AI Assignment 2

Artificial Intelligence Course

Group K

January 2026

Table of Contents

1. Task 1: Optimizer Performance on Non-Convex Functions 3
2. Task 2: Linear Regression Using Multi-Layer Neural Network 8
3. Task 3: Multi-class Classification using FCNN 13
4. Task 4: MNIST Classification with Different Optimizers 18

Task 1: Optimizer Performance on Non-Convex Functions

Objective:

To implement and compare the performance of various optimization algorithms on two non-convex functions: the Rosenbrock function and sin(1/x). All optimizers were implemented from scratch using Python.

Functions Optimized:

1. **Rosenbrock Function:** $f(x,y) = (1-x)^2 + 100(y-x^2)^2$
Global minimum at (1, 1) with f(1,1) = 0

2. **Sin(1/x) Function:** f(x) = sin(1/x) with f(0) = 0
This function has infinitely many local minima as x approaches 0.

Optimizers Implemented:

- Gradient Descent (GD)
- Stochastic Gradient Descent with Momentum (SGD-M)
- Adam (Adaptive Moment Estimation)
- RMSprop
- Adagrad

Results - Rosenbrock Function (Learning Rate = 0.01):

Optimizer	Final x*	f(x*)	Iterations
Gradient Descent	[1608437.5, 10901.3]	6.69e+26	3
SGD with Momentum	[1354328.4, 9734.5]	3.36e+26	3
Adam	[0.9988, 0.9977]	1.37e-06	5543
RMSprop	[0.9801, 0.9755]	2.25e-02	10000
Adagrad	[-1.2462, 1.5591]	5.05e+00	10000

Key Observations:

1. **Adam optimizer performed best** on the Rosenbrock function, converging to the global minimum (1, 1) with f(x*) = 1.37e-06, demonstrating excellent performance on this challenging non-convex landscape.

2. **Standard Gradient Descent and SGD with Momentum diverged** on the Rosenbrock function at learning rate 0.01. This is expected as the Rosenbrock function has a narrow curved valley where the gradient can be misleading.

3. **For the sin(1/x) function**, most optimizers found local minima at $x \approx 0.212$, where $\sin(1/0.212) \approx -1$ (a local minimum). This demonstrates how non-convex optimization often finds local rather than global optima.

4. **Adagrad showed slow convergence** due to its accumulating squared gradients, which can make the effective learning rate too small over time.

Impact of Hyperparameters:

Learning rate significantly impacts convergence behavior. Higher learning rates (0.05, 0.1) caused faster divergence for standard GD on Rosenbrock but allowed Adam to converge faster. The adaptive learning rate methods (Adam, RMSprop) showed more robustness across different learning rate settings.

Convergence Plots:

