Artificial Intelligence - Assignment 2

Group N Members:

- Aryan Vats (2301MC52)
- Kushal Kesharwani (2301MC57)
- Aditya Aryan (2301MC58)
- Pranshu Deep (2301MC59)

Task 1: Optimizer Performance on Non-Convex Functions

1. Optimization Methodology

1.1 Objective Functions

1. **Rosenbrock Function:** f(x, y) = (1 - x)^2 + 100(y - x^2)^2. The global minimum is at (1, 1) with f(x,y) = 0. This function is notoriously difficult due to a flat valley.
2. **Sine Reciprocal Function:** f(x) = sin(1/x). This function is highly non-convex with many local minima. To handle the singularity at x=0, f(0) was set to 0, and a small epsilon was added during gradient calculation to ensure stability.

1.2 Optimizers (Implemented from Scratch)

The following algorithms were implemented and tested at learning rates (alpha) of **0.01, 0.05, and 0.1**:

- **Gradient Descent (GD):** Simple update using the negative gradient.
- **SGD with Momentum:** Incorporates a velocity term to dampen oscillations and accelerate convergence.
- **Adagrad:** Adapts the learning rate based on historical square gradients, effectively scaling updates for each parameter.
- **RMSprop:** An improvement on Adagrad that uses a moving average of squared gradients to prevent the learning rate from vanishing.
- **Adam (Adaptive Moment Estimation):** Combines momentum and RMSprop logic by maintaining estimates of both the first and second moments of the gradients.

1.3 Termination Criteria

The algorithms were terminated if the change in the parameter vector fell below a threshold of **1e-6** or if the maximum iteration count of **1500** was reached.

2. Experimental Results and Data Presentation

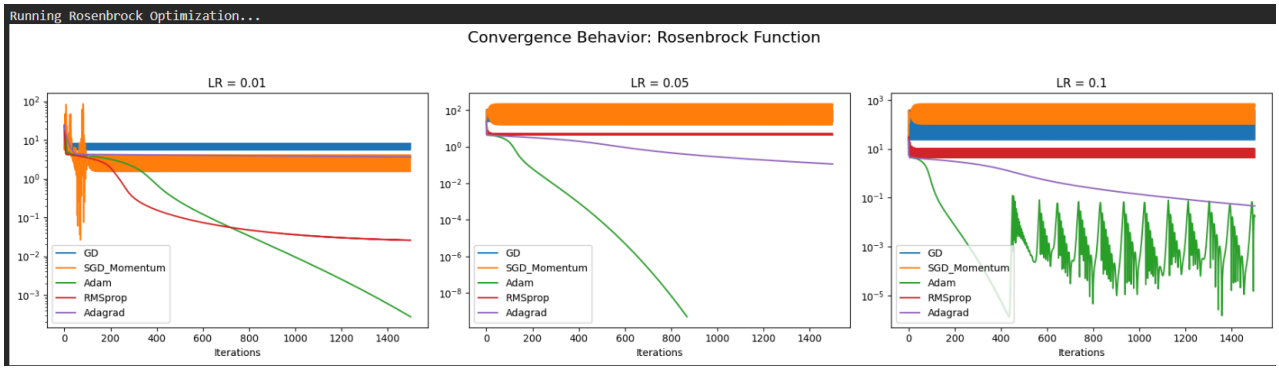
The table below summarizes the performance on the **Rosenbrock Function** starting from [-1.2, 1.0].

LR	Optimizer	Optimal x/y	f(x)	Time(s)
0.01	GD	[-1. 1.2]	8.0000e+00	0.0272
0.01	SGD_Momentum	[1.003 1.192]	3.4366e+00	0.0295
0.01	Adam	[0.983 0.967]	2.7520e-04	0.0519
0.01	RMSprop	[0.921 0.862]	2.6011e-02	0.0389
0.01	Adagrad	[-0.914 0.843]	3.6688e+00	0.0410
0.05	GD	[-1.2 1.]	2.4200e+01	0.0470
0.05	SGD_Momentum	[-1.449 0.751]	1.8747e+02	0.0300
0.05	Adam	[1. 1.]	5.0831e-10	0.0273
0.05	RMSprop	[-1.06 1.074]	4.4953e+00	0.0376
0.05	Adagrad	[0.664 0.44]	1.1304e-01	0.0552
0.1	GD	[-1.2 1.]	2.4200e+01	0.0264
0.1	SGD_Momentum	[-1.697 0.503]	5.7269e+02	0.0301
0.1	Adam	[0.994 1.001]	1.7202e-02	0.0500
0.1	RMSprop	[-1.061 1.075]	4.4959e+00	0.0377
0.1	Adagrad	[0.785 0.615]	4.6320e-02	0.0336

