Comprehensive Report on Optimization Algorithms and Neural Network Architectures

AI Collaborator

January 2026

Abstract

This report details the implementation and comparative analysis of various optimization algorithms applied to mathematical functions, regression tasks, and classification problems. We explore manual implementations of Gradient Descent, Momentum, RMSprop, and Adam, and evaluate their performance on synthetic datasets and the MNIST digit database.

1 Task 1: Optimization of Mathematical Functions

The objective of this task was to compare the convergence properties of five optimizers: Gradient Descent (GD), SGD with Momentum, Adagrad, RMSprop, and Adam.

1.1 Objective Functions

Two distinct functions were tested:

1. **Rosenbrock Function**: A non-convex function where the minimum sits in a narrow, parabolic valley.

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

2. **Sin-Inverse Function**: A highly oscillatory function near the origin.

$$f(x) = \sin\left(\frac{1}{x}\right)$$

1.2 Observations

The Rosenbrock function proved challenging for standard GD due to the "banana-shaped" valley. Adaptive methods like **Adam** and **RMSprop** reached the global minimum $(1, 1)$ significantly faster by adjusting the learning rate for each parameter.

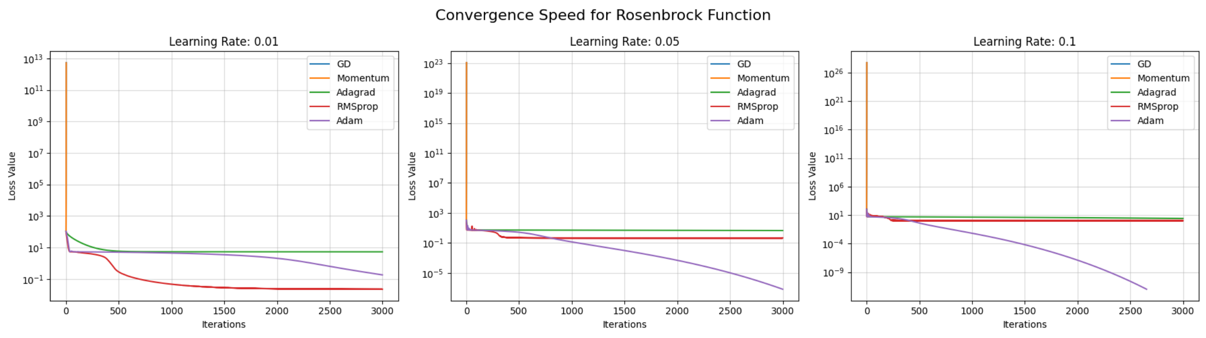

Figure 1: Loss convergence across different learning rates for the Rosenbrock Function.

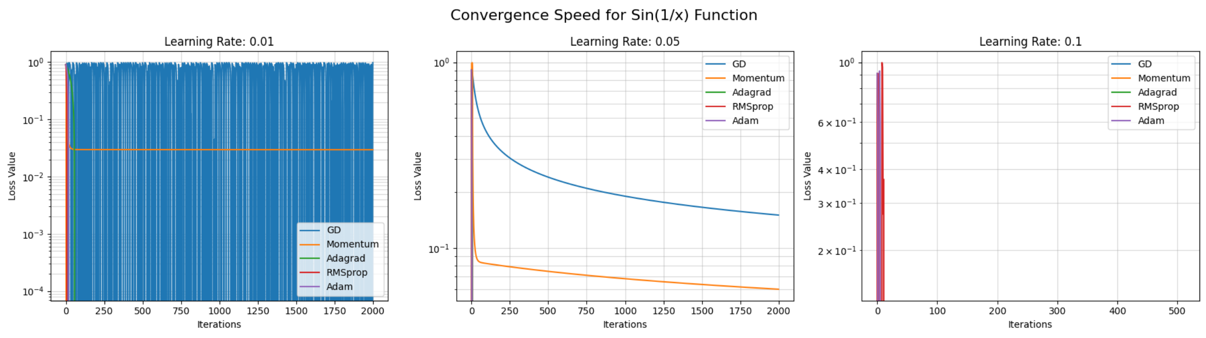

Figure 2: Loss convergence for the Sin(1/x) function showing oscillation handling.

