Binary Classification: Analysis of Logistic Regression and MLP

Zikun Zhuang
Southern University of Science and Technology
Shenzhen, China
12110418@mail.sustech.edu.cn

***Abstract*—Binary classification is a foundational problem in machine learning, and this project examines the performance of logistic regression and MLP (Multi-Layer Perceptron) in addressing this problem.**

Experiments are conducted on synthetic datasets represented in a two-dimensional plane, comparing the outcomes of logistic regression and neural networks with various hidden layer structures and neuron quantities. Evaluation metrics such as recall, precision, and F1 score are utilized to objectively assess the classification performance of the models.

Additionally, an analysis is carried out to understand the influence of hyperparameters, learning rates, and tolerance on the training process. The study investigates the impact of adjusting the number of neurons in each hidden layer and the overall layer count on the model's performance. The findings provide insights into the behavior of the loss function under different hyperparameters, and these results are presented through plots and tables.

***Index Terms*—binary classification, logistic regression, neural networks, Multi-Layer Perceptron, hyperparameters, performance evaluation**

I. Introduction

This document presents a comprehensive exploration of logistic regression and neural networks in the context of solving binary classification problems. The implementation involves a LogisticRegression class and a Multi-Layer Perceptron (MLP) class using Python and Numpy, with a primary focus on analyzing and comparing the performance of these models.

The investigation centers on the accuracy and complexity of the developed models. By conducting experiments on synthetic datasets, an evaluation is made regarding the practicality and efficiency of logistic regression and neural networks with varying hidden layer configurations. The study aims to provide insights into the accuracy and complexity of implementing and utilizing these models for binary classification tasks.

The analysis takes into consideration the impact of hyperparameters on the training process and explores how changes in the number of neurons in each hidden layer and the overall layer count influence model performance.

II. Problem formulation

The binary classification problem addressed in this project involves the categorization of samples into two classes based on their features. We aim to design and implement effective models for binary classification, specifically focusing on logistic regression and neural networks.

The primary objectives include:

- Implementing a LogisticRegression class using Python and Numpy for binary classification.
- Developing an MLP (Multi-Layer Perceptron) class to create fully connected neural networks with customizable hidden layer configurations.
- Experimenting with synthetic datasets to assess the performance of logistic regression and neural networks under different settings, such as varying hidden layer structures and neuron numbers or using different activation functions.

III. Method and algorithms

A. Methodology

This project primarily utilizes methods such as result comparison, parameter analysis, and references to existing literature for parameter optimization and model enhancement. These approaches aim to achieve clear and distinct experimental results, facilitating the evaluation of binary classification performance. Additionally, the use of specialized model algorithms and parameters ensures the model aligns with the requirements of binary classification, leading to optimal performance.

B. Algorithms

1) Logistic Regression: In contrast to linear binary classification models, logistic regression introduces the sigmoid function to replace the sign function, providing a continuous and differentiable output. The algorithm involves the calculation of predicted values, loss function computation, and gradient computation for weights. Finally, the decision boundary for binary classification tasks is visualized. The loss function, representing the disparity between predicted and actual values, is minimized through gradient descent during training.

- Prediction Output
  The sigmoid function is employed as the output function, given by:

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \tag{1}$$

  where

$$\sigma(\mathbf{z}) = \frac{1}{1 + \exp(-\mathbf{z})} \tag{2}$$

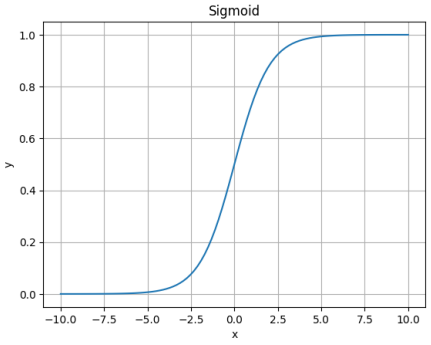

Fig. 1. Sigmoid function with $w_0 = 0, w_1 = 1$

- Loss Function
  The maximization of the likelihood function

$$L(\mathbf{w}) = \prod_{i=1}^{N} (1 - p(C = 0|\mathbf{x}^{(i)}))^{t^{(i)}} p(C = 0|\mathbf{x}^{(i)})^{1-t^{(i)}} \tag{3}$$

