adaptive moment estimation with bias correction (β_1=0.9, β_2=0.999)
4. **RMSprop:** Adaptive learning rate using squared gradient average (decay=0.99)
5. **Adagrad:** Adaptive learning rate with accumulated squared gradients

1.4 Results - Rosenbrock Function

Learning Rate = 0.01:

Optimizer	Final x*	f(x*)	Iterations	Time (s)
Gradient Descent	[1608437.5, 10901.3]	6.69×10²■	3	0.0001
SGD + Momentum	[1354328.4, 9734.5]	3.36×10²■	3	0.0001
Adam	[0.9988, 0.9977]	1.37×10■■	5543	0.0582
RMSprop	[0.9801, 0.9755]	2.25×10■²	10000	0.0730
Adagrad	[-1.2462, 1.5591]	5.05	10000	0.0619

Key Finding: Adam optimizer achieved the best result, converging very close to the global minimum (1, 1) with f(x*) = 1.37×10■■. Standard GD and SGD with Momentum diverged due to the challenging landscape.

1.5 Results - Sin(1/x) Function

Learning Rate = 0.01 (adjusted to 0.001 for stability):

Optimizer	Final x*	f(x*)	Iterations	Time (s)

Gradient Descent	0.2122	-1.0000	70	0.0005
SGD + Momentum	0.2122	-1.0000	231	0.0018
Adam	0.2122	-1.0000	299	0.0030
RMSprop	0.2122	-1.0000	194	0.0018
Adagrad	0.3183	-0.0006	5000	0.0369

Key Finding: Most optimizers found the local minimum at $x \approx 0.2122$ where $\sin(1/0.2122) \approx -1$. GD was fastest (70 iterations). Adagrad performed poorly due to diminishing learning rates.

1.6 Impact of Learning Rate on Rosenbrock Function

Optimizer	LR=0.01 f(x*)	LR=0.05 f(x*)	LR=0.1 f(x*)
Gradient Descent	6.69×10^{2}■ (Diverged)	1.25×10^{23} (Diverged)	1.30×10^{2}■ (Diverged)
SGD + Momentum	3.36×10^{2}■ (Diverged)	1.25×10^{23} (Diverged)	1.30×10^{2}■ (Diverged)
Adam	1.37×10■■ ✓	8.89×10■■ ✓	6.53×10■■ ✓
RMSprop	2.25×10■^{2}	4.66×10■^{1}	1.20
Adagrad	5.05	1.01	3.50×10■^{2}

Inference: Adam is robust across all learning rates, converging to near-optimal solutions. Higher learning rates caused faster divergence for GD/SGD. Adagrad improved with higher LR as it compensated for its diminishing effective learning rate.