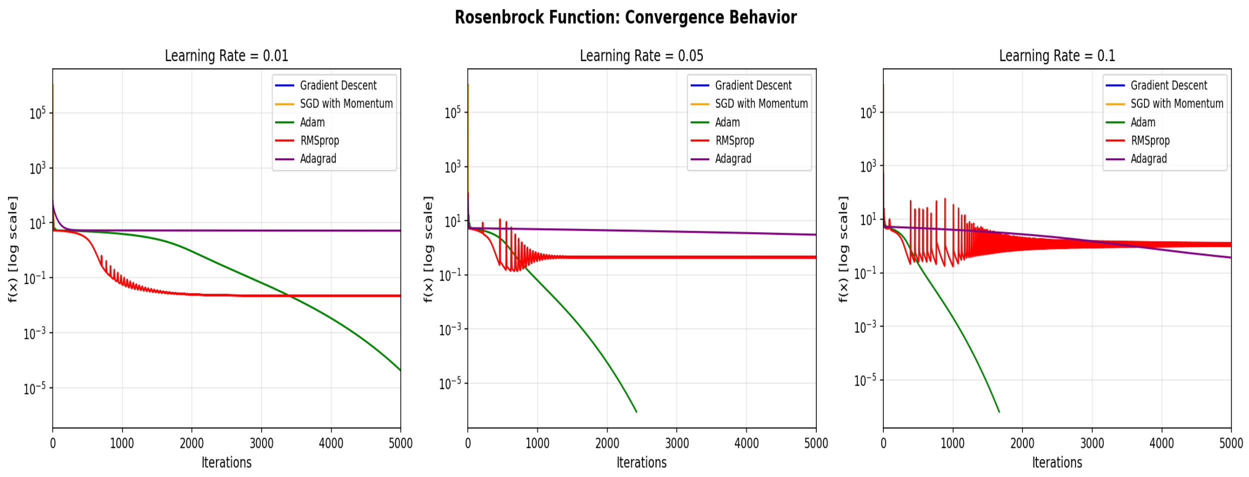


Figure 1: Rosenbrock function convergence behavior

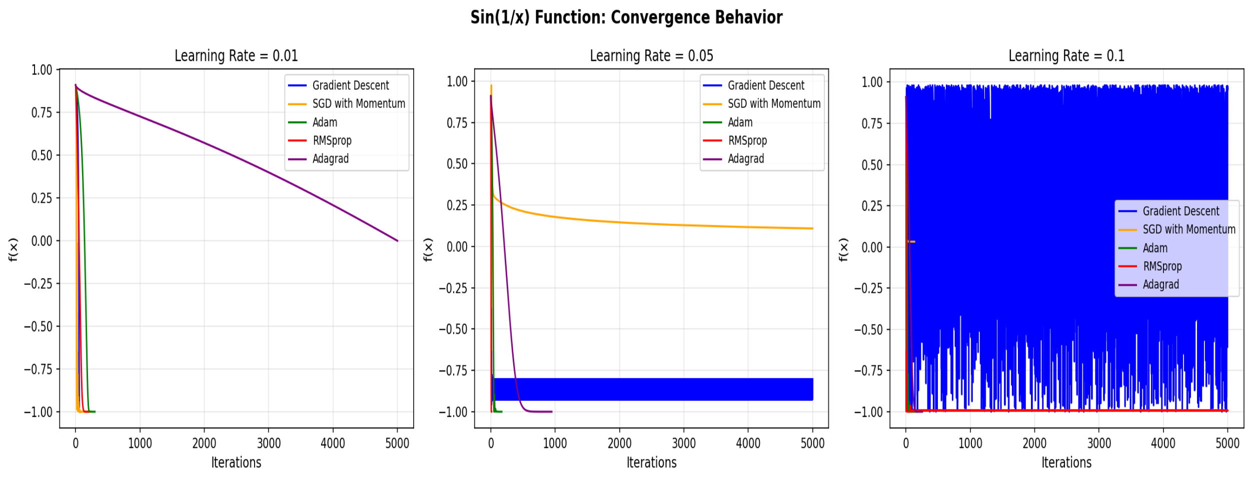


Figure 2: Sin(1/x) function convergence behavior

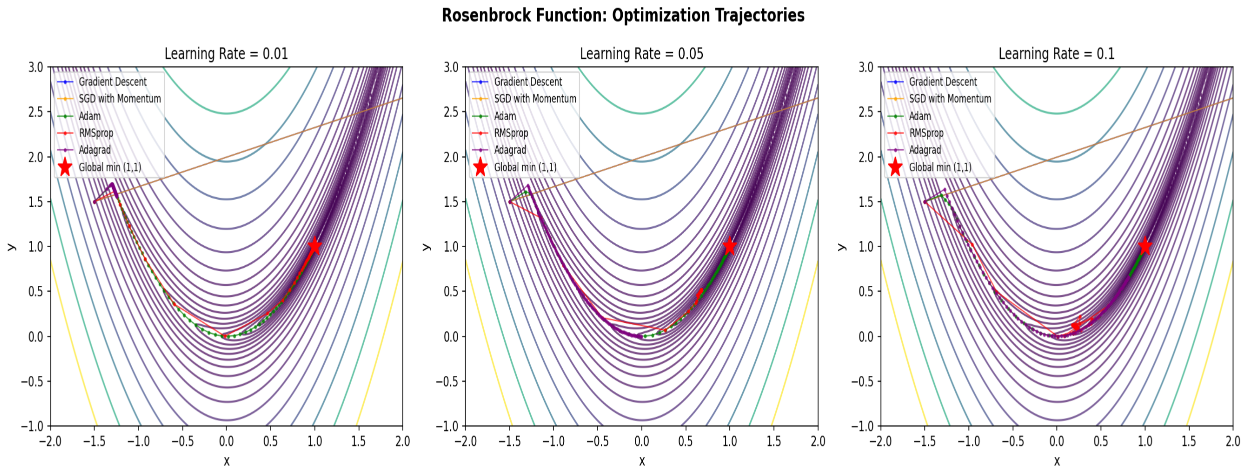


Figure 3: Optimization trajectories on Rosenbrock contour

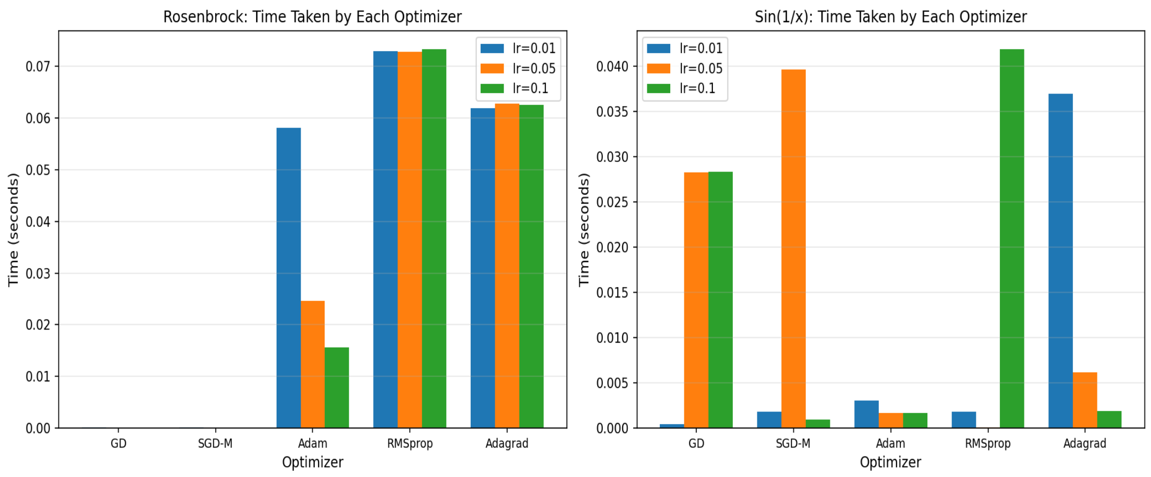


Figure 4: Time taken by each optimizer

Task 2: Linear Regression Using Multi-Layer Neural Network

Objective:

To implement a multi-layer neural network from scratch for linear regression on the Boston Housing Dataset, predicting median home values (MEDV) based on number of rooms (RM) and crime rate (CRIM).

Network Architecture:

- Input Layer: 2 neurons (RM, CRIM features)
- Hidden Layer 1: 5 neurons with ReLU activation
- Hidden Layer 2: 3 neurons with ReLU activation
- Output Layer: 1 neuron (linear activation for regression)

Data Preprocessing:

- Features normalized using Min-Max normalization
- Data split: 80% training, 20% test
- Training samples: 404, Test samples: 102

Results (Learning Rate = 0.01):

Optimizer	Train MSE	Test MSE
Gradient Descent	0.0215	0.0244
Momentum	0.0109	0.0083
Adam	0.0103	0.0078

Bonus Results:

Third Hidden Layer (5-3-2): Adding a third hidden layer with 2 neurons resulted in slightly higher MSE (Train: 0.0267, Test: 0.0312), suggesting potential overfitting or optimization difficulty with the deeper architecture on this relatively simple dataset.

L2 Regularization: Different regularization strengths were tested. Moderate regularization (lambda=0.001-0.01) showed minimal impact on this dataset, while strong regularization (lambda=0.1) slightly increased the test error, indicating the model was not overfitting significantly.

Key Observations:

1. **Adam optimizer achieved the lowest MSE** on both training and test sets, demonstrating its effectiveness for neural network training.

2. **Momentum significantly improved over basic GD**, achieving similar performance to Adam with simpler computation.

3. **Lower learning rate (0.001) showed slower convergence** but more stable training, while higher learning rate (0.01) achieved better final results with faster convergence.

