Artificial Intelligence Assignment 2

Task 1: Optimizer Performance on Non-Convex Functions

Submitted By:

Sparsh Ranjan	(2301MC50)
Ishika Khanagwal	(2301MC09)
Tarushi Singh	(2301MC39)
Kavya Relhan	(2301MC55)

Date: February 5, 2026

1 Task Objective

The primary objective of this task is to implement and analyze the performance of various optimization algorithms on non-convex functions from scratch using Python. We aim to:

- Optimize the **Rosenbrock function** (multi-variable) and the **Sine-Reciprocal function** (single-variable).
- Compare the convergence behavior of five optimizers: Gradient Descent (GD), SGD with Momentum, Adagrad, RMSprop, and Adam.
- Analyze the impact of different learning rates ($\alpha = 0.01, 0.05, 0.1$) on convergence speed and stability.

2 Methodology

The solution was implemented in Python using the ‘numpy‘ library. The methodology involved:

2.1 Function Definitions

Two non-convex functions were defined with their corresponding analytical gradients:

1. **Rosenbrock Function:** $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$.
2. **Sine-Reciprocal Function:** $f(x) = \sin(1/x)$.
    - *Constraint Handling:* To handle the singularity at $x = 0$, a condition was added: if $|x| < 10^{-6}$, return 0.

2.2 Optimizer Implementation

Five optimizers were implemented. Each iteratively updates parameters until:

- **Convergence:** Gradient norm $< 10^{-6}$.
- **Max Iterations:** 5000 iterations.
- **Divergence Check:** Gradient norm $> 10^8$ (safety break for exploding gradients).

3 Theoretical Background

3.1 Gradient Descent (GD)

Updates parameters in the opposite direction of the gradient.

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t) \tag{1}$$

3.2 SGD with Momentum

Accumulates a velocity vector (v_t) to dampen oscillations.

$$v_t = \gamma v_{t-1} + \alpha \nabla J(\theta_t), \quad \theta_{t+1} = \theta_t - v_t \tag{2}$$

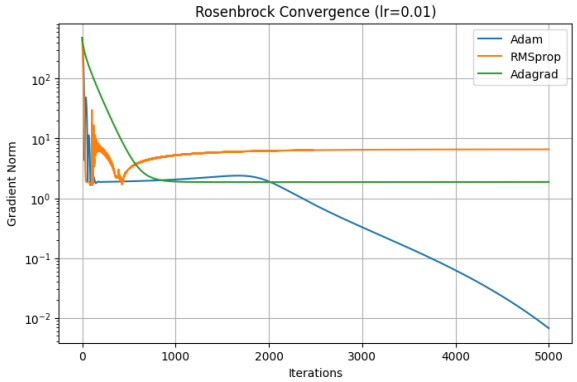
3.3 Adaptive Methods

- **Adagrad:** Scales gradients by the inverse square root of accumulated squared gradients.
- **RMSprop:** Uses an exponential moving average to fix Adagrad's diminishing learning rate.
- **Adam:** Combines Momentum and RMSprop for robust convergence.

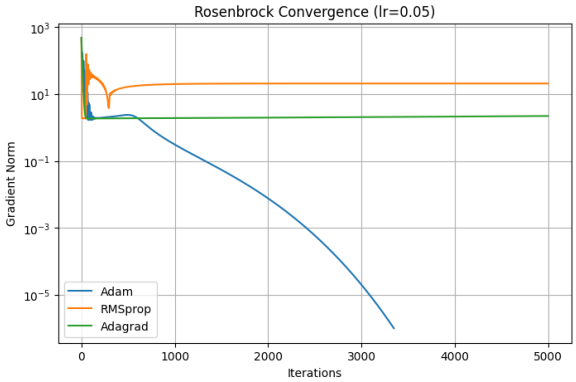
4 Results and Inferences

4.1 Rosenbrock Function Optimization

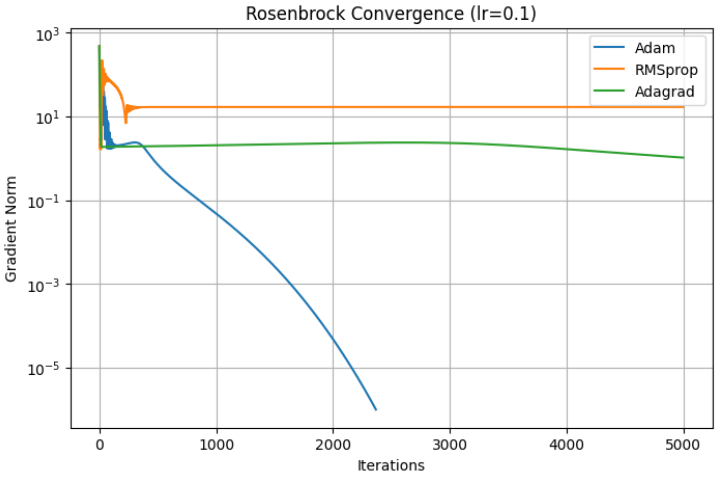
The following plots show the convergence behavior on the Rosenbrock valley.