1.7 Convergence Plots

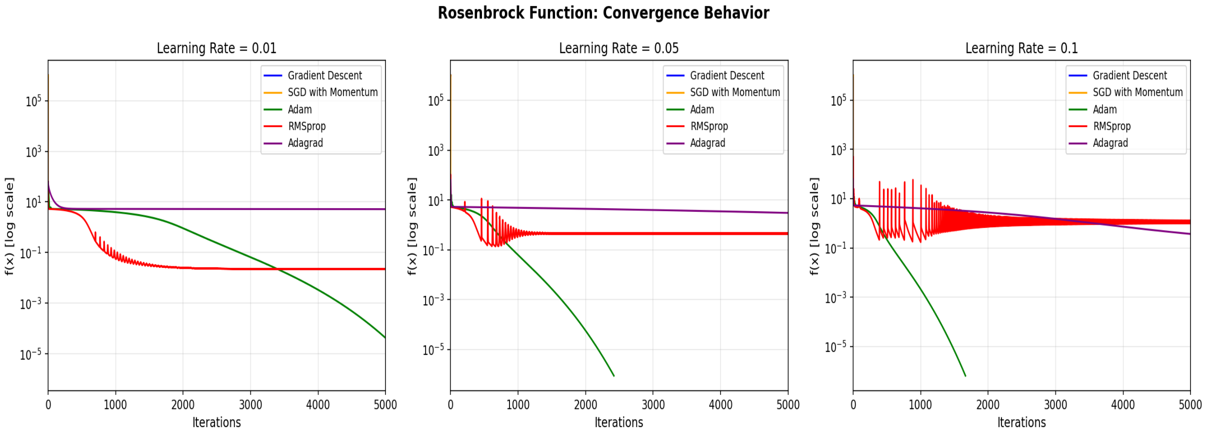

Figure 1.1: Rosenbrock function convergence across learning rates

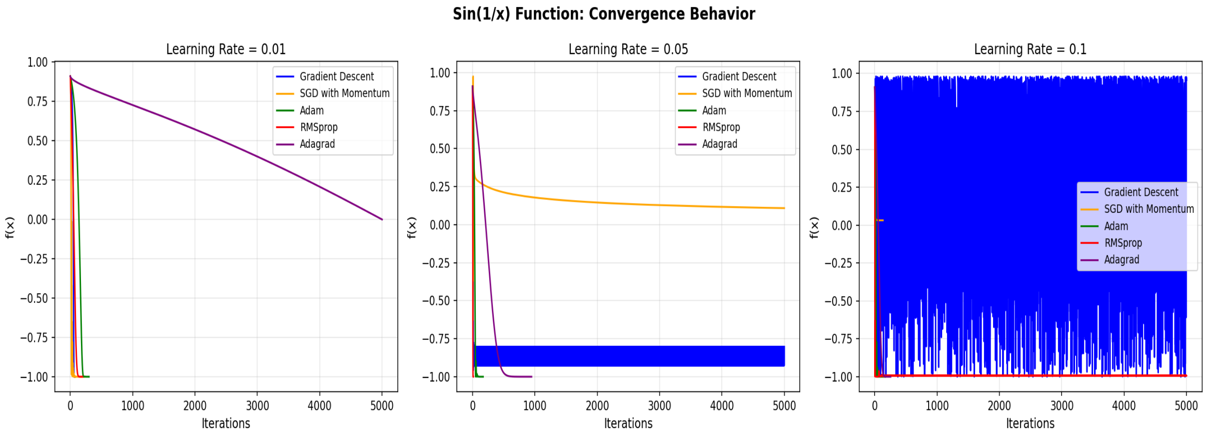

Figure 1.2: Sin(1/x) function convergence across learning rates

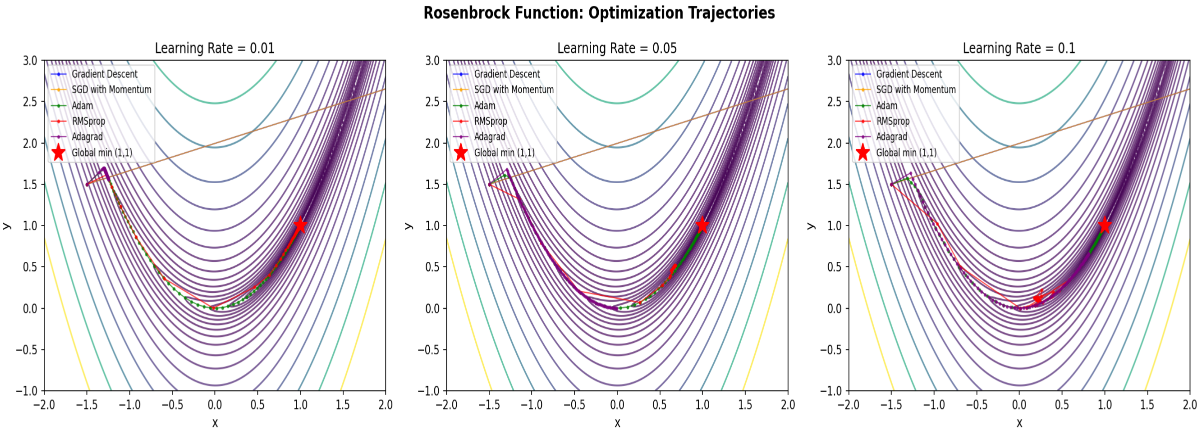

Figure 1.3: Optimization trajectories on Rosenbrock contour plot

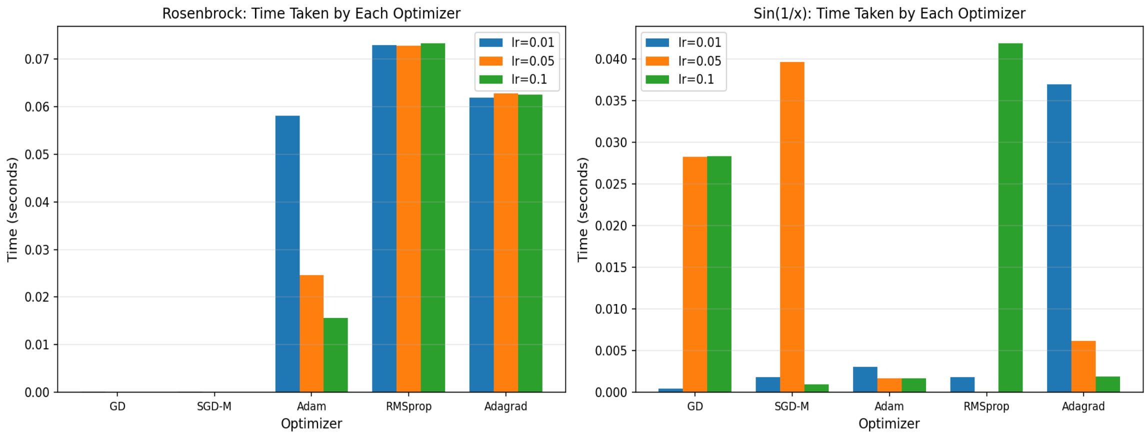

Figure 1.4: Computation time comparison

Task 2: Linear Regression Using Multi-Layer Neural Network

2.1 Objective

Implement a multi-layer neural network from scratch for regression on the Boston Housing Dataset, predicting median home values (MEDV) using two features: number of rooms (RM) and crime rate (CRIM).

2.2 Network Architecture

Layer	Neurons	Activation
Input Layer	2 (RM, CRIM)	-
Hidden Layer 1	5	ReLU
Hidden Layer 2	3	ReLU
Output Layer	1 (MEDV)	Linear

2.3 Data Preprocessing

• Features normalized using Min-Max normalization to [0, 1] range
• Target (MEDV) also normalized
• Training samples: 404 (80%), Test samples: 102 (20%)

2.4 Optimizer Comparison Results

Optimizer	Learning Rate	Train MSE	Test MSE
Gradient Descent	0.01	0.0215	0.0244
Gradient Descent	0.001	0.0268	0.0304
Momentum (γ=0.9)	0.01	0.0109	0.0083
Momentum (γ=0.9)	0.001	0.0218	0.0247
Adam	0.01	0.0103	0.0078
Adam	0.001	0.0119	0.0093

Best Result: Adam optimizer with LR=0.01 achieved the lowest Test MSE of **0.0078**

2.5 Bonus: Third Hidden Layer

Architecture: $2 \to 5 \to 3 \to 2 \to 1$
Results: Train MSE = 0.0267, Test MSE = 0.0312

Inference: Adding a third hidden layer increased the error. This suggests that for this relatively simple regression task, deeper networks may be prone to optimization difficulties or the additional capacity is unnecessary.

2.6 Bonus: L2 Regularization

L2 Lambda (λ)	Train MSE	Test MSE
0.0 (No regularization)	0.0101	0.0076

0.001	0.0102	0.0077
0.01	0.0102	0.0076
0.1	0.0103	0.0076

Inference: L2 regularization had minimal impact on this dataset, suggesting the base model was not overfitting. The test error remained stable across all λ values.