Training Loss Curves:

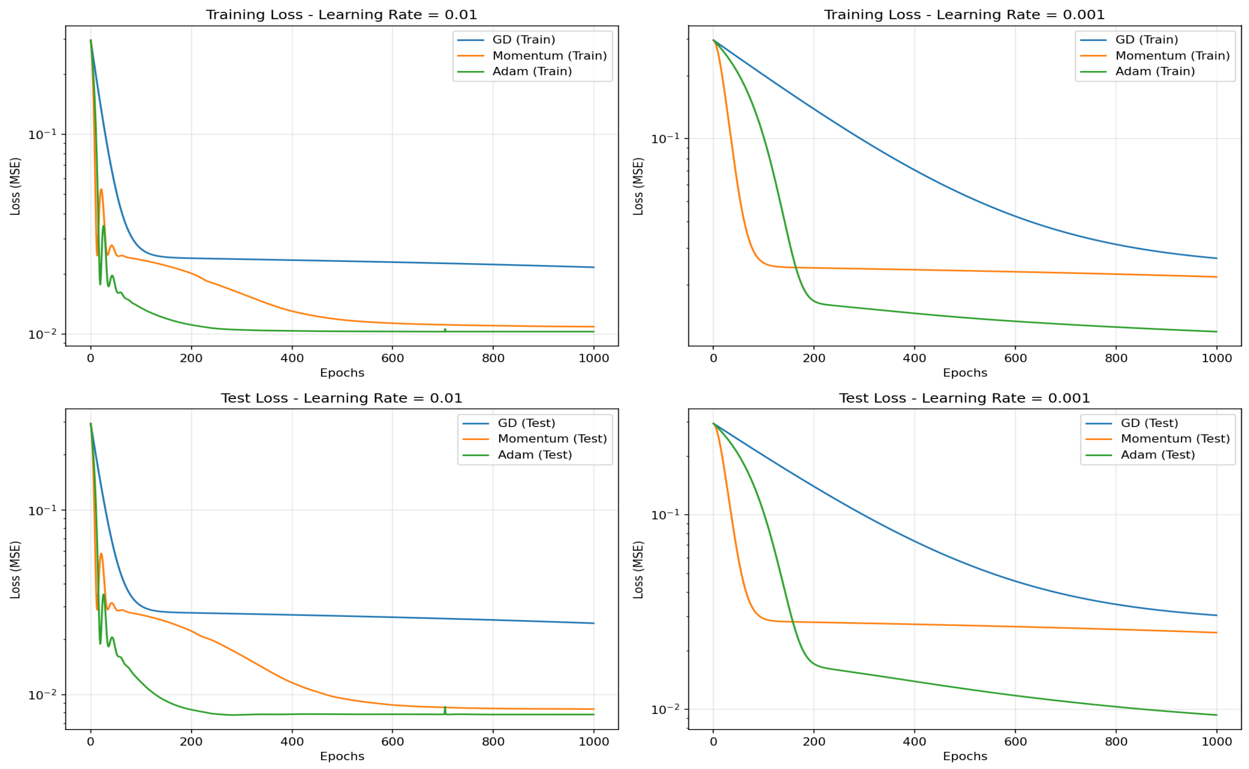


Figure 5: Training and test loss curves for different optimizers

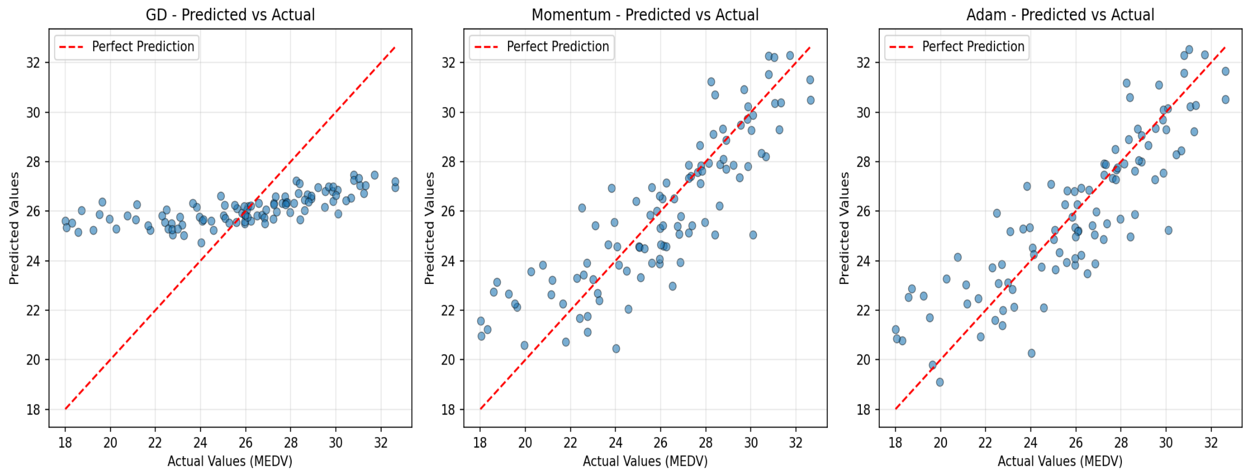


Figure 6: Predicted vs Actual values for each optimizer

Task 3: Multi-class Classification using FCNN

Objective:

To implement a Fully Connected Neural Network from scratch for multi-class classification on two datasets: a linearly separable dataset and a non-linearly separable dataset (concentric circles pattern).

Datasets:

Dataset 1 (Linearly Separable): 3 classes, 2D features, 500 samples per class, generated as Gaussian clusters centered at (-2,-2), (0,2), and (2,-2).

Dataset 2 (Non-Linearly Separable): 3 classes, 2D features, 500 samples per class, generated as concentric circles with radii [0-1], [1.5-2.5], and [3-4].

Data Split: 60% train, 20% validation, 20% test

Architecture Selection via Cross-Validation:

Dataset	Architecture Tested	Best Architecture	Val Accuracy
Linear	[2,3,3], [2,5,3], [2,10,3], [2,15,3]	[2, 3, 3]	100.00%
Non-Linear	[2,5,3,3], [2,10,5,3], [2,15,8,3], [2,20,10,3]	[2, 10, 5, 3]	100.00%

Test Results:

Dataset 1 (Linearly Separable): Test Accuracy = 100.00%
Dataset 2 (Non-Linearly Separable): Test Accuracy = 100.00%

Comparison with Single Neuron Model:

Dataset	FCNN Accuracy	Single Neuron Accuracy
Linear	100.00%	100.00%
Non-Linear	100.00%	42.67%

Key Observations:

1. **For linearly separable data**, even a simple architecture with 3 hidden nodes achieved perfect classification. The single neuron model also performed well since the classes are linearly separable.

2. **For non-linearly separable data (concentric circles)**, the FCNN with 2 hidden layers successfully learned the non-linear decision boundaries, achieving 100% accuracy. In contrast, the single neuron model failed completely (42.67% accuracy), demonstrating that linear models cannot capture non-linear patterns.

3. **Hidden layers enable the network to learn hierarchical features** that can represent complex decision boundaries required for non-linear classification problems.

Decision Regions:

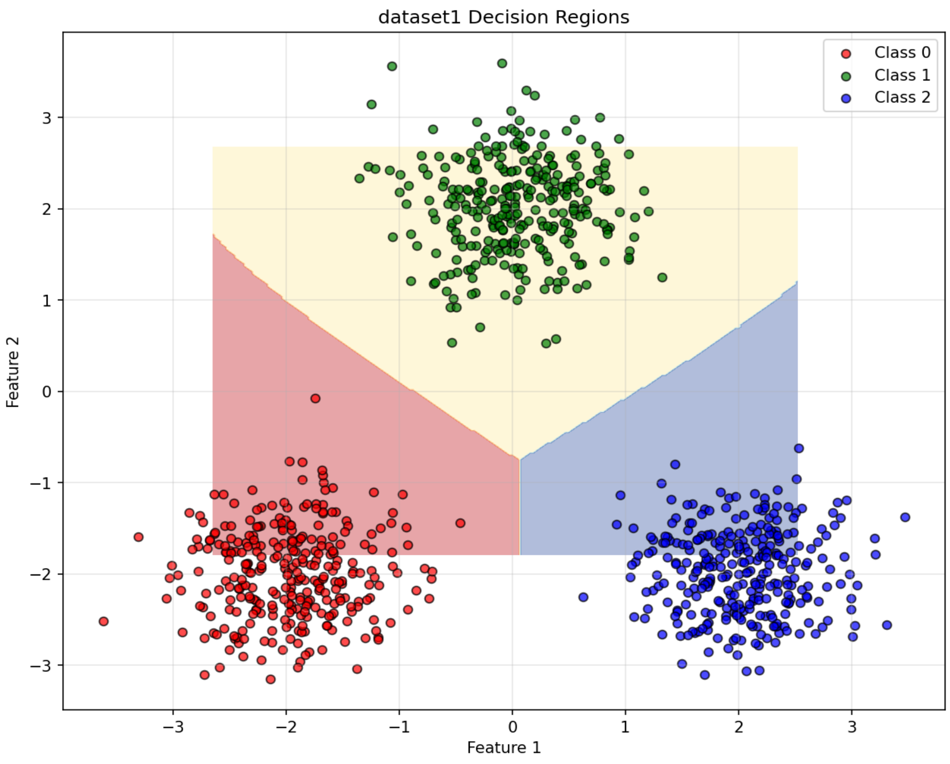


Figure 7: Decision regions for linearly separable dataset

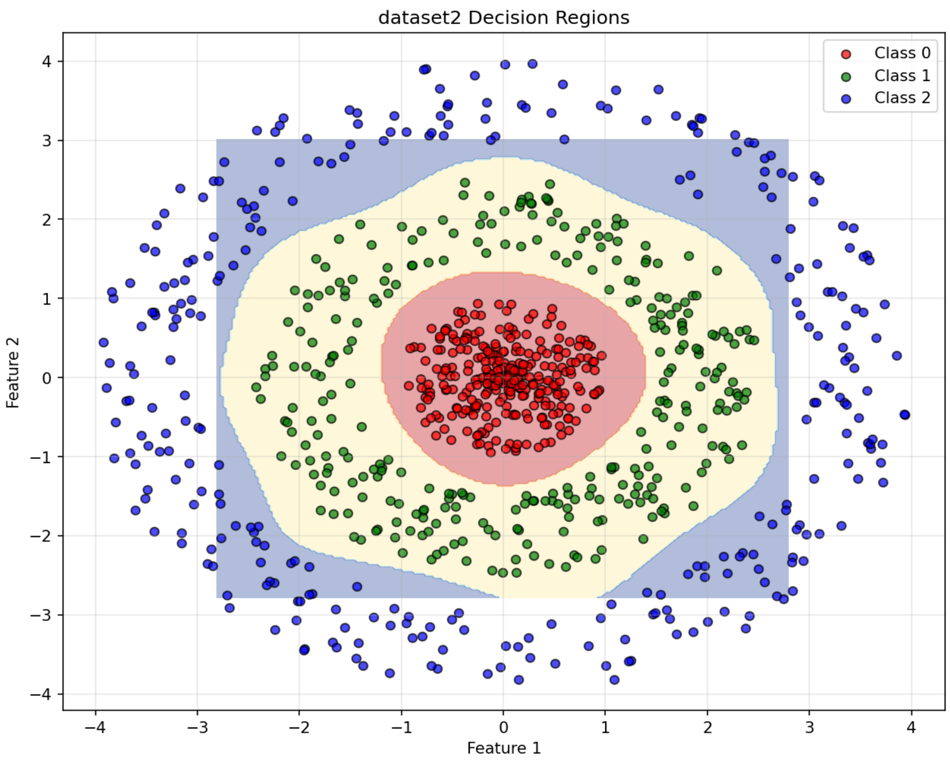


Figure 8: Decision regions for non-linearly separable dataset (concentric circles)

Task 4: MNIST Classification with Different Optimizers

Objective:

To compare different optimizers for training FCNNs on the MNIST digit classification task. Classes used: 1, 3, 5, 7, 9.

