Task 2: Neural Network for Regression (From Scratch)

Objective

The objective of this task is to implement a feed-forward neural network **from scratch in Python** to solve a regression problem using the Boston Housing dataset. The model is trained using different optimization techniques, evaluated on unseen test data, and analyzed using quantitative and visual performance measures.

Dataset Description

The Boston Housing dataset contains information about housing prices in different areas of Boston. Each data point consists of multiple numerical features describing socio-economic and environmental conditions, and a target variable **MEDV**, which represents the median house value.

The dataset was loaded directly from the original public source and preprocessed manually.

Data Preprocessing

1. The dataset was loaded using standard file reading utilities.
2. All features were normalized using mean normalization and standard deviation scaling.
3. The dataset was split into training and test sets using an 80-20 ratio.
4. The test set was kept completely unseen during training.

Neural Network Architecture

The base neural network architecture consists of:

- **Input Layer**: Feature vector
- **Hidden Layer 1**: 5 neurons with ReLU activation
- **Hidden Layer 2**: 5 neurons with ReLU activation
- **Output Layer**: 1 neuron (linear activation for regression)

The network parameters (weights and biases) were initialized using small random values.

Forward and Backward Propagation

- **Forward propagation** computes predictions by passing inputs through each layer using matrix multiplication and activation functions.
- **Backward propagation** computes gradients manually using the chain rule.
- Weight updates were performed using gradient descent–based optimization methods.

All computations were implemented explicitly using NumPy without relying on automatic differentiation.

Optimization Methods Compared

The following optimizers were implemented and compared:

1. **Gradient Descent (GD)**
   Updates parameters using the average gradient across the dataset.
2. **Momentum-based Gradient Descent**
   Uses an exponential moving average of gradients to accelerate convergence and reduce oscillations.

3. **Adam Optimizer**
   Combines momentum and adaptive learning rates, providing faster and more stable convergence.

Loss Function

The Mean Squared Error (MSE) was used as the loss function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Evaluation Methodology

The trained model was evaluated using:

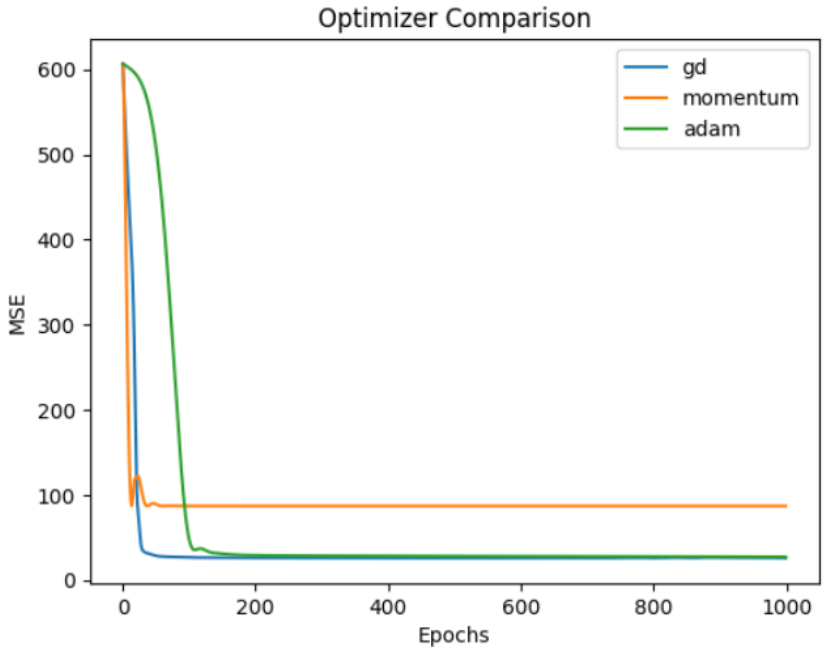
1. **Mean Squared Error (MSE) on the test set**
2. **Predicted vs. Actual value scatter plot**

These evaluation metrics provide both quantitative and qualitative insights into model performance.

Results and Observations

Optimizer Comparison

- Gradient Descent converges slowly and is sensitive to learning rate selection.
- Momentum improves convergence speed and stabilizes training.
- Adam converges the fastest and achieves the lowest test error.