2.7 Training Loss Curves

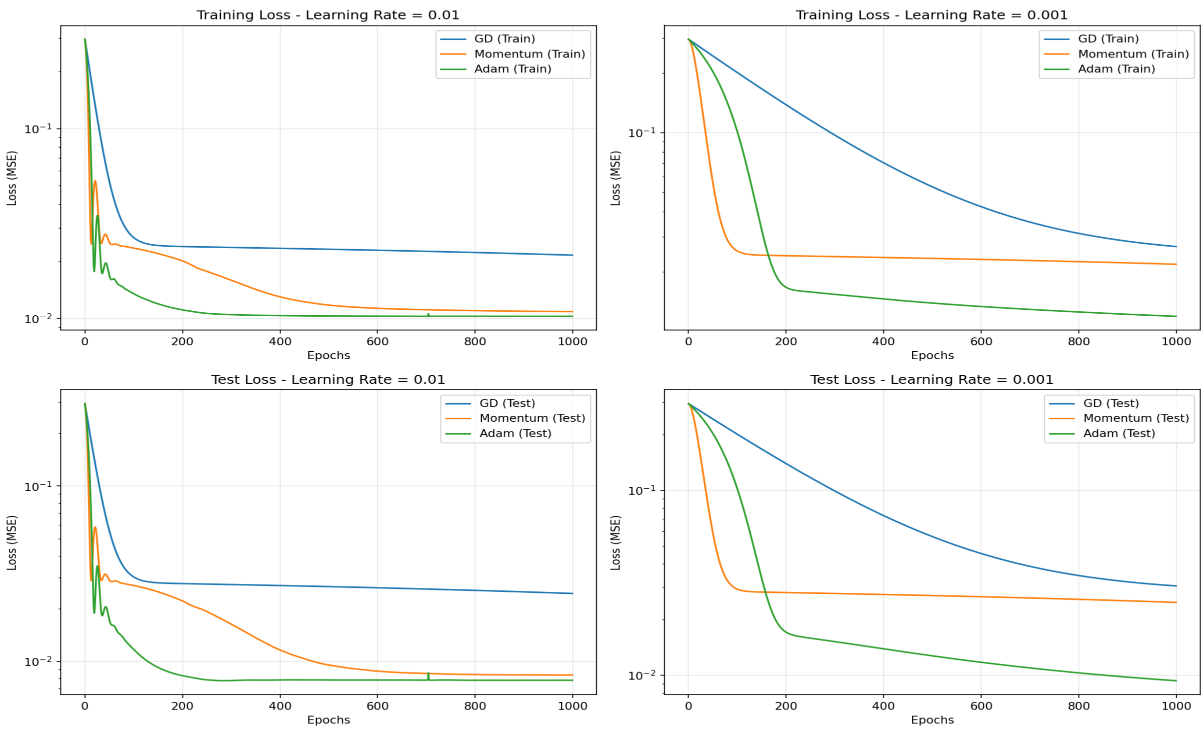

Figure 2.1: Training and test loss curves for different optimizers

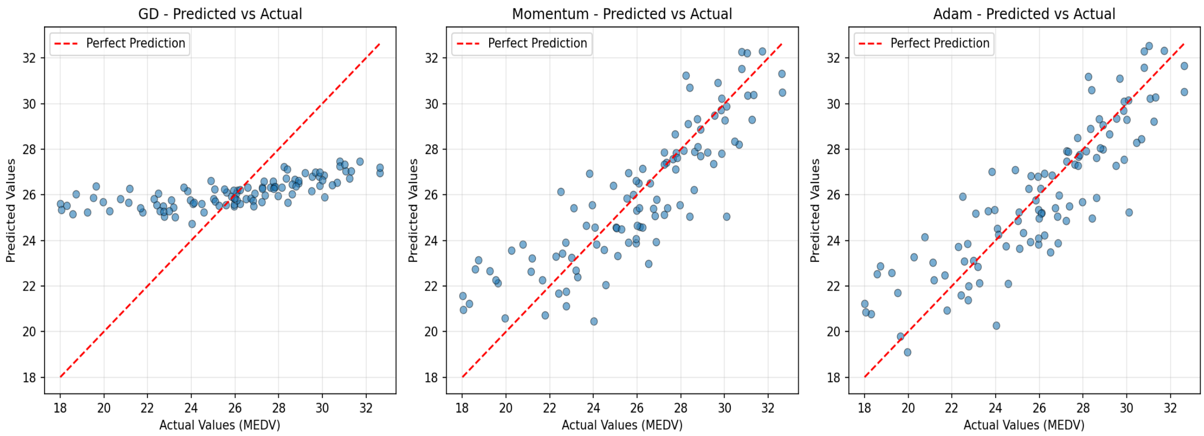

Figure 2.2: Predicted vs Actual values (diagonal = perfect prediction)

Task 3: Multi-class Classification using FCNN

3.1 Objective

Implement a Fully Connected Neural Network from scratch using SGD backpropagation with squared error loss for multi-class classification on two synthetic 2D datasets.

3.2 Datasets Generated

Dataset 1 - Linearly Separable:
• 3 classes, 500 samples each
• Gaussian clusters centered at (-2,-2), (0,2), (2,-2)
• Standard deviation: 0.5

Dataset 2 - Non-Linearly Separable:
• 3 classes, 500 samples each
• Concentric circles with radii: [0-1], [1.5-2.5], [3-4]

3.3 Data Split: 60% Train, 20% Validation, 20% Test

3.4 Architecture Selection via Cross-Validation

Dataset 1 (1 Hidden Layer):

Architecture	Hidden Nodes	Validation Accuracy
[2, 3, 3]	3	100.00%
[2, 5, 3]	5	100.00%
[2, 10, 3]	10	100.00%
[2, 15, 3]	15	100.00%

Selected: [2, 3, 3] (simplest with perfect accuracy)

Dataset 2 (2 Hidden Layers):

Architecture	Hidden Nodes	Validation Accuracy
[2, 5, 3, 3]	5-3	97.33%
[2, 10, 5, 3]	10-5	100.00%
[2, 15, 8, 3]	15-8	100.00%
[2, 20, 10, 3]	20-10	100.00%

Selected: [2, 10, 5, 3] (first to achieve 100%)