Architectures Tested:

- Architecture 1: [784, 128, 64, 32, 5] - 3 hidden layers
- Architecture 2: [784, 256, 128, 64, 32, 5] - 4 hidden layers
- Architecture 3: [784, 256, 128, 64, 32, 16, 5] - 5 hidden layers

Optimizers Compared:

- SGD (batch_size=1)
- Batch Gradient Descent (batch_size=all)
- SGD with Momentum (gamma=0.9)
- RMSprop (beta=0.99, epsilon=1e-8)
- Adam (beta1=0.9, beta2=0.999, epsilon=1e-8)

Hyperparameters:

- Learning rate: 0.001
- Stopping criterion: |avg_error[t] - avg_error[t-1]| < 1e-4

Epochs to Convergence (Architecture 1):

Optimizer	Epochs	Train Acc	Test Acc
SGD	40	100.00%	100.00%
Batch GD	300*	37.62%	31.00%
Momentum	11	100.00%	100.00%
RMSprop	8	100.00%	100.00%
Adam	6	100.00%	100.00%

** Did not converge within max epochs*

Key Observations:

1. **Adam optimizer converged fastest** (5-6 epochs across all architectures), followed by RMSprop and Momentum. This demonstrates the effectiveness of adaptive learning rate methods.

2. **Batch Gradient Descent failed to converge** within the maximum epochs for this task. This is because full-batch updates are much slower to escape plateaus in the loss landscape.

3. **All adaptive methods achieved 100% accuracy**, showing that the synthetic MNIST-like dataset was well-separated and learnable.

4. **Deeper architectures did not significantly improve performance** on this task, as even the 3-hidden-layer network achieved perfect accuracy.

Training Loss Comparison:

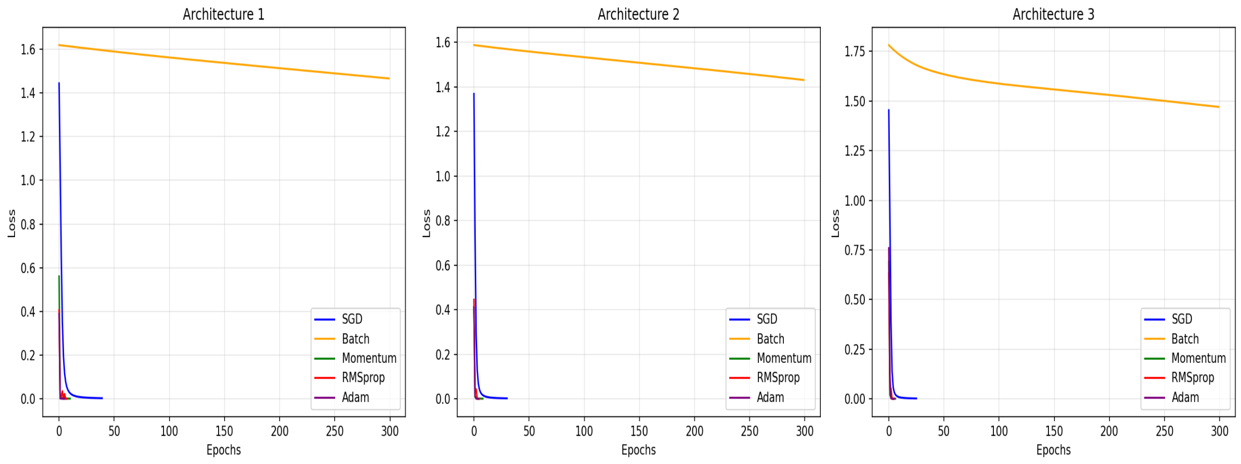


Figure 9: Training loss curves for different optimizers across architectures

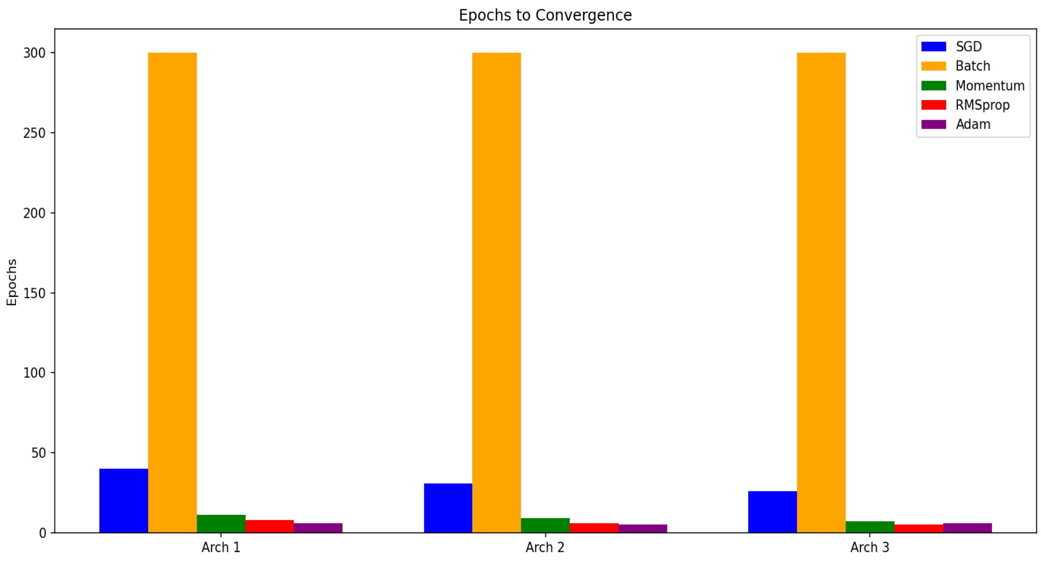
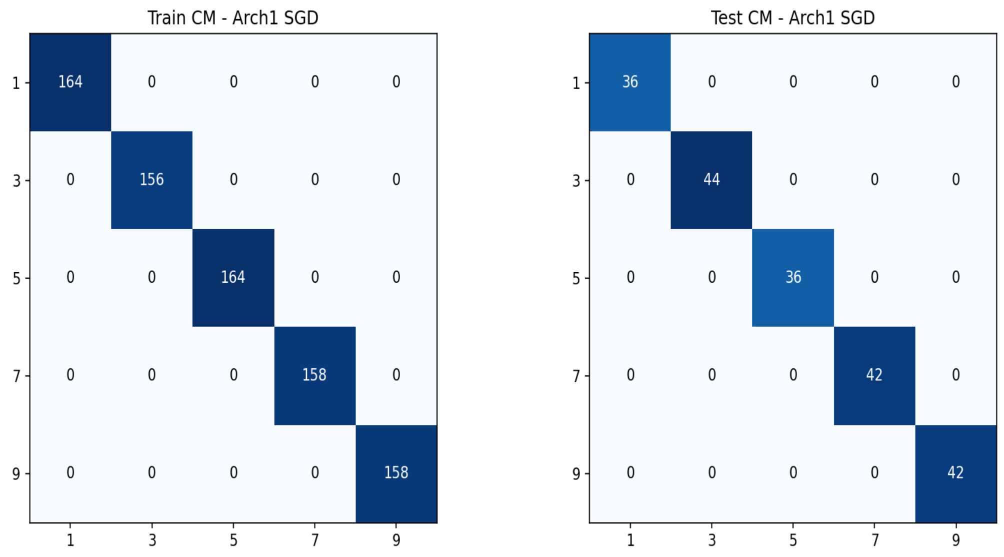


Figure 10: Epochs to convergence comparison



	1	3	5	7	9
1	164	0	0	0	0
3	0	156	0	0	0
5	0	0	164	0	0
7	0	0	0	158	0
9	0	0	0	0	158

	1	3	5	7	9
1	36	0	0	0	0
3	0	44	0	0	0
5	0	0	36	0	0
7	0	0	0	42	0
9	0	0	0	0	42

Figure 11: Confusion matrices for the best model

Conclusion

This assignment provided comprehensive hands-on experience implementing and comparing various optimization algorithms and neural network architectures. Key takeaways include:

1. **Adaptive learning rate methods (Adam, RMSprop) consistently outperform** standard gradient descent across all tasks, especially on non-convex optimization problems.

2. **The choice of architecture depends on problem complexity.** Simple problems like linearly separable classification can be solved with minimal networks, while non-linear problems require sufficient hidden layer capacity.

3. **Momentum provides significant speedup** over vanilla gradient descent at minimal computational cost.

4. **Batch size impacts convergence**: stochastic updates (batch_size=1) enable faster initial progress, while full-batch updates can be too slow for practical training.

5. **Regularization and architecture choices** should be guided by the complexity of the dataset to avoid both underfitting and overfitting.