—

2 Task 2: Neural Network for Regression (Boston Housing)

A three-layer Feed-Forward Neural Network was implemented from scratch using NumPy to predict housing prices.

2.1 Architecture and Implementation

- **Input Layer**: 13 features (normalized).
- **Hidden Layers**: 5 and 3 neurons with ReLU activation.
- **Output**: 1 neuron (Linear activation).
- **Loss Function**: Mean Squared Error (MSE).

2.2 Optimization Comparison

We compared standard GD, Momentum, and Adam. Adam provided the most stable descent and lowest final Test MSE, effectively navigating the loss landscape of the Boston Housing dataset.

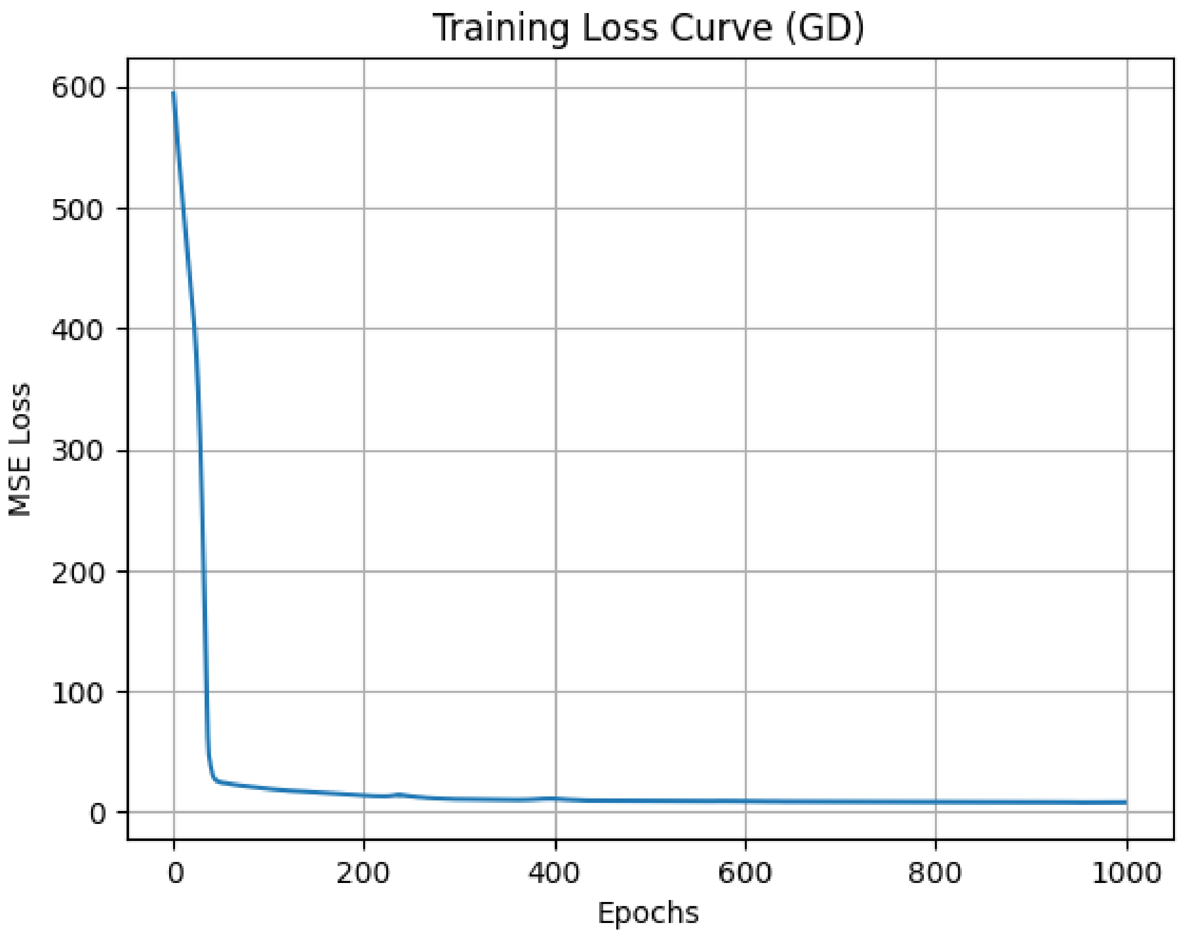

Figure 3: Training loss over 1000 epochs using Gradient Descent.