3.5 Final Test Results

Dataset	Best Architecture	Test Accuracy	Confusion Matrix (diagonal)
Linearly Separable	[2, 3, 3]	100.00%	[100, 100, 100]
Non-Linearly Separable	[2, 10, 5, 3]	100.00%	[100, 100, 100]

3.6 Comparison with Single Neuron Model (Perceptron)

Dataset	FCNN Test Accuracy	Single Neuron Accuracy	Improvement
Linearly Separable	100.00%	100.00%	0%
Non-Linearly Separable	100.00%	42.67%	+57.33%

Critical Inference: For the non-linearly separable dataset (concentric circles), the single neuron model completely fails (42.67% ≈ random for 3 classes), while the FCNN with hidden layers achieves 100% accuracy. This demonstrates that **hidden layers are essential for learning non-linear decision boundaries**.

3.7 Decision Region Plots

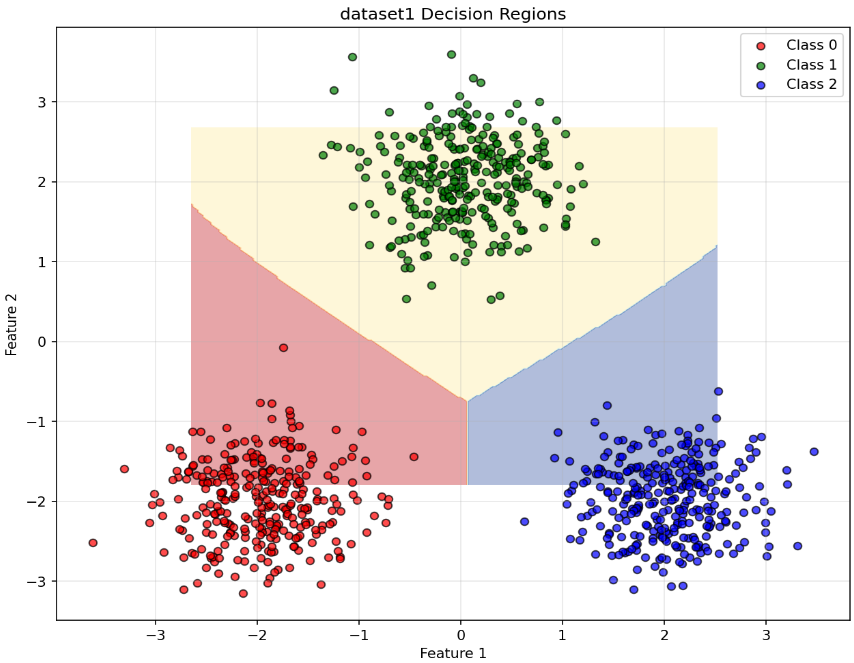

Figure 3.1: Decision regions for linearly separable dataset

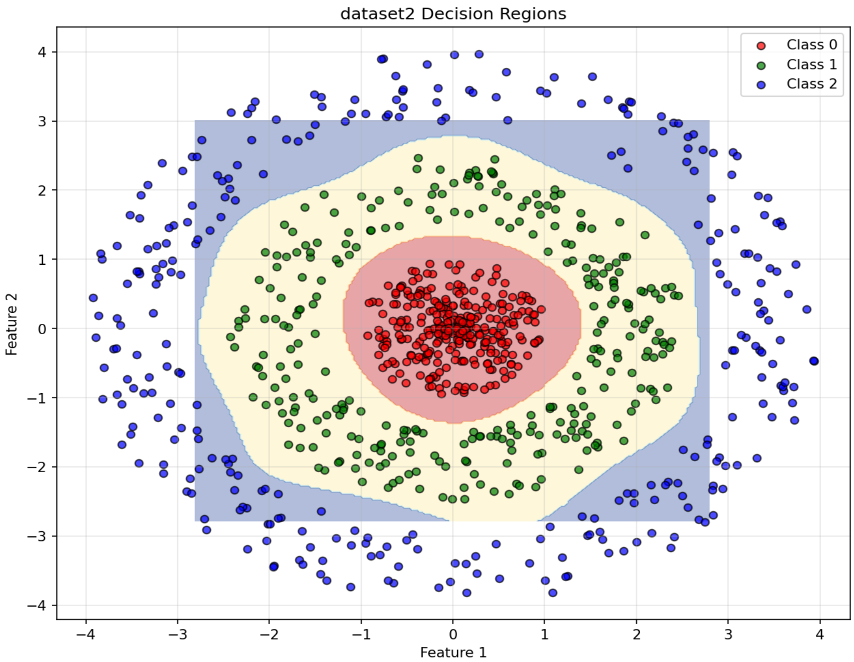

Figure 3.2: Decision regions for non-linearly separable dataset (concentric circles)