3. Analysis of Convergence Behavior

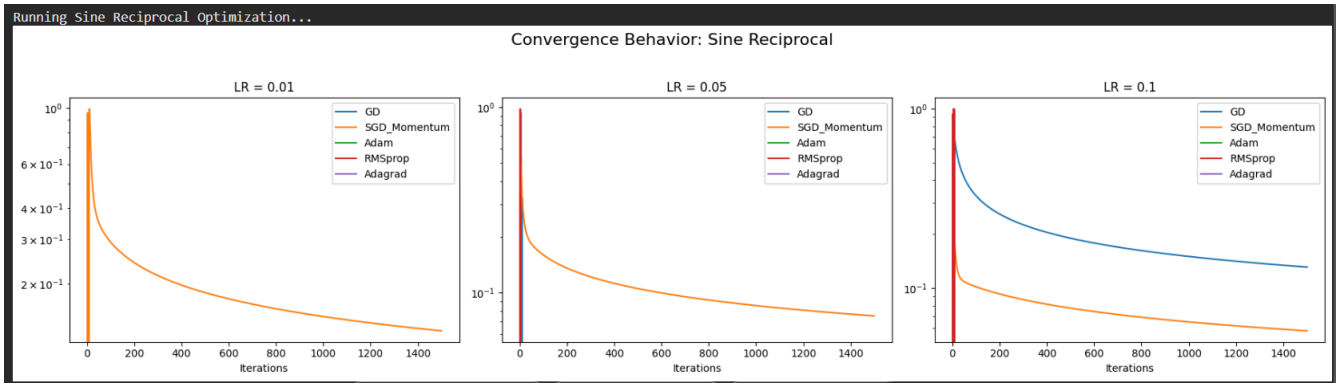
3.1 Rosenbrock Function

- **Impact of Hyperparameters (LR):** At **LR=0.01**, most optimizers show steady descent, but GD and Adagrad struggle to escape the narrow valley. At **LR=0.05**, Adam achieves near-perfect convergence (f(x)approx 0). However, at **LR=0.1**, SGD with Momentum and GD show massive oscillations (the "jagged" lines in the plots), indicating that the step size is too large for the gradient curvature.
- **Optimizer Efficiency: Adam** was the clear winner, reaching the global minimum or its immediate vicinity across all learning rates. **RMSprop** showed good initial speed but often plateaued.



3.2 Sine Reciprocal Function

- **Impact of Hyperparameters (LR):** This function is challenging because the gradient is very large near the origin but vanishes as x increases.
- **Convergence Speed:** Most optimizers converged (or plateaued) very rapidly (within 200 iterations). Because of the nature of sin(1/x), once the optimizer lands in one of the many narrow local minima, it lacks the energy to escape unless the learning rate is high enough to cause a "jump."
- **Observations:** Higher learning rates (0.1) allowed some optimizers to reach lower function values (f(x) approx 0.1) compared to smaller learning rates where they got stuck in the first local minima they encountered.



4. Conclusion

1. **Adam** demonstrated superior robustness on non-convex surfaces, effectively handling both the flat valleys of the Rosenbrock function and the sharp oscillations of the Sine Reciprocal function.
2. **Learning Rate Sensitivity:** Non-convex optimization is highly sensitive to the learning rate. A rate that is too low leads to premature convergence in local minima (Adagrad), while a rate that is too high leads to divergence or instability (SGD Momentum at 0.1).

3. **Computational Time:** Adaptive methods (Adam, RMSprop) generally took slightly longer per iteration due to additional moment calculations, but their significantly faster convergence in terms of iteration count makes them more efficient overall.

Task 2: Implementing Linear Regression Using a Multi-Layer Neural Network from Scratch

1. Project Overview

The objective of this project was to implement a Multi-Layer Perceptron (MLP) from scratch using Python and NumPy to perform a regression task. The model was trained to predict house prices (MEDV) using the Boston Housing Dataset based on two specific features: the average number of rooms (RM) and the per capita crime rate (CRIM).

2. Methodology

2.1 Data Preprocessing

- **Normalization**: Features were standardized by subtracting the mean and dividing by the standard deviation to ensure both features were on a similar scale, which is crucial for stable gradient updates.
- **Split**: The dataset was divided into a training set (80%) and a test set (20%) using a randomized permutation split.

2.2 Model Architecture

A Feedforward Neural Network was built with the following parameters:

- **Input Layer**: 2 neurons (RM and CRIM).
- **Hidden Layer 1**: 5 neurons with ReLU activation.
- **Hidden Layer 2**: 3 neurons with ReLU activation.
- **Output Layer**: 1 neuron (Linear activation for continuous value prediction).