  is transformed into the minimization of the loss function

$$\begin{aligned} l_{\log}(\mathbf{w}) = &- \sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0|\mathbf{x}^{(i)}, \mathbf{w})) \\ &- \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0|\mathbf{x}^{(i)}, \mathbf{w}) \end{aligned} \tag{4}$$

  which further simplifies to:

$$l(\mathbf{w}) = \sum_{i} t^{(i)} z^{(i)} + \sum_{i} \log(1 + \exp(-\mathbf{z}^{(i)})) \tag{5}$$

  where

$$z = \mathbf{w}^T \mathbf{x} + w_0 \tag{6}$$

- Weight Updates
  The gradient of weights is given by:

$$\frac{\partial l}{\partial w_j} = \sum_{i} (t^{(i)} x_j^{(i)} - x_j^{(i)} \cdot \frac{\exp(-z^{(i)})}{1 + \exp(-z^{(i)})}) \tag{7}$$

  During model training, weights are updated using gradient descent:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \lambda \frac{\partial l}{\partial w_j^{(t)}} \tag{8}$$

2) Multilayer Perceptron (MLP): MLP consists of input, hidden, and output layers, with the number of hidden layers and neurons per layer customizable. The algorithm involves forward propagation, backward computation, and gradient descent training. Forward propagation calculates the output values of each hidden layer and the final output layer. Subsequently, the loss function of the MLP is computed based on the predicted output values. Backward computation includes calculating gradients for input, hidden, and output layer weights, facilitating gradient descent for model training.

- Forward Propagation
  Assuming the output layer's activation function is $g(\mathbf{z})$, and each layer has an activation function $f(\mathbf{z})$, the forward propagation for a single hidden layer is given by:

$$h_j(\mathbf{x}) = f(v_{j0} + \sum_{i=1}^{D} x_i v_{ji}) \tag{9}$$

$$o_k(\mathbf{x}) = g(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}) \tag{10}$$

  For multiple hidden layers, the weights are replaced with the corresponding weights from the previous hidden layer, and the inputs are replaced with the outputs of the previous hidden layer.

  1) Activation Function Selection
     For binary classification tasks in MLP, choosing the appropriate activation function is crucial. ReLU is selected as the activation function for hidden layers, and sigmoid is chosen for the output layer:

$$f(\mathbf{z}) = \text{ReLU}(\mathbf{z}) = \max\{0, \mathbf{z}\} \tag{11}$$

$$g(\mathbf{z}) = \sigma(\mathbf{z}) = \frac{1}{1 + \exp(-\mathbf{z})} \tag{12}$$

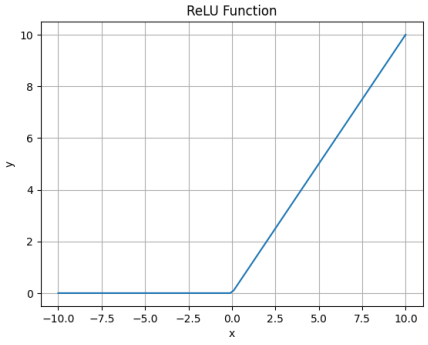

Fig. 2. ReLU function

     ReLU is chosen for its ability to mitigate the vanishing gradient problem, and sigmoid is suitable for binary classification, similar to logistic regression.

  2) Weight Initialization
     Appropriate weight initialization is crucial for different activation functions. For ReLU, He initialization [1] is applied:

$$W_i \sim \mathcal{N}\left[0, \frac{2}{n}\right] \tag{13}$$

     where n is the number of neurons in the previous layer.

For sigmoid, Xavier initialization [2] is used:

$$V \sim \mathcal{N}\left[0, \frac{2}{n_i + n_{i+1}}\right] \tag{14}$$

where n_i is the input dimension of the previous layer, and n_{i+1} is the number of neurons in the current layer.

- Backward Computation

1) Output Layer Weight Gradient Calculation

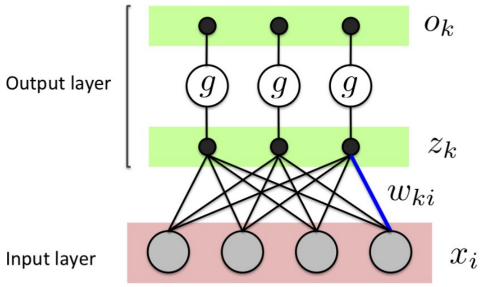

Fig. 3. MLP with zero hidden layers

The gradient of weights for the output layer is given by:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial z_k}\frac{\partial z_k}{\partial w_{ki}} \tag{15}$$

Assuming the mean squared error loss function:

$$E = \frac{1}{2}\sum_{n=1}^{N}(t_k^{(n)} - o_k^{(n)})^2 \tag{16}$$

then

$$\frac{\partial E}{\partial o_k^{(n)}} = o_k^{(n)} - t_k^{(n)} \tag{17}$$

and

$$o_k^{(n)} = \sigma(z_k^{(n)}) \tag{18}$$

resulting in

$$\frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} = o_k^{(n)}(1 - o_k^{(n)}) \tag{19}$$

Thus, we can derive the gradient of weights for the output layer:

$$\frac{\partial E}{\partial w_{ki}} = \sum_{n=1}^{N}(o_k^{(n)} - t_k^{(n)})o_k^{(n)}(1 - o_k^{(n)})x_i^{(n)} \tag{20}$$

2) Hidden Layer Weight Calculation
For a single hidden layer:

$$\frac{\partial E}{\partial h_j^{(n)}} = \sum_k \frac{\partial E}{\partial o_k^{(n)}}\frac{\partial o_k^{(n)}}{\partial z_k^{(n)}}\frac{\partial z_k^{(n)}}{\partial h_j^{(n)}} \tag{21}$$

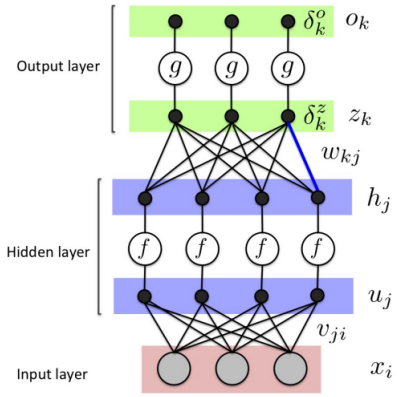

Fig. 4. MLP with one hidden layer

Substituting the previously calculated result:

$$\frac{\partial E}{\partial h_j^{(n)}} = \sum_k (o_k^{(n)} - t_k^{(n)})o_k^{(n)}(1 - o_k^{(n)})w_{kj} \tag{22}$$

The gradient for weight v is then:

$$\frac{\partial E}{\partial v_{ji}} = \sum_{n=1}^{N}\frac{\partial E}{\partial h_j^{(n)}}\frac{\partial h_j^{(n)}}{\partial u_j^{(n)}}\frac{\partial u_j^{(n)}}{\partial v_{ji}} \tag{23}$$

where

$$\frac{\partial h_j^{(n)}}{\partial u_j^{(n)}} = \begin{cases} 1 & \text{if } h_j^{(n)} > 0, \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

$$\frac{\partial u_j^{(n)}}{\partial v_{ji}} = x_i^{(n)} \tag{25}$$

The same procedure applies for multiple hidden layers.

3) Gradient Updates
For any gradient w,

$$w^{(t+1)} \leftarrow w^{(t)} - \lambda\frac{\partial E}{\partial w^{(t)}}$$

The above steps are repeated in a training loop for forward propagation, backward computation, and gradient updates to minimize the loss function and complete MLP training.

IV. Experiment Results and Analysis

To control for variables and compare the performance of different models, all experiments in this project are conducted using the same dataset for model training.

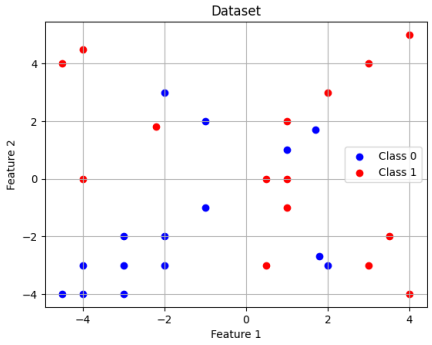

Fig. 5. Data-set for the binary classification problem

A. Logistic Regression

To assess the impact of different learning rates on a logistic regression model, the dataset was utilized, and the number of iterations was fixed at 100,000. The obtained results are presented in the following table:

Comparing the final loss values for different learning rates, it is observed that the loss increases as the learning rate exceeds 0.01, indicating a degradation in model accuracy. Additionally, examining the convergence speed of the loss, a reduction of 76.52% in the loss is achieved after 30 iterations at a learning rate of 0.01, demonstrating the best performance. In conclusion, with 100,000 iterations, the optimal learning rate is determined to be 0.01.

TABLE I
COMPARISON OF DIFFERENT LEARNING RATES

λ	Final Loss Value	$\Delta loss$ after 30 iterations ($\Delta l = \frac{l_0 - l}{l_0}$)
0.001	15.751	57.85%
0.01	15.751	76.52%
0.05	18.28	73.60%
0.1	31.74	54.20%

Results after 100,000 iterations.

Fig. 6. Decision boundary for logistic regression with $\lambda = 0.01$

To evaluate the logistic regression model with a learning rate of 0.01 and 100,000 iterations, the results are assessed through the Confusion Matrix, Precision, Recall, and F1 Score, yielding the following outcomes:

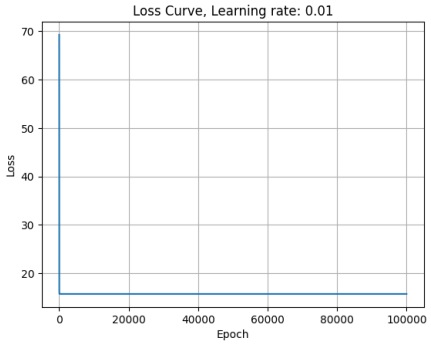

Fig. 7. Loss curve for logistic regression with $\lambda = 0.01$

TABLE II
PERFORMANCE METRICS FOR LOGISTIC REGRESSION MODEL

Confusion Matrix	Precision	Recall	F1 Score
$\begin{bmatrix} 10 & 5 \\ 2 & 13 \end{bmatrix}$	0.72	0.87	0.79

The Confusion Matrix illustrates the model's performance in terms of true positive (10), false positive (5), false negative (2), and true negative (13). Precision, Recall, and F1 Score are calculated as 0.72, 0.87, and 0.79, respectively, providing a comprehensive assessment of the model's classification performance.

B. MLP

1) L2 Regularization: Due to the tendency of Multilayer Perceptron (MLP) models to overfit, especially in cases with a large number of neurons or layers, L2 regularization [3] is incorporated in this project to prevent overfitting. The modified loss function is defined as follows:

$$E_{L2} = E + \frac{\alpha}{2}\left(\sum_{i=1}^{N_1} v_i^2 + \sum_{i=1}^{N_2} w_i^2 + \sum_{i=1}^{N_3}\sum_{j=1}^{N_i} k_{ij}^2\right) \tag{26}$$

Here, E_{L2} represents the loss function with added L2 regularization, E is the original loss function, α is the hyperparameter controlling the regularization term, and v_i, w_i, and k_{ij} denote the weights of the input layer, output layer, and hidden layers, respectively.

Simultaneously, the gradients of the weights are adjusted. For any weight w, the new gradient is given by:

$$\frac{\partial E}{\partial w}_{\text{new}} = \frac{\partial E}{\partial w} + \alpha w \tag{27}$$

2) Analysis of Experimental Results:

- Single Hidden Layer
  Consideration is initially given to a single hidden layer. MLP models with hidden layer neuron counts of 5, 10, and 20 are employed for binary classification on the dataset.
  For each model, the learning rate λ, regularization coefficient α, and the number of iterations are adjusted.

The performance is evaluated based on the loss function curve, the final value of the loss function, Confusion Matrix, Precision, Recall, and F1 Score. Smaller final loss values and higher Precision, Recall, and F1 Scores indicate better model performance. Taking the case of neuron count $n = 20$ as an example (with 150,000 iterations):

TABLE III
PARAMETERS AND FINAL LOSS FOR $n = 20$

#	λ	α	**Final Loss**
1	0.003	0.000	1.001
2	0.003	0.003	0.180
3	0.010	0.003	0.176
4	0.010	0.000	1.000

The Confusion Matrix, Precision, Recall, and F1 Score for each result are as follows:

TABLE IV
PERFORMANCE METRICS FOR $n = 20$

#	**Confusion Matrix**	**Precision**	**Recall**	**F1 Score**
1	$\begin{bmatrix} 13 & 2 \\ 0 & 15 \end{bmatrix}$	0.88	1.0	0.9375
2	$\begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix}$	1.00	1.0	1.0000
3	$\begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix}$	1.00	1.0	1.0000
4	$\begin{bmatrix} 13 & 2 \\ 0 & 15 \end{bmatrix}$	0.88	1.0	0.9375

By comparing the tables, it is evident that the model performs best when $n = 20$, $\lambda = 0.01$, and $\alpha = 0.003$. This is attributed to the overfitting phenomenon in the model when $n = 20$, necessitating the introduction of a regularization term to mitigate overfitting. Additionally, a too-small learning rate can slow down the convergence of the loss function.

For the models with neuron counts of 5 and 10, a similar comparative approach is employed to identify the optimal parameter combinations. The binary classification performances for neuron counts 5, 10, and 20 are compared (all with 150,000 iterations):

TABLE V
PARAMETERS AND FINAL LOSS FOR DIFFERENT NEURON COUNTS

n	λ	α	**Final Loss**
5	0.03	0.000	1.6000
10	0.01	0.000	0.0005
20	0.01	0.003	0.1760

The corresponding performance metrics are provided in the following table:

TABLE VI
PERFORMANCE METRICS FOR DIFFERENT NEURON COUNTS

n	**Confusion Matrix**	**Precision**	**Recall**	**F1 Score**
5	$\begin{bmatrix} 13 & 2 \\ 2 & 13 \end{bmatrix}$	0.87	0.87	0.87
10	$\begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix}$	1.00	1.00	1.00
20	$\begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix}$	1.00	1.00	1.00

Comparison reveals that as the neuron count increases, it is necessary to moderately decrease the learning rate to achieve better performance. Excessive neuron counts may lead to overfitting, necessitating the introduction of an L2 regularization term to restrict weight growth. Overall, for the binary classification dataset used in this project, the model with 10 neurons performs the best.

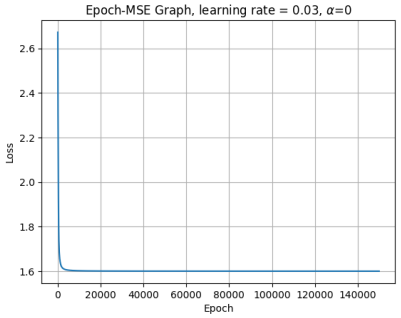

Fig. 8. Loss curve for single hidden layer MLP with $n = 5$

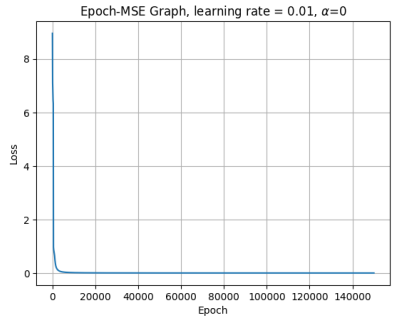

Fig. 9. Loss curve for single hidden layer MLP with $n = 10$

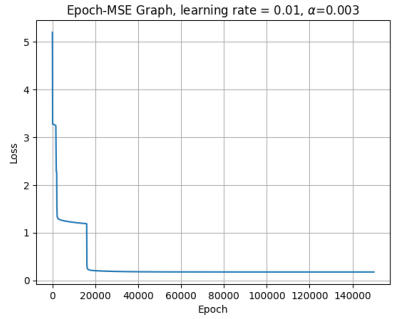

Fig. 10. Loss curve for single hidden layer MLP with $n = 20$

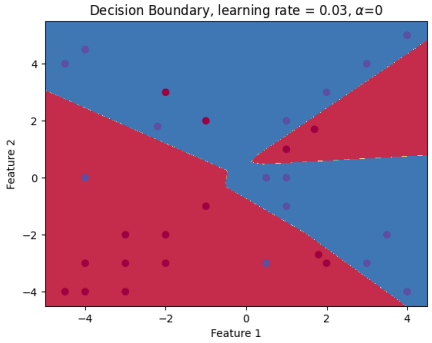

Fig. 11. Decision boundary for single hidden layer MLP with $n = 5$

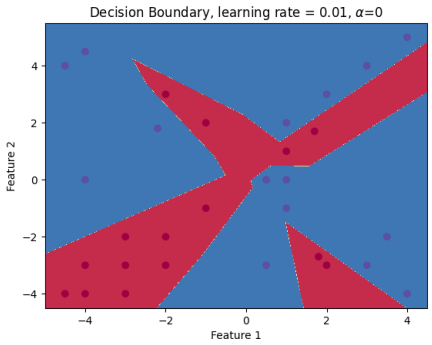

Fig. 12. Decision boundary for single hidden layer MLP with $n = 10$

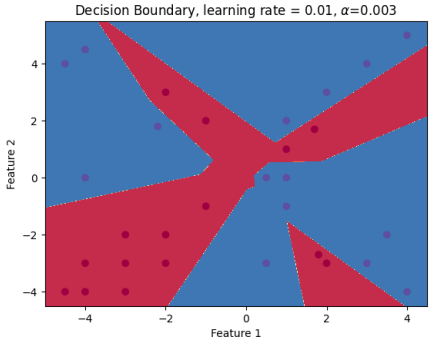

Fig. 13. Decision boundary for single hidden layer MLP with $n = 20$

- Multiple Hidden Layers
  Next, we consider the case of multiple hidden layers. With multiple hidden layers, the number of neurons in each layer can vary. We compare MLP models with hidden layer structures [5], [5,4], and [6,5,5,5] (all with 150,000 iterations). The numbers represent the current layer's neuron count.
  Similar to the single hidden layer case, we search for the best parameter combinations for each model to compare the performance of models with different hidden layer structures.
  The corresponding performance are as follows:

TABLE VII
PARAMETERS AND FINAL LOSS FOR DIFFERENT HIDDEN LAYER STRUCTURES

Hidden Layer Structure	λ	α	**Final Loss**
5	0.03	0.000	1.6000
5,4	0.007	0.000	0.5016
6,5,5,5	0.01	0.005	0.0819

TABLE VIII
PERFORMANCE FOR DIFFERENT HIDDEN LAYER STRUCTURES

Hidden Layer Structure	**Confusion Matrix**	**F1 Score**
5	$\begin{bmatrix} 13 & 2 \\ 2 & 13 \end{bmatrix}$	0.87
5,4	$\begin{bmatrix} 15 & 0 \\ 1 & 14 \end{bmatrix}$	0.97
6,5,5,5	$\begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix}$	1.00

Comparing the models, when the number of neurons in each layer is similar, choosing a multi-layer MLP model can result in better performance. However, it is crucial to introduce a regularization term to prevent overfitting. Additionally, adjusting the parameters for the [6,5,5,5] structure reveals that increasing the regularization coefficient α requires a corresponding increase in the learning rate λ to allow the loss function to converge more rapidly to a smaller value. However, a higher α can lead to more pronounced fluctuations in the iteration-loss curve. Thus, finding a balance between α and λ is essential for achieving better binary classification results.

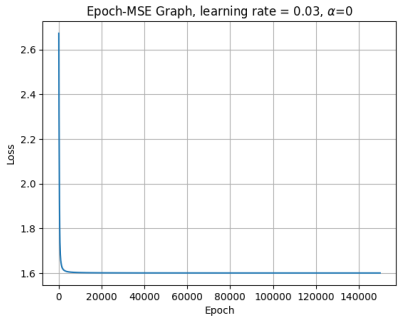

Fig. 14. Loss curve for single hidden layer MLP with $n = 5$

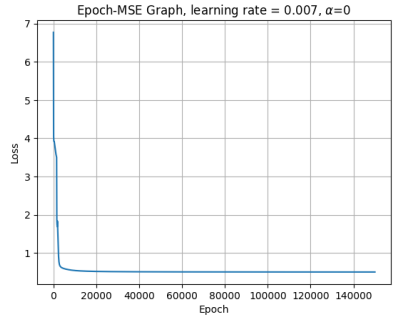

Fig. 15. Loss curve for 2 hidden layers ([5,4]) MLP

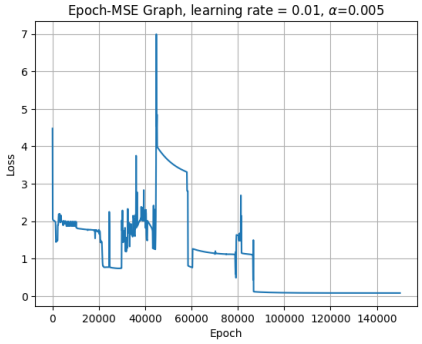

Fig. 16. Loss curve for 4 hidden layers ([6,5,5,5]) MLP

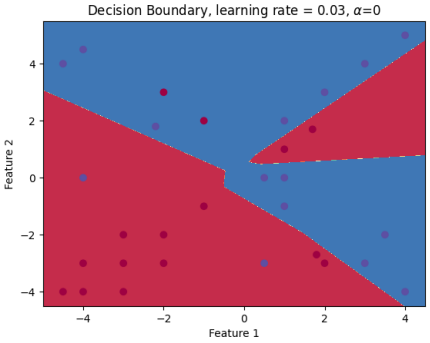

Fig. 17. Decision boundary for single hidden layer MLP with $n = 5$

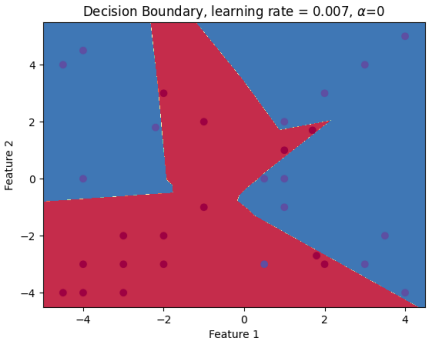

Fig. 18. Decision boundary for 2 hidden layers ([5,4]) MLP

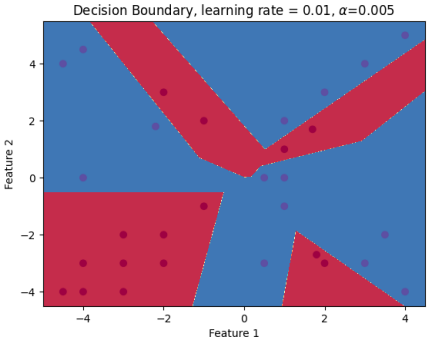

Fig. 19. Decision boundary for 4 hidden layers ([6,5,5,5]) MLP

- More Complex Dataset
  In comparison to the logistic regression model, MLP exhibits stronger binary classification capabilities. Therefore, we consider a more complex scenario involving a randomly generated dataset with 100 data points to observe the classification performance of the MLP model.

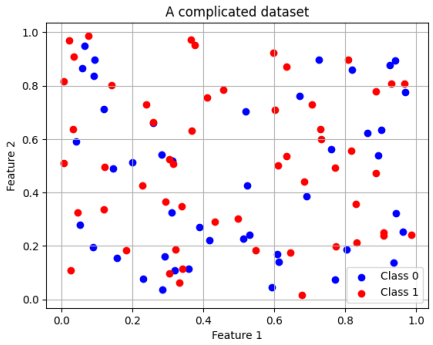

Fig. 20. A complex dataset

Two distinct MLP models were trained on a complex dataset, differing in their hidden layer structures: a single layer with 100 neurons and a four-layer architecture with 50 neurons each. Both models underwent 175,000 iterations.

For the single hidden layer model with 100 neurons, the parameters were set as follows: a learning rate (λ) of 0.0001 and a regularization coefficient (α) of 5×10^{-5}. The four-layer model with 50 neurons per layer was configured with a learning rate (λ) of 0.0001 and no regularization ($\alpha = 0$).

Comparison of the models using Confusion Matrix and F1 Score reveals superior performance of the four-layer model, achieving flawless binary classification on the dataset.

TABLE IX
PERFORMANCE METRICS FOR MLP MODELS WITH DIFFERENT HIDDEN LAYER STRUCTURES

Hidden Layer Structure	Confusion Matrix	F1 Score
Single Layer (100 Neurons)	$\begin{bmatrix} 33 & 12 \\ 12 & 43 \end{bmatrix}$	0.78
Four Layers (50 Neurons Each)	$\begin{bmatrix} 45 & 0 \\ 0 & 55 \end{bmatrix}$	1.00

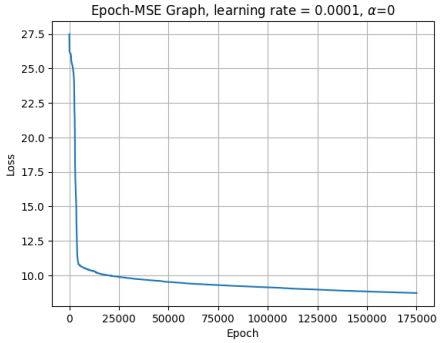

Fig. 21. Loss Curve for single hidden layer MLP with $n = 100$

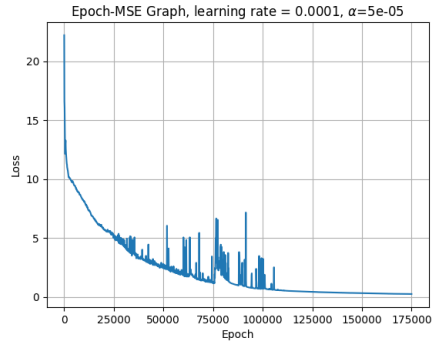

Fig. 22. Loss Curve for 4 hidden layers MLP (each with 50 neurons)

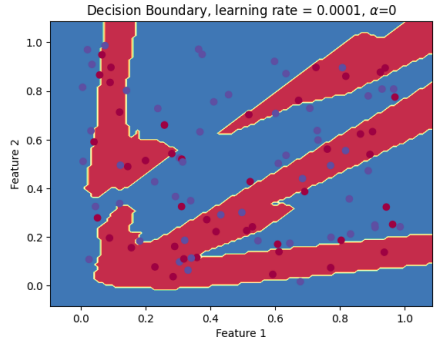

Fig. 23. Decision boundary for single hidden layer MLP with $n = 100$

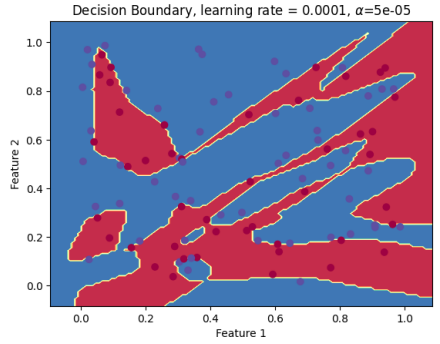

Fig. 24. Decision boundary for 4 hidden layers MLP (each with 50 neurons)

In summary, MLP demonstrates notable learning capabilities even in the presence of complex models. It can achieve effective binary classification performance by adjusting the number of hidden layers and various hyperparameters.

V. Conclusion and Future Problems

A. Conclusion

This project delved into the core algorithms of logistic regression and MLP neural networks, strategically selecting function models and hyperparameters for binary classification tasks. Comprehensive comparisons were made using various metrics, including the loss function, Confusion Matrix, Precision, Recall, and F1 Score.

In logistic regression, the study probed the impact of diverse learning rates on the model's learning effectiveness, pinpointing an optimal rate for stable and efficient binary classification.

For MLP, the investigation initially focused on the influence of neuron count in a single hidden layer. The study emphasized the necessity of selecting an appropriate number of neurons to achieve robust performance, avoiding pitfalls like underfitting or overfitting. To counter overfitting, L2 regularization with a penalty term was introduced, and the study explored how learning rates varied with different neuron counts.

Subsequently, the study delved into the dynamics of multiple hidden layers, scrutinizing how the number of hidden layers influenced model effectiveness. The intricate relationships between the number of hidden layers, learning rates, and the interplay between learning rates and penalty terms were also thoroughly examined.

B. Future Problem

1) Residual Networks (ResNets) and Gradient Vanishing: Explore the impact of residual network structures on mitigating the issue of gradient vanishing. Implement He initialization and Xavier initialization for weights to address the problem of vanishing gradients.

2) Complex Binary Classification Tasks: The current project used relatively simple datasets; future work should involve exploring more complex binary classification tasks to better understand the capabilities and limitations of logistic regression and MLP neural networks in intricate scenarios.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, pp. 1026-1034, doi: 10.1109/ICCV.2015.123.

[2] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the thirteenth international conference on artificial intelligence and statistics, March 2010, pp. 249-256. JMLR Workshop and Conference Proceedings.

[3] C. M. Bishop, Pattern Recognition and Machine Learning, 1st ed. Springer, 2006.