Multilayer Perceptron From Scratch for Wine Quality Prediction

1 Introduction

A Multilayer Perceptron (MLP) is a fully connected feedforward neural network capable of learning non-linear relationships between input features and target outputs. In this project, an MLP is implemented entirely from scratch using NumPy to predict the quality of red wine based on physicochemical properties. No deep learning frameworks are used, allowing full control and understanding of forward propagation, backpropagation, and parameter updates.

2 Dataset Description

The Wine Quality (Red) dataset from the UCI Machine Learning Repository contains 1599 samples. Each sample has 11 physicochemical input features such as acidity, pH, sulphates, and alcohol content. The target variable is the wine quality score, an integer typically ranging from 3 to 8.

3 Data Preprocessing

3.1 Feature Standardization

All input features are standardized using mean normalization:

$$X_{scaled} = \frac{X - \mu_X}{\sigma_X} \tag{1}$$

Standardization ensures that all features contribute equally to learning and improves convergence speed during training.

3.2 Train-Test Split

The dataset is split into training (80%) and testing (20%) subsets. The training set is used to learn parameters, while the test set evaluates generalization performance.

3.3 Target Scaling

The target variable is also standardized:

$$y_{scaled} = \frac{y - \mu_y}{\sigma_y} \tag{2}$$

Target scaling stabilizes gradient updates when using Mean Squared Error loss.

4 Network Architecture

The architecture of the MLP is as follows:

- Input layer: 11 neurons (one per feature)
- Hidden layer 1: 16 neurons with ReLU activation
- Hidden layer 2: 8 neurons with ReLU activation
- Output layer: 1 neuron with linear activation

Two hidden layers are used to allow the network to learn hierarchical feature representations. The number of neurons decreases gradually to compress information before prediction.

5 Weight Initialization

Weights are initialized using He initialization:

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}}}\right) \tag{3}$$

He initialization is specifically designed for ReLU networks and maintains stable activation variance across layers. Biases are initialized to zero.

6 Activation Functions

The Rectified Linear Unit (ReLU) activation function is used in hidden layers:

$$\text{ReLU}(z) = \max(0, z) \tag{4}$$

ReLU introduces non-linearity, prevents vanishing gradients, and enables efficient training.

The derivative used during backpropagation is:

$$\text{ReLU}'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases} \tag{5}$$

The output layer uses a linear activation because the problem is a regression task.

7 Forward Propagation

Forward propagation computes outputs layer by layer:

$$Z_1 = XW_1 + b_1 \tag{6}$$
$$A_1 = \text{ReLU}(Z_1) \tag{7}$$
$$Z_2 = A_1W_2 + b_2 \tag{8}$$
$$A_2 = \text{ReLU}(Z_2) \tag{9}$$
$$Z_3 = A_2W_3 + b_3 \tag{10}$$
$$\hat{y} = Z_3 \tag{11}$$

8 Loss Function

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

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{12}$$

MSE penalizes larger errors more strongly and is suitable for continuous target prediction.

9 Backpropagation

Backpropagation computes gradients using the chain rule.

Output layer gradient:

$$\frac{\partial \mathcal{L}}{\partial Z_3} = \frac{2}{m}(\hat{y} - y) \tag{13}$$

Hidden layer gradients:

$$\frac{\partial \mathcal{L}}{\partial Z_2} = \frac{\partial \mathcal{L}}{\partial A_2} \cdot \text{ReLU}'(Z_2) \tag{14}$$
$$\frac{\partial \mathcal{L}}{\partial Z_1} = \frac{\partial \mathcal{L}}{\partial A_1} \cdot \text{ReLU}'(Z_1) \tag{15}$$

Weight and bias gradients are computed using matrix multiplication.

10 Parameter Updates

Parameters are updated using batch gradient descent:

$$W \leftarrow W - \eta\frac{\partial \mathcal{L}}{\partial W} \tag{16}$$
$$b \leftarrow b - \eta\frac{\partial \mathcal{L}}{\partial b} \tag{17}$$

where η is the learning rate (0.05).

11 Training Process

The model is trained for 3000 epochs. During each epoch:

1. Forward propagation is performed
2. Loss is computed
3. Backpropagation computes gradients
4. Parameters are updated

Training loss is recorded to analyze convergence.

12 Testing and Evaluation

After training, predictions on the test set are unscaled:

$$y_{pred} = y_{scaled} \cdot \sigma_y + \mu_y \tag{18}$$

Model performance is visualized using:

- Training loss vs epochs plot
- Actual vs predicted wine quality scatter plot

13 Conclusion

This project demonstrates a complete implementation of a Multilayer Perceptron from scratch. By manually implementing preprocessing, initialization, forward propagation, backpropagation, and optimization, the internal working of neural networks is fully exposed and understood.