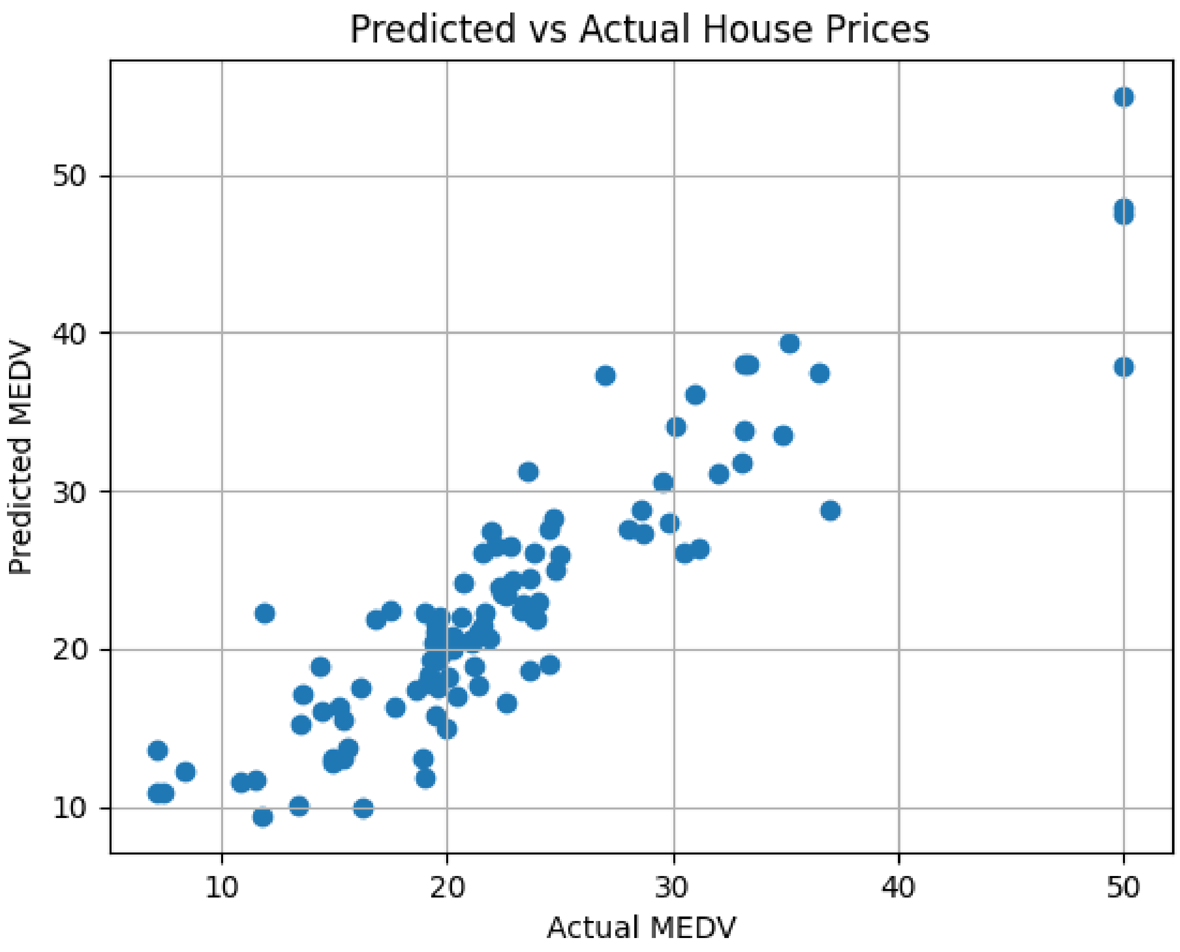

Figure 4: Regression performance: Predicted vs. Actual MEDV values.