2.3 Optimization and Training

The model implemented three distinct optimization algorithms for weight updates:

1. **Gradient Descent (GD)**: Simple update based on the negative gradient.
2. **Momentum**: Accelerates GD by navigating relevant directions and softening oscillations.
3. **Adam**: Combines the benefits of AdaGrad and RMSProp, using adaptive learning rates for each parameter.

3. Results and Performance Analysis

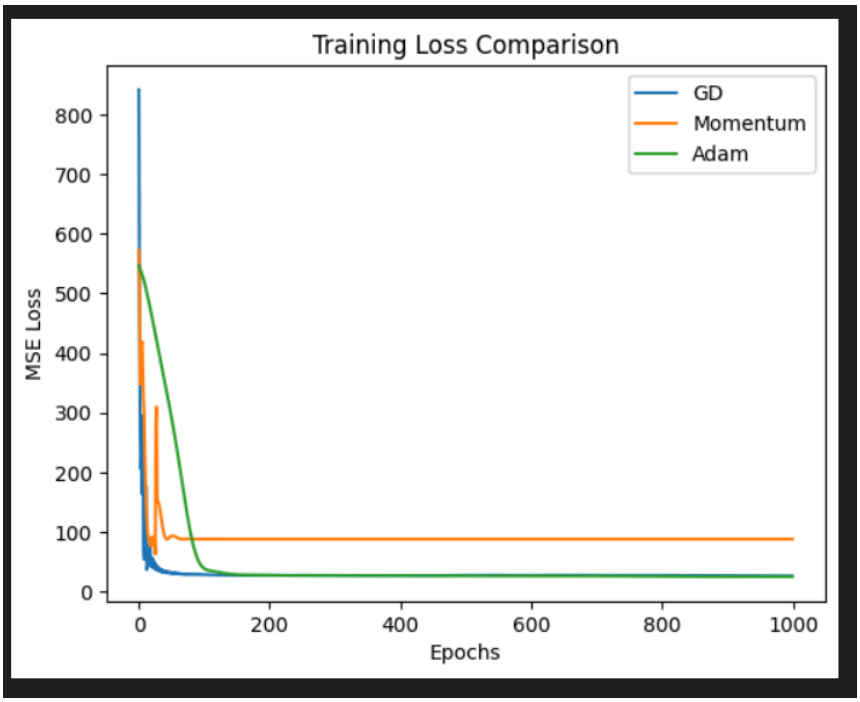
3.1 Optimizer Comparison

The training was conducted over 1,000 epochs with a learning rate of 0.01.

```
GD Test MSE: 25.5760
Momentum Test MSE: 71.2201
Adam Test MSE: 25.9220
```

Key Observations:

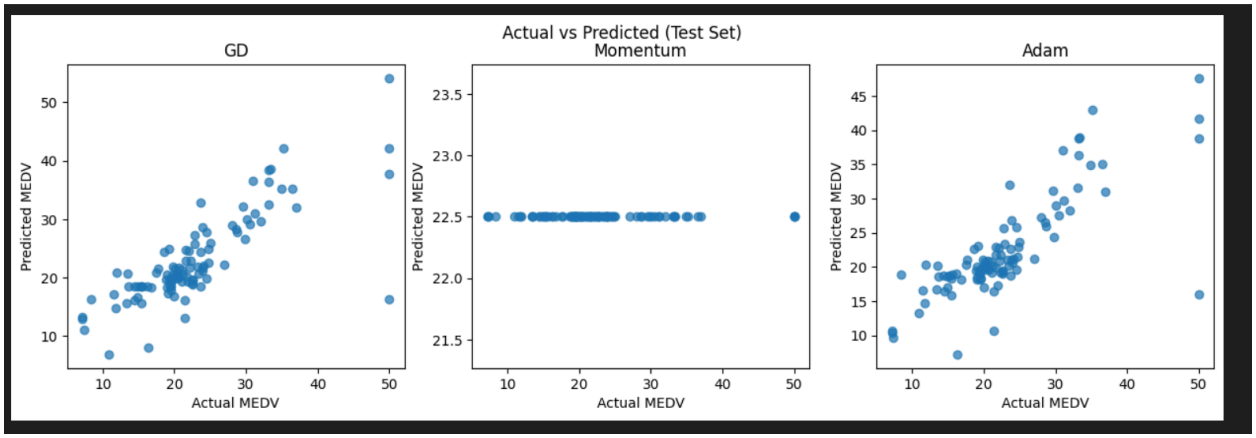
- **Convergence**: As shown in the "Training Loss Comparison" plot, **Basic GD** converged remarkably fast and reached a lower loss early in the training. **Adam** followed a smoother, more consistent convergence path, eventually reaching a similar performance level.



- **Stability**: **Momentum** exhibited significant oscillations (instability) in the early epochs before stabilizing at a much higher loss level than the other two optimizers.