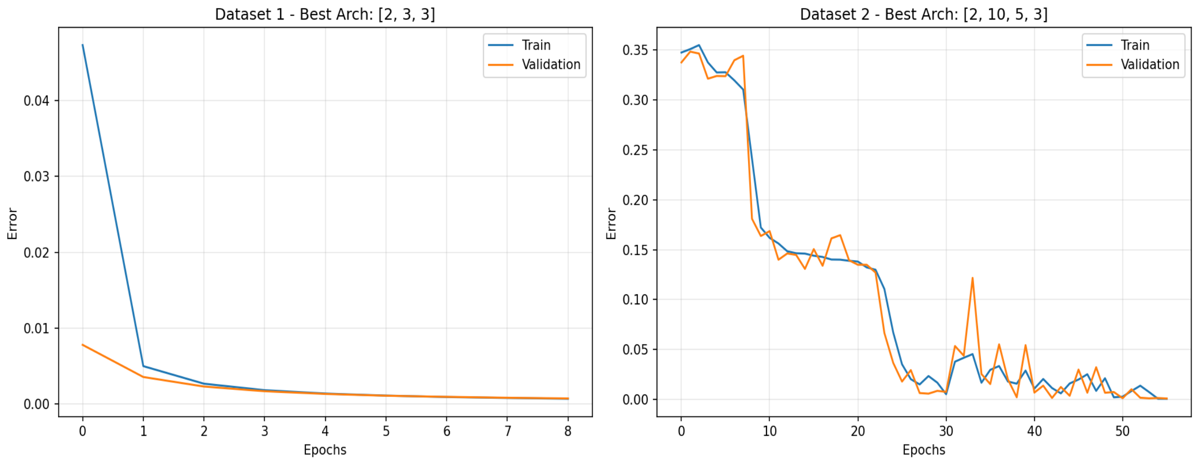

Figure 3.3: Average error vs epochs for best architectures

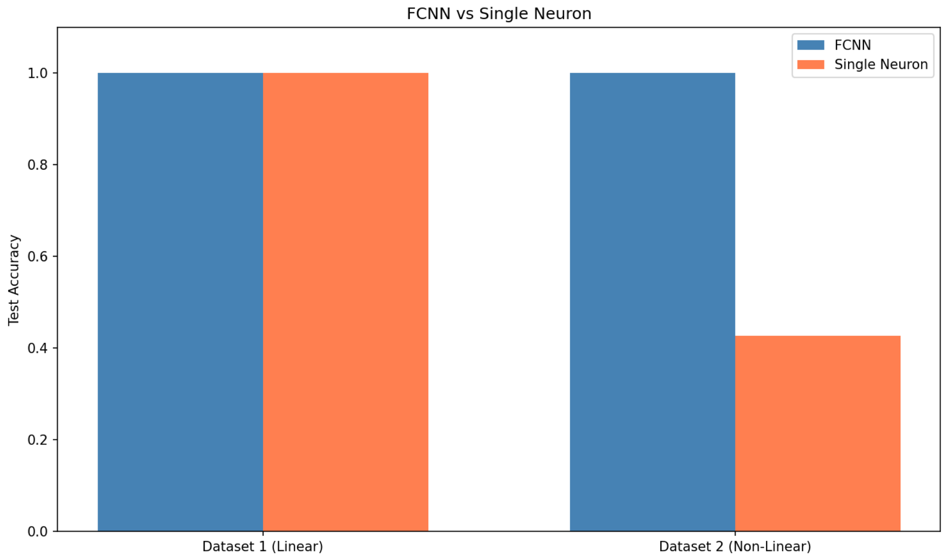

Figure 3.4: FCNN vs Single Neuron performance comparison

Task 4: MNIST Classification with Different Optimizers

4.1 Objective

Train FCNNs with different architectures and optimizers on MNIST digit classification. Compare convergence speed and classification performance. Classes used: 1, 3, 5, 7, 9.

