Assignment 1 Part A : Logistic Regression with Varying Batch Sizes

Deep Learning - EE569

Motaz M Alharbi
2190203271

Instuctor: Dr. Nuri Benbarka

Contents

1 Introduction

This report details the implementation of a logistic regression model using a custom-built computational graph framework. The primary objectives were to construct a `Linear` computation node, integrate it into the model, introduce batch processing capabilities, and conduct an empirical study on the effects of varying batch sizes on the model's training dynamics and performance. The experiment systematically analyzes training loss, convergence speed, and final classification accuracy across a range of batch sizes, from stochastic (size 1) to large mini-batches.

2 Task 1: Implementing a Linear Computation Node

2.1 Task Description

The initial task was to create a `Linear` computation node. This node encapsulates the affine transformation fundamental to linear models, defined by the equation:

$$y = A \cdot x + b \tag{1}$$

where A is the weight matrix, x is the input vector, and b is the bias vector. The node must support both a forward pass to compute the output y and a backward pass to compute the gradients of the loss with respect to its inputs (A, x, and b).

2.2 Implementation Details

The `Linear` class was implemented inheriting from a base `Node` class.

Forward Pass: The `forward` method computes $A \cdot x + b$ using NumPy's matrix multiplication (`@`) and broadcasting for the addition of the bias term.

Backward Pass: The `backward` method calculates the gradients based on the chain rule, using the upstream gradient $\frac{\partial L}{\partial y}$ propagated from the subsequent node. The gradients are:

$$\frac{\partial L}{\partial x} = A^T \cdot \frac{\partial L}{\partial y} \tag{2}$$

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial y} \cdot x^T \tag{3}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N} \left(\frac{\partial L}{\partial y} \right)_i \tag{4}$$

where the sum for the bias gradient is performed across the batch dimension. The implementation was verified by ensuring its correct integration into the full computational graph.

3 Task 2: Integrating the Linear Node

3.1 Model Architecture

The newly created `Linear` node was integrated into a logistic regression model. This integration simplified the computation graph, replacing what would have been separate `Multiply` and `Addition` nodes. The complete model architecture is as follows:

$$\textit{Input } (x, y_{true}) \rightarrow \textit{Parameters } (A, b) \rightarrow \textbf{Linear} \rightarrow \textit{Sigmoid} \rightarrow \textit{BCE Loss}$$

This streamlined graph correctly computes the forward pass and propagates gradients back to the trainable parameters (A and b) during the backward pass.

4 Task 3: Introducing Batching

4.1 Description

To improve computational efficiency and training stability, the model was modified to process data in batches. Instead of feeding one sample at a time, the input x becomes a matrix of shape $(n_{\text{features}}, n_{\text{batch\_size}})$, where each column is a data point.

4.2 Implementation

The main training loop in `a.py` was updated to slice the shuffled training data into mini-batches of a specified size. The `Linear` node's implementation already supported batch operations due to NumPy's broadcasting rules, particularly for adding the bias vector b to the matrix product $A \cdot x$. The loss calculation was also adjusted to average over the batch.

5 Task 4: Investigating the Effect of Batch Size

5.1 Experimental Setup

An experiment was designed to observe the impact of batch size on model training. The model was trained for 50 epochs with a fixed learning rate of 0.03. The following batch sizes were tested: 1, 4, 8, 16, 64, 128, 256, 512, and 1024. For each batch size, the training loss per epoch and the final accuracy on a held-out test set were recorded.

5.2 Results and Discussion

The experiment yielded clear insights into the trade-offs associated with batch size selection.

5.2.1 Training Loss

The training loss behavior across epochs is visualized in the bottom-right panel of Figure 1.

Small Batch Sizes (1, 4, 8): These sizes exhibit rapid initial convergence, as seen by the steep drop in loss during the first few epochs. However, the loss curves are noticeably noisier, reflecting the high variance of gradient estimates from small samples.

Large Batch Sizes (128, 256, ...): These sizes show a much smoother, more stable descent in training loss. The gradient estimates are more accurate representations of the true gradient. However, the rate of convergence in terms of epochs is slower, as each epoch involves fewer weight updates.

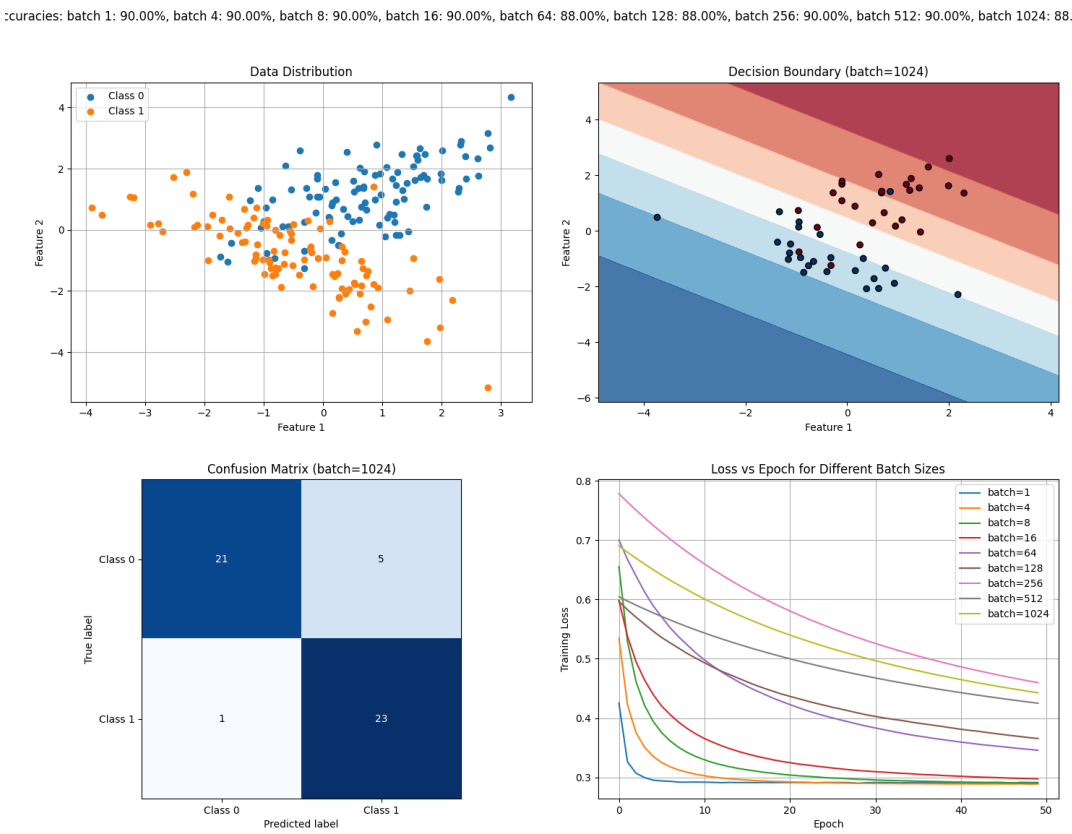


Figure 1: Combined results showing data distribution, decision boundary for batch size 1024, confusion matrix, and a comparison of training loss vs. epoch for all tested batch sizes.

5.2.2 Model Accuracy

The final test accuracies for each batch size are summarized in Table 1. For this dataset, the final accuracy was not highly sensitive to the batch size, with most models achieving an accuracy between 88.00% and 90.00%. This suggests that all training regimens found a similarly effective decision boundary.

Table 1: Test Accuracy for Different Batch Sizes

Batch Size	**Test Accuracy (%)**
1	90.00
4	90.00
8	90.00
16	90.00
64	88.00
128	88.00
256	90.00
512	90.00
1024	88.00

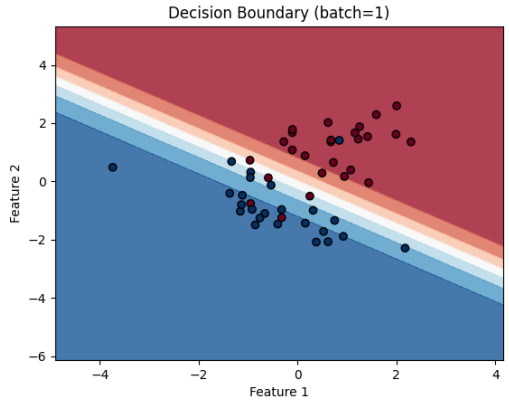
5.2.3 Decision Boundaries and Confusion Matrices

The final decision boundaries learned by the model were consistent across all batch sizes, as shown in Figure 2. They all successfully separate the two classes of data. This is further corroborated by the confusion matrices in Figure 3, which show similar patterns of true positives, true negatives, and errors.

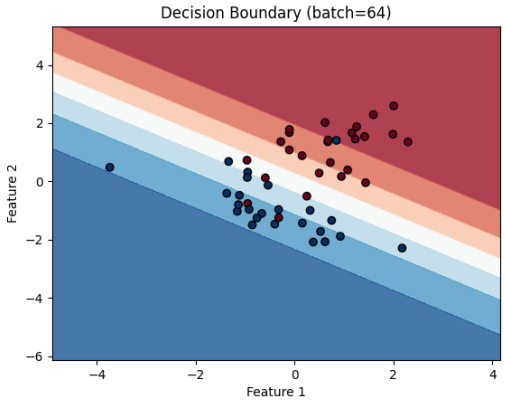
6 Conclusion

This assignment successfully demonstrated the process of building, training, and evaluating a logistic regression model within a custom computational graph framework. The investigation into batch size revealed a classic machine learning trade-off. Smaller batches lead to faster but more erratic convergence, while larger batches provide stable but slower convergence per epoch. For the given problem, the choice of batch size had a minimal impact on the final model's classification accuracy, but significantly altered the training process. This highlights that selecting an optimal batch size is problem-dependent and often involves balancing computational efficiency with convergence speed and stability.

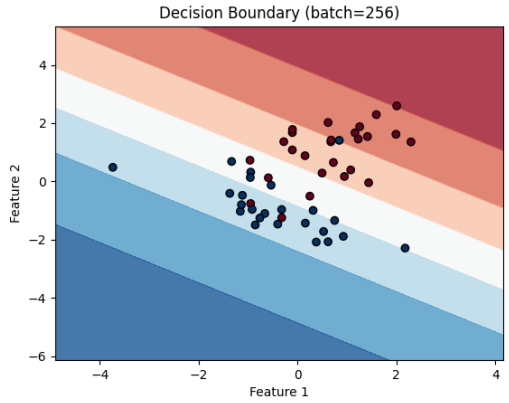
A Additional Figures



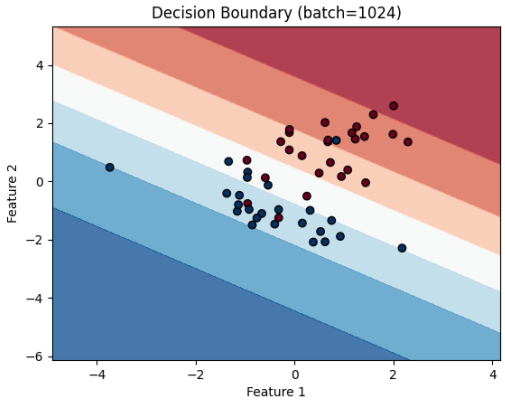
(a) Batch Size = 1



(b) Batch Size = 64

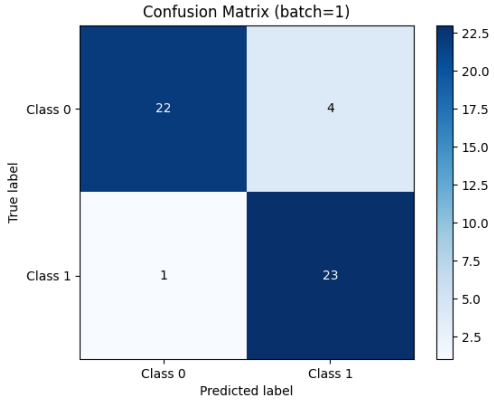


(c) Batch Size = 256