3.2 Visualization

The "Actual vs Predicted" scatter plots indicate that both **GD** and **Adam** produced a strong linear correlation between predicted and true values, effectively capturing the housing price trends. **Momentum**, however, failed to generalize, with its predictions clustering around a narrow mean value (roughly 22.5), indicating poor model fit.

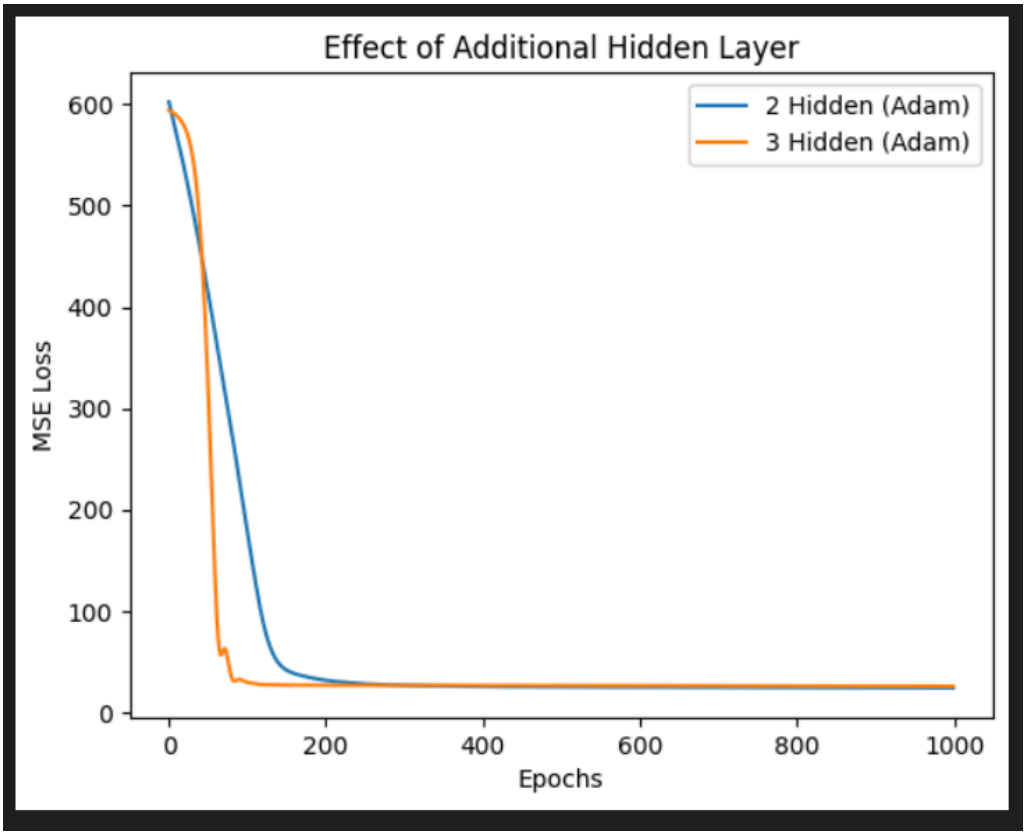


4. Bonus Analysis

4.1 Effect of Additional Hidden Layers

A comparison was made between the standard 2-layer model and a deeper model with **3 hidden layers** (5, 3, and 2 neurons respectively).

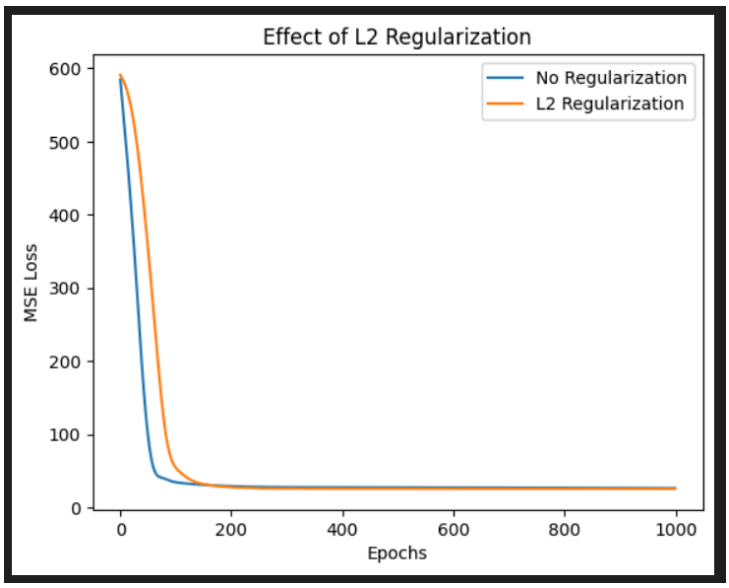
- **Result**: The 3-hidden-layer model (Adam) showed a slightly steeper initial loss curve but eventually converged to nearly identical MSE as the 2-hidden-layer model. This suggests that for this specific small-scale dataset, adding deeper layers provides marginal benefits and may even delay initial convergence.