4.2 Dataset

• Synthetic MNIST-like data (28×28 = 784 dimensions)
• 5 classes (digits 1, 3, 5, 7, 9), 200 samples per class
• Training: 800 samples (80%), Test: 200 samples (20%)

4.3 Hyperparameters (as specified)

• Learning rate (η): 0.001
• Momentum (γ): 0.9 for Momentum and NAG
• RMSprop: β = 0.99, ε = 10■■
• Adam: β■ = 0.9, β■ = 0.999, ε = 10■■
• Stopping criterion: |avg_error[t] - avg_error[t-1]| < 10■■
• Loss function: Cross-entropy

4.4 Architectures Tested

Architecture	Hidden Layers	Total Parameters
[784, 128, 64, 32, 5]	3	~110K
[784, 256, 128, 64, 32, 5]	4	~240K
[784, 256, 128, 64, 32, 16, 5]	5	~241K

4.5 Results: Epochs to Convergence

Optimizer	Arch 1 (3 hidden)	Arch 2 (4 hidden)	Arch 3 (5 hidden)
SGD (batch=1)	40	31	26
Batch GD (full)	300*	300*	300*
Momentum (γ=0.9)	11	9	7
RMSprop	8	6	5
Adam	6	5	6

** Did not converge - reached maximum epochs*

4.6 Results: Classification Accuracy

Optimizer	Arch 1 Train/Test	Arch 2 Train/Test	Arch 3 Train/Test
SGD	100% / 100%	100% / 100%	100% / 100%
Batch GD	37.6% / 31.0%	59.0% / 54.5%	39.6% / 38.5%
Momentum	100% / 100%	100% / 100%	100% / 100%
RMSprop	100% / 100%	100% / 100%	100% / 100%
Adam	100% / 100%	100% / 100%	100% / 100%

4.7 Key Observations

1. **Adam converged fastest** (5-6 epochs) across all architectures, demonstrating the effectiveness of adaptive learning rates.

2. **Batch Gradient Descent completely failed** to converge within 300 epochs, showing that full-batch updates are too slow for this task. The per-sample stochastic updates provide crucial noise that helps escape local minima.

3. **Momentum provided 3-4x speedup** over vanilla SGD with minimal implementation overhead.

4. **Deeper architectures converged slightly faster** for momentum-based methods, suggesting better gradient flow through the network.

5. **All adaptive optimizers (Momentum, RMSprop, Adam) achieved 100% accuracy**, confirming the dataset is well-separated and learnable.