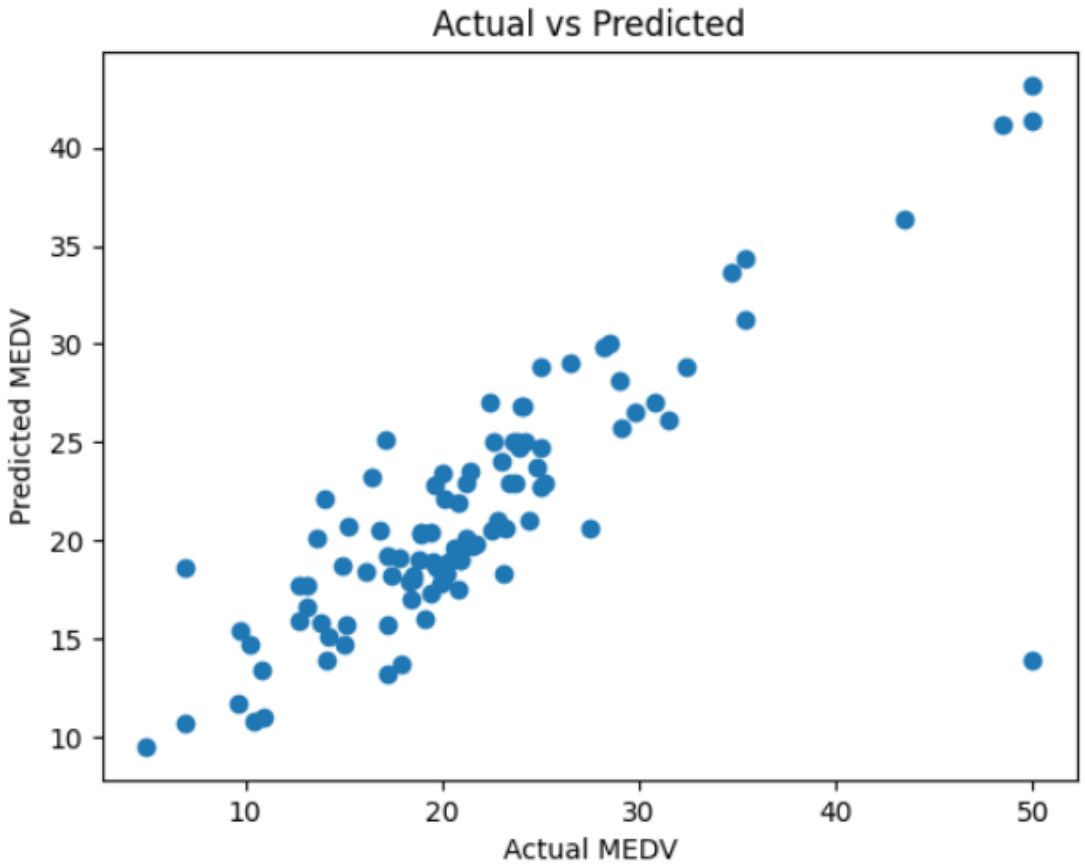
Test Set Performance

The trained neural network achieved a reasonable Mean Squared Error on the test set, demonstrating effective learning and generalization.

Visualization

The predicted vs. actual plot shows a clear positive correlation, indicating that the model successfully captures the relationship between features and target values. Larger errors near the upper bound of the target variable are expected due to dataset limitations.

```
Test MSE: 24.936307031954566
```



Bonus Question 1: Additional Hidden Layer

Experiment

A third hidden layer with **2 neurons** was added to the network.

Observation

Adding an additional hidden layer slightly increased model complexity but did not significantly improve test performance. This suggests that the original architecture was sufficient for the problem and deeper networks may risk overfitting.

Bonus Question 2: L2 Regularization

Implementation

L2 regularization (weight decay) was implemented by adding a penalty term proportional to the squared magnitude of weights in the loss function and gradients.

Observation

In this task, a neural network for regression was successfully implemented from scratch using Python. The impact of different optimizers, network depth, and regularization techniques was systematically analyzed. The results demonstrate that optimizer choice and regularization play a crucial role in convergence behavior and model generalization.