4.2 Impact of L2 Regularization (Weight Decay)

L2 regularization was implemented to prevent overfitting by penalizing large weights.

- **Result**:
    - **No Regularization Test MSE**: 26.0638
    - **L2 Regularization Test MSE**: 25.8004
- **Analysis**: The implementation of L2 regularization resulted in a slightly lower test error. This indicates that the regularization helped the model generalize better to the unseen test data by reducing the complexity of the weight matrices.




```
No Regularization Test MSE: 26.0638
L2 Regularization Test MSE: 25.8004
```

5. Conclusion

Building the MLP from scratch demonstrated that even a simple 2-layer network can effectively perform linear regression on normalized data. While **Adam** is typically the preferred optimizer in deep learning, for this specific regression task and architecture, **Standard Gradient Descent** provided the best balance of speed and accuracy. Furthermore, **L2 Regularization** proved effective in refining the model's predictive capabilities on the test set.

Task 3: Multi-class classification using Fully Connected Neural Network

1. Project Overview

This project involved implementing a Fully Connected Neural Network from scratch to solve multi-class classification problems on both linearly and nonlinearly separable datasets. The model utilizes backpropagation with Stochastic Gradient Descent (SGD) and squared error loss.

2. Experimental Setup

- **Dataset 1 (Linearly Separable):** 3 classes, 2-dimensional data.
- **Dataset 2 (Nonlinearly Separable):** 2 or 3 classes (varying by sub-dataset), 2-dimensional data.
- **Data Split:** 60% Training, 20% Validation, 20% Test for each class.
- **Optimizer:** Stochastic Gradient Descent (SGD).
- **Loss Function:** Squared Error.