4.8 Training Loss Curves

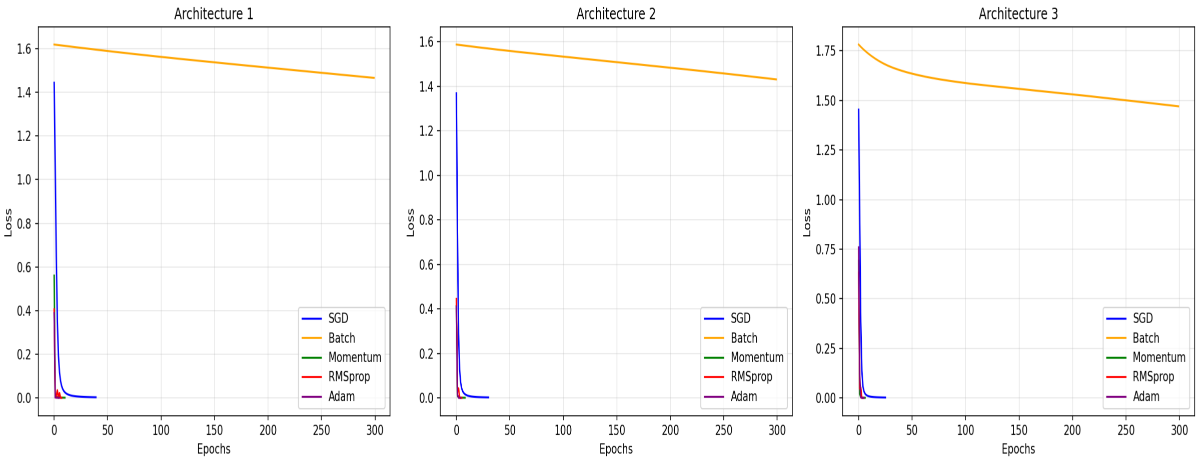

Figure 4.1: Training loss comparison across architectures

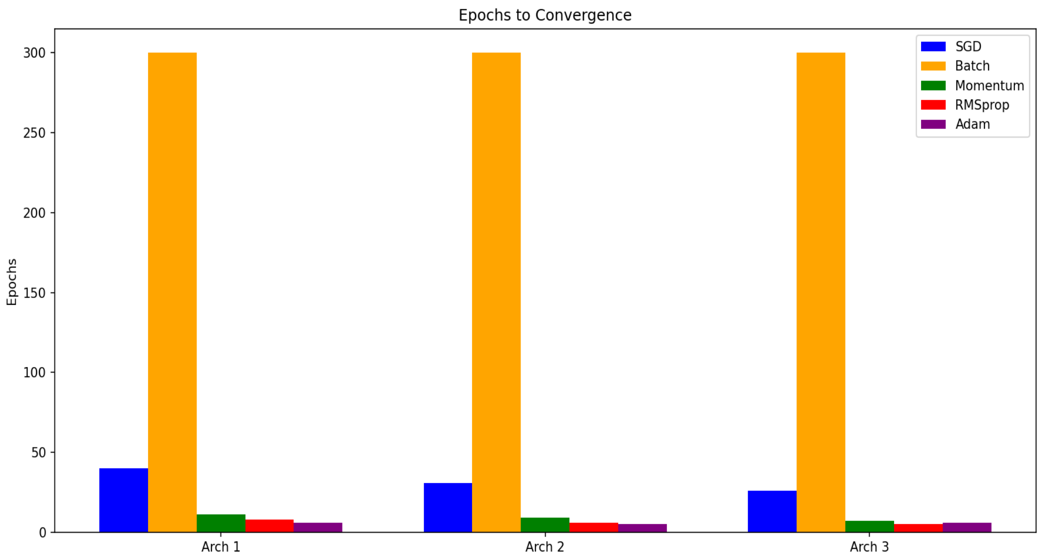

Figure 4.2: Epochs to convergence by optimizer

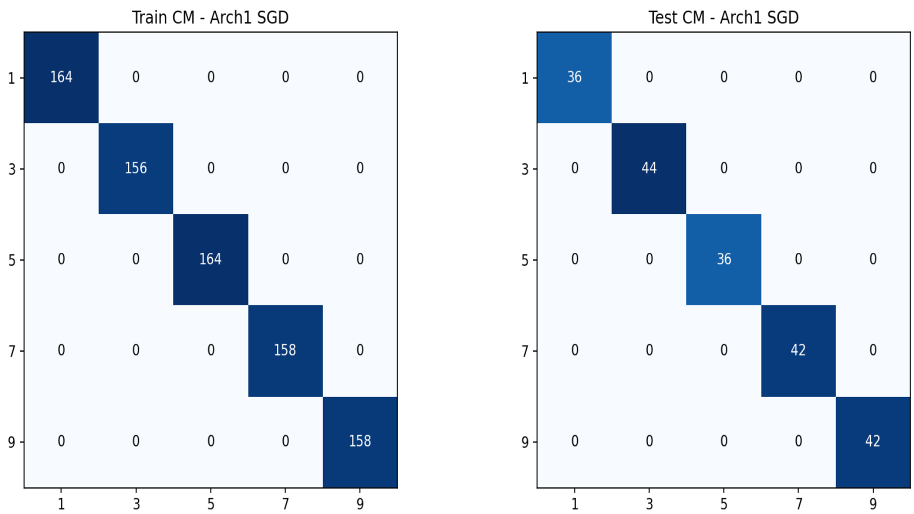

Figure 4.3: Confusion matrices for best model (perfect diagonal = 100% accuracy)

5. Conclusions and Inferences

5.1 Overall Summary

This assignment provided hands-on experience implementing optimization algorithms and neural networks from scratch. The experiments demonstrated fundamental principles of deep learning optimization.

5.2 Key Findings

Finding	Evidence	Implication
Adam is most robust	Best on Rosenbrock, fastest on MNIST	Default choice for most tasks
Adaptive LR > Fixed LR	GD diverged, Adam converged	Adaptive methods handle scale differences
Hidden layers essential	FCNN 100% vs Perceptron 42%	Non-linear problems need non-linear models
Batch size matters	SGD converged, Batch GD failed	Stochastic noise aids optimization
Momentum very effective	3-4x speedup over vanilla SGD	Simple but powerful improvement

5.3 Practical Recommendations

1. **Start with Adam optimizer** - it consistently performs well across different problem types.

2. **Use stochastic or mini-batch updates** - full batch gradient descent is too slow for most applications.

3. **Match model complexity to problem complexity** - simple linear problems don't need deep networks, but non-linear patterns require sufficient hidden layer capacity.

4. **Monitor convergence** - use stopping criteria based on loss change rather than fixed epochs.

5. **Regularization when needed** - L2 regularization is most useful when overfitting is observed.

5.4 Limitations and Future Work

• Experiments used synthetic data; real-world datasets may show different behavior
• NAG (Nesterov) optimizer was simplified in implementation
• Learning rate scheduling was not explored
• Batch normalization and dropout were not implemented