—

3 Task 3: Multi-class Classification

We evaluated a manual Neural Network (Tanh hidden units, Softmax output) on two synthetic datasets.

3.1 Linear vs. Non-Linear Datasets

1. **Linear**: Three Gaussian clusters. The network easily found linear separators.
2. **Non-Linear (Spiral)**: A complex dataset where classes wrap around each other. This required a higher number of hidden units (20) and more epochs to capture the curvature.

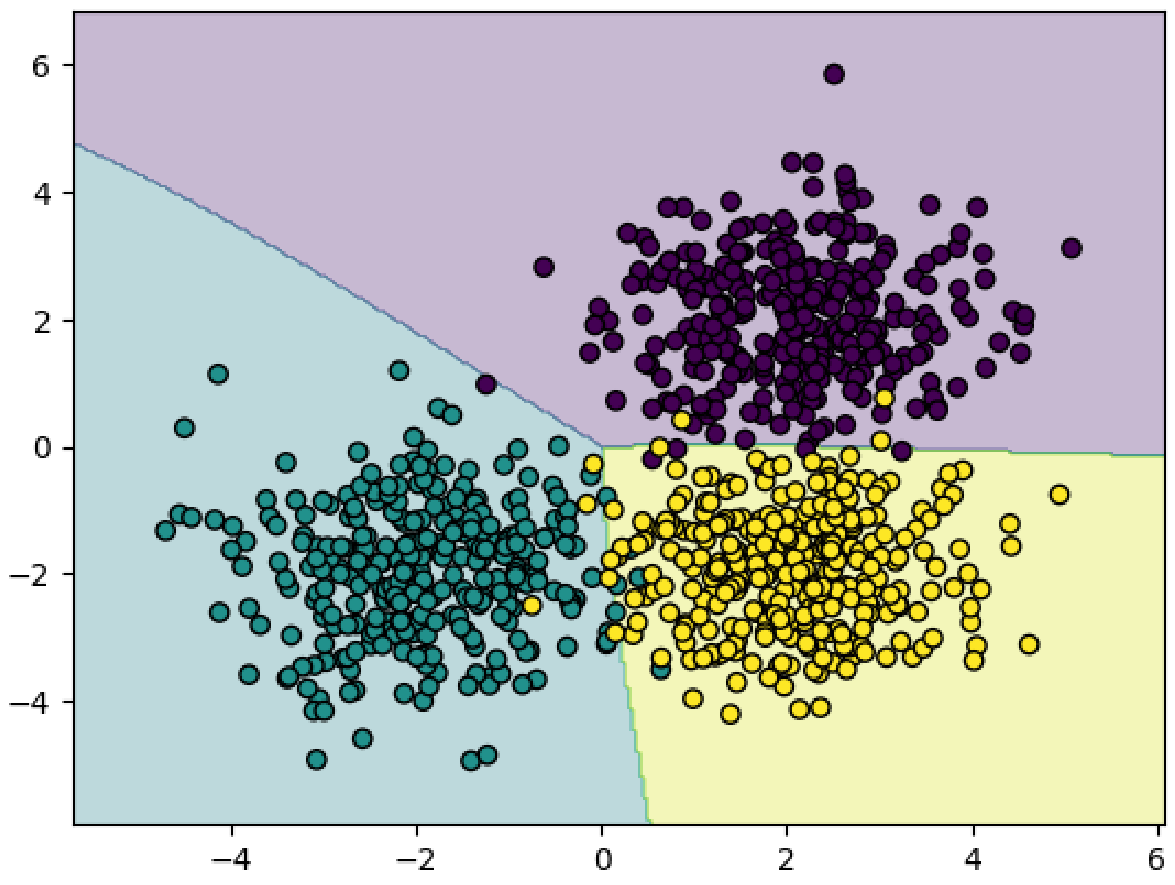

Figure 5: Decision boundaries for the linearly separable 3-class dataset.

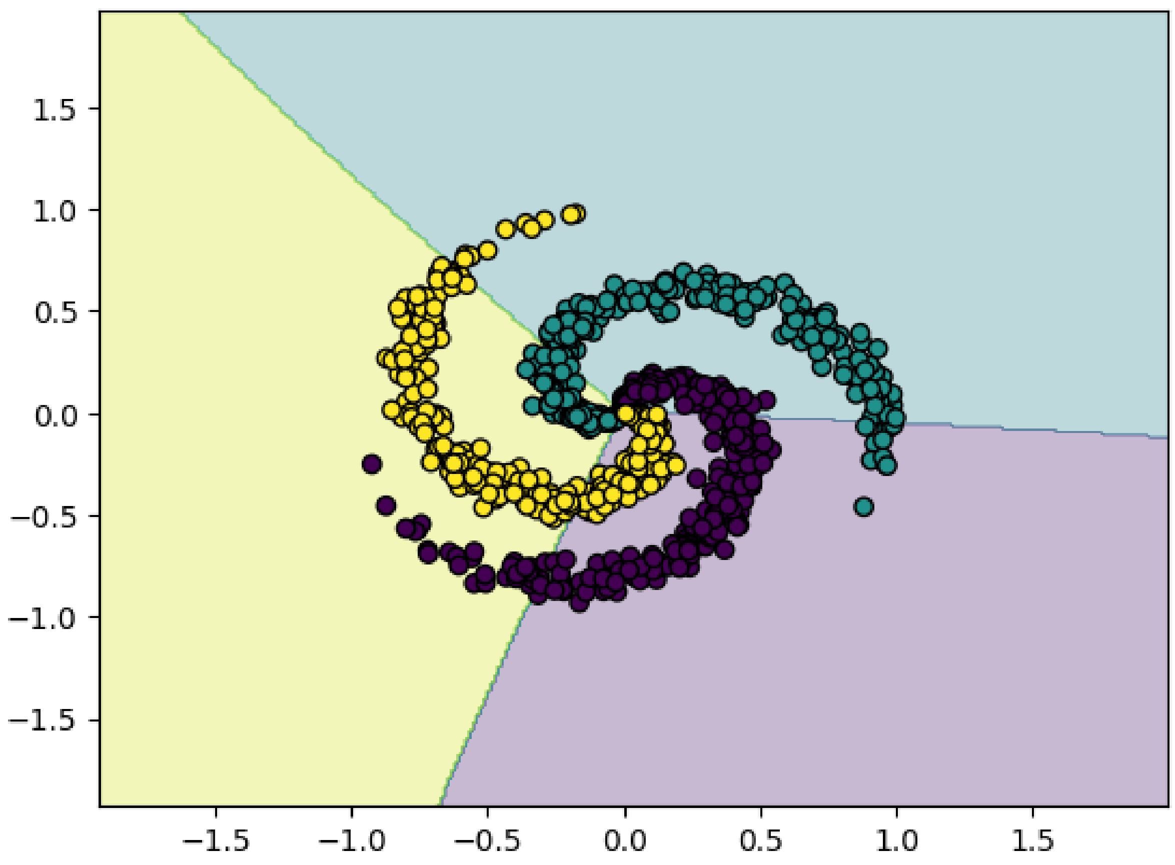

Figure 6: Non-linear decision boundaries for the spiral dataset.

—

4 Task 4: MNIST Digit Classification with PyTorch

The final task involved classifying digits 0-4 from the MNIST dataset using a Deep Fully Connected Neural Network (FCNN) implemented in PyTorch.

4.1 Model Configuration

- **Architecture**: $[784 \to 256 \to 128 \to 64 \to 5]$.
- **Optimizers**: Comparison of SGD, Batch GD, Momentum, NAG, RMSProp, and Adam.
- **Initialization**: Xavier Uniform initialization was used to prevent vanishing/exploding gradients.

4.2 Results

Adaptive optimizers (Adam, RMSProp) converged in fewer epochs compared to vanilla SGD. Nesterov Accelerated Gradient (NAG) showed superior performance over standard

Momentum.

Figure 7: Training loss comparison for various PyTorch optimizers.

5 Conclusion

Through these tasks, it is evident that while Gradient Descent is the foundational optimization algorithm, adaptive methods like **Adam** are generally superior for both deep learning and complex mathematical optimization due to their ability to handle sparse gradients and non-convex surfaces.