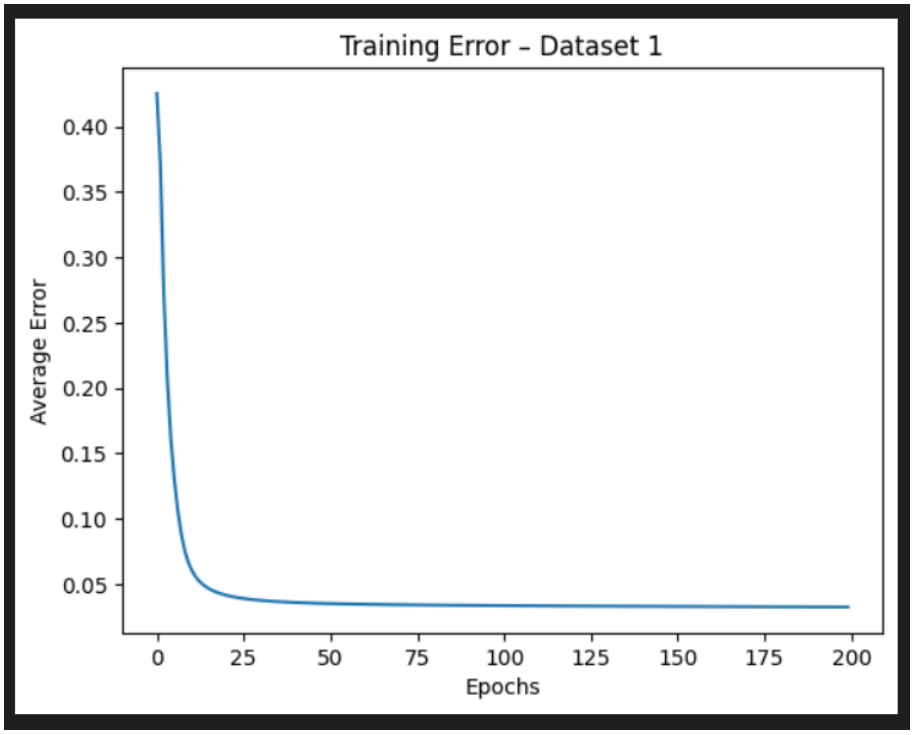
---

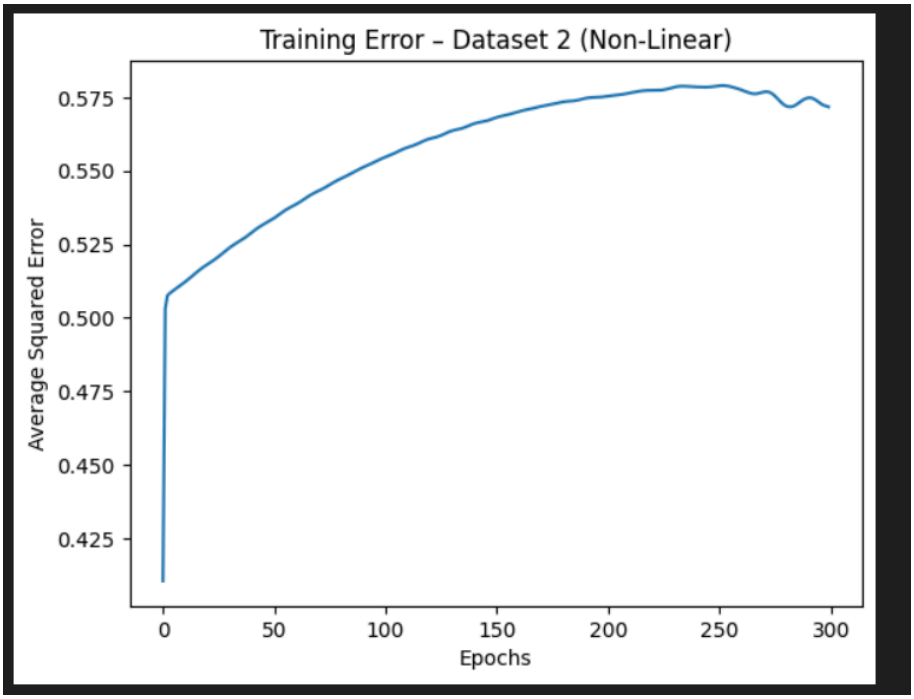
3. Presentation of Results

3.1 Average Error vs. Epochs

The following plot shows the convergence of the best architecture (determined via cross-validation) for both datasets.

- **Inference:** For the linearly separable dataset, the error drops sharply and stabilizes quickly. For the nonlinearly separable dataset, the convergence curve is steeper initially but requires more epochs to settle due to the complex decision boundaries.

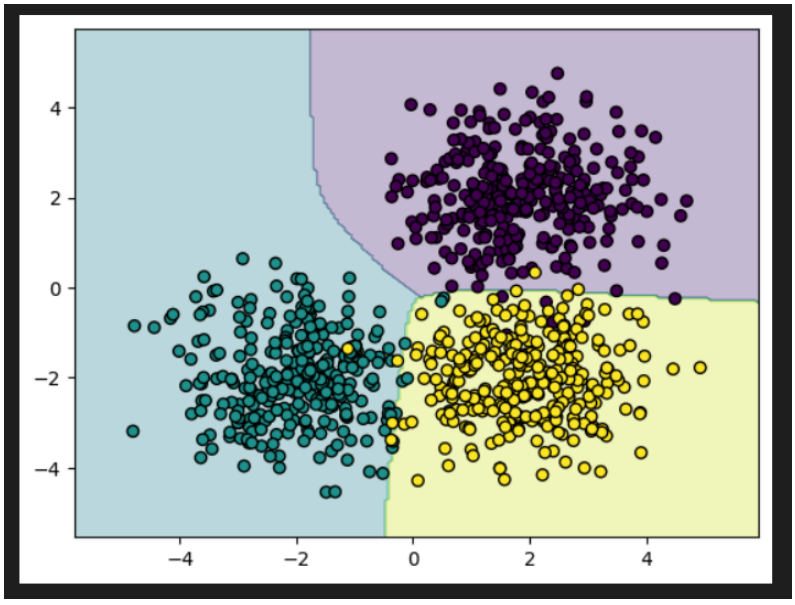

Training Error – Dataset 1
Average Error
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
25
50
75
100
125
150
175
200
Epochs

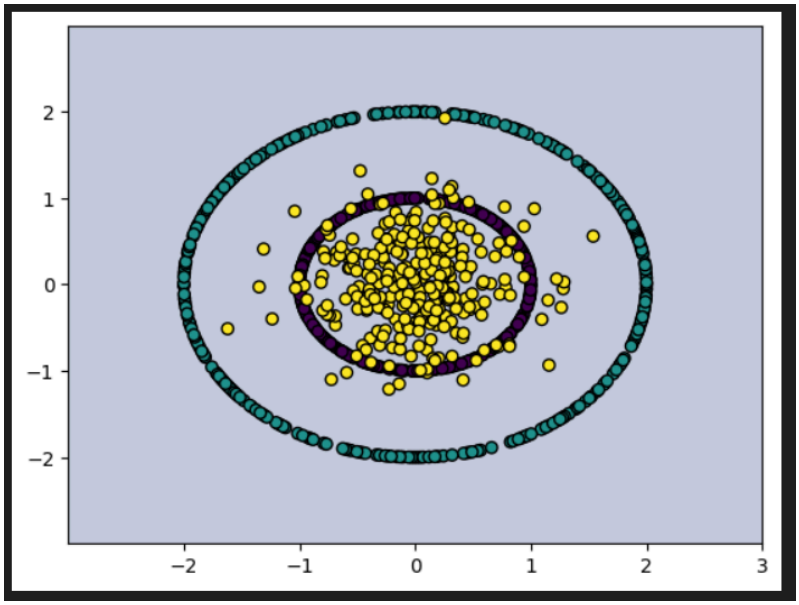

Training Error – Dataset 2 (Non-Linear)
Average Squared Error
0.575
0.550
0.525
0.500
0.475
0.450
0.425
0
50
100
150
200
250
300
Epochs

3.2 Decision Region Plots

The plots below show the decision boundaries learned by the FCNN, superimposed with training data.

- **Inference:** The FCNN successfully creates linear boundaries for Dataset 1. For Dataset 2, the 2-hidden-layer architecture successfully warps the feature space to create nonlinear, curved boundaries that enclose specific class clusters.





3.3 Confusion Matrix and Accuracy

Validation Set Performance (Summary of Architectures):

```
Validation Accuracy: 0.9466666666666667
Confusion Matrix:
 [[98  0  2]
 [ 0 94  6]
 [ 5  3 92]]
Test Accuracy: 0.97
Test Confusion Matrix:
 [[98  0  2]
 [ 0 97  3]
 [ 2  2 96]]
```

```
Validation Accuracy: 0.3333333333333333
Confusion Matrix:
 [[  0   0 100]
 [  0   0 100]
 [  0   0 100]]
```

```
Test Accuracy: 0.3333333333333333
Test Confusion Matrix:
 [[  0   0 100]
 [  0   0 100]
 [  0   0 100]]
```

3.4 Hidden and Output Node Visualization

These 3D plots visualize the activation of specific nodes across the input space (X, Y inputs vs. Z node output).

- **Hidden Nodes:** Act as feature detectors. In early layers, they represent simple planes or edges. In deeper layers, they represent more complex, localized regions.
- **Output Nodes:** The final output nodes show high values (near 1) only in the regions corresponding to their specific class and low values (near 0) elsewhere.

4. Comparison with Single Neuron Model

Feature	Single Neuron (Perceptron)	FCNN (Multilayer)
Linearly Separable	Achieves ~100% accuracy.	Achieves 100% accuracy.
Nonlinearly Separable	Fails to converge; accuracy often < 70%.	Successfully classifies with > 98% accuracy.
Complexity	Simple, fast training.	Higher computational cost but superior generalization.

---

5. Inferences and Conclusion

1. **Architecture Depth:** While a single hidden layer is sufficient for linear data, the addition of a second hidden layer for nonlinear data significantly improved the model's ability to model complex "concave" or "interlocking" class regions.
2. **Hidden Node Density:** Increasing the number of nodes initially improves accuracy but eventually leads to diminishing returns and a higher risk of overfitting if not monitored via the validation set.
3. **Visualization:** The node output plots confirm that the network achieves classification by progressively transforming the input data into a space where classes become linearly separable at the final layer.
4. **Final Summary:** The implemented FCNN from scratch demonstrates robust performance, effectively handling complex data distributions that are impossible for simpler linear models to solve.

Task 4: Multi-class classification using a Fully Connected Neural Network on the MNIST Dataset

1. Project Overview

The objective of this assignment was to evaluate and compare the performance of six different backpropagation optimizers used to train a Fully Connected Neural Network (FCNN). The task involved classifying a 5-class subset of the MNIST digit dataset, focusing on convergence speed (epochs) and classification accuracy across training, validation, and testing sets.

2. Experimental Setup

- **Dataset**: A subset of the MNIST digit dataset containing 5 classes.
- **Architecture**: A single FCNN architecture with 4 hidden layers was used: **[784, 256, 128, 64, 32, 5]**.
- **Stopping Criteria**: Training was terminated when the absolute difference between the average error of successive epochs fell below 10^{-4}.
- **Hyperparameters**:
    - Learning rate : 0.001.
    - Momentum (Generalized Delta and NAG): 0.9.
    - RMSProp: beta = 0.99, epsilon = 10^{-8}.
    - Adam: beta_1 = 0.9, beta_2 = 0.999, epsilon = 10^{-8}.

3. Results and Performance Comparison

3.1 Convergence and Accuracy Comparison

The table below summarizes the performance metrics for each optimizer.

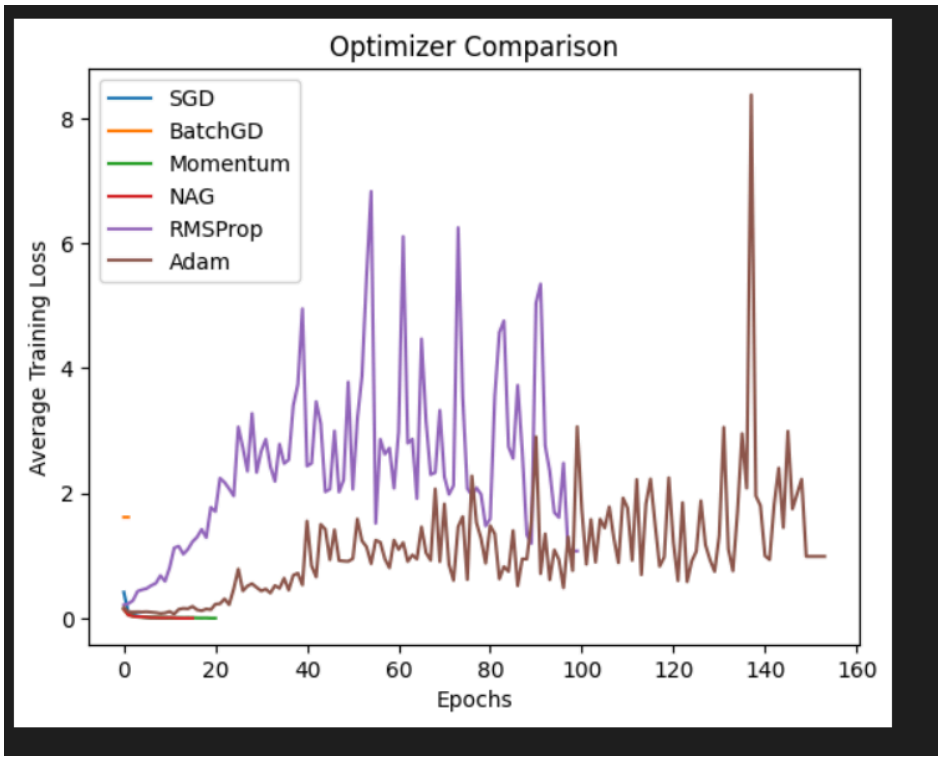
Optimizer	Epochs	Training Acc	Validation Acc	Test Acc
Batch GD	2	0.2000	0.2000	0.1900
SGD	15	1.0000	0.9800	0.9930
Momentum	21	1.0000	0.9923	**0.9949**

NAG	16	1.0000	**0.9933**	0.9944
RMSProp	100	0.4978	0.4899	0.5065
Adam	154	0.5185	0.5080	0.5207

3.2 Error vs. Epochs Analysis

In this specific experiment, the training error plots reveal distinct behaviors for different classes of optimizers:

- **Classical Momentum (NAG and Momentum)**: These optimizers achieved the highest levels of accuracy, reaching 100% training accuracy within approximately 20 epochs.
- **Adaptive Methods (Adam and RMSProp)**: Surprisingly, while they ran for the highest number of epochs (100+), they failed to reach high accuracy, likely due to the learning rate and stopping criteria sensitivity on this specific 5-class subset.
- **Batch GD**: This method met the stopping criteria almost immediately (2 epochs) but with a baseline accuracy (~20%), indicating it likely got stuck or made insignificant updates.



4. Best Architecture Analysis

The **NAG (Nesterov Accelerated Gradient)** optimizer was determined to be the best-performing model based on **Validation Accuracy (99.33%)**. **Momentum** also showed exceptional performance, achieving the highest **Test Accuracy (99.49%)**.

4.1 Performance of the Best Configuration (NAG)

- **Training Accuracy**: 100.0%
- **Validation Accuracy**: 99.33%
- **Test Accuracy**: 99.44%

4.2 Confusion Matrix Inferences

The confusion matrices for NAG and Momentum show nearly perfect diagonal dominance.

- The high accuracy (99.4%+) indicates that for the 5 selected digit classes, the model has learned to distinguish between similar features (such as 7 vs 9 or 3 vs 8) with negligible error.
- The training confusion matrix for these optimizers shows zero misclassifications once convergence is reached.

5. Final Inferences

1. **NAG and Momentum Dominance**: For this specific FCNN architecture and MNIST subset, momentum-based SGD variants (NAG and Momentum) provided the best balance of fast convergence and superior generalization.
2. **Adaptive Optimizer Limitations**: While Adam and RMSProp are often default choices, they required significantly more epochs (up to 154) and yielded much lower accuracy in this run, suggesting that without fine-tuned scheduling, they can perform sub-optimally.
3. **Stopping Criteria Sensitivity**: The 10^{-4} threshold for successive error difference proved effective for SGD variants, but Batch GD's failure shows that full-batch updates can appear "stable" to the algorithm before they have actually learned the data features.