(a) Learning Rate $\alpha = 0.01$



(b) Learning Rate $\alpha = 0.05$



(c) Learning Rate $\alpha = 0.1$

Figure 1: Convergence Analysis on Rosenbrock Function

Inferences on Rosenbrock:

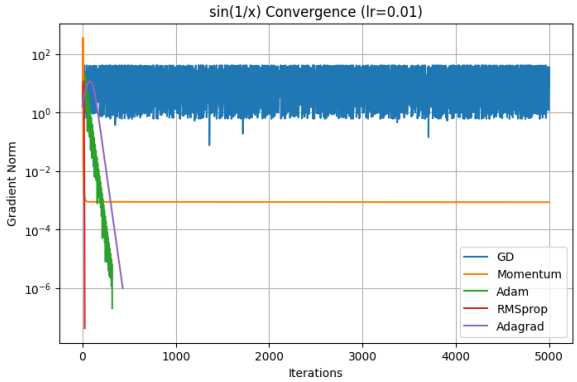
1. **Divergence of GD and Momentum:** In graphs (b) and (c) (high learning rates), GD and Momentum are absent. This is because the step size was too large for

the steep walls of the Rosenbrock valley, causing the gradients to explode and the algorithms to terminate immediately.

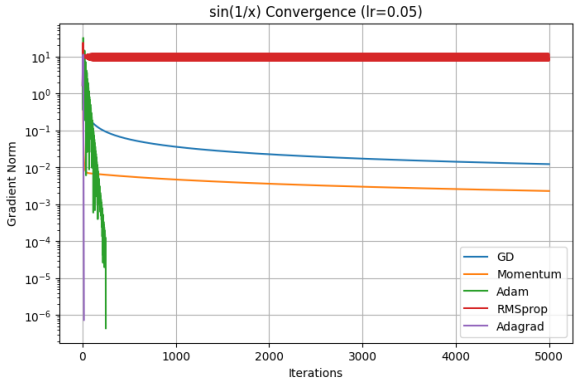
2. **Stability of Adam:** Adam and RMSprop appear in all plots, proving they are robust enough to handle high learning rates by adaptively scaling down the step size in steep regions.

4.2 Sine-Reciprocal Function Optimization

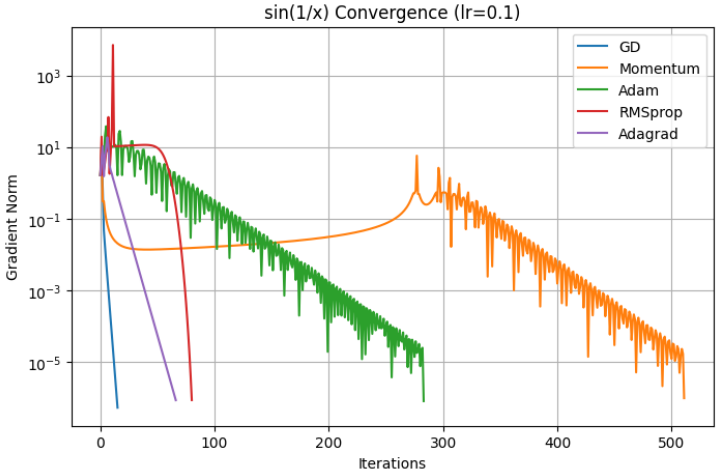
The following plots show the convergence behavior on $f(x) = \sin(1/x)$.



(a) Learning Rate $\alpha = 0.01$



(b) Learning Rate $\alpha = 0.05$



(c) Learning Rate $\alpha = 0.1$

Figure 2: Convergence Analysis on Sine-Reciprocal Function

Inferences on Sine Function:

1. **Local Minima:** All optimizers converged rapidly. However, due to the oscillating nature of $\sin(1/x)$, they likely settled in local minima rather than the global minimum.

2. **Oscillations:** Higher learning rates (Graph 3) show more oscillation before convergence, particularly for Momentum-based methods.

5 Conclusion

The experiments confirm that while Vanilla Gradient Descent is simple, it is fragile on complex landscapes like Rosenbrock. Adaptive methods, particularly **Adam**, provide the best balance between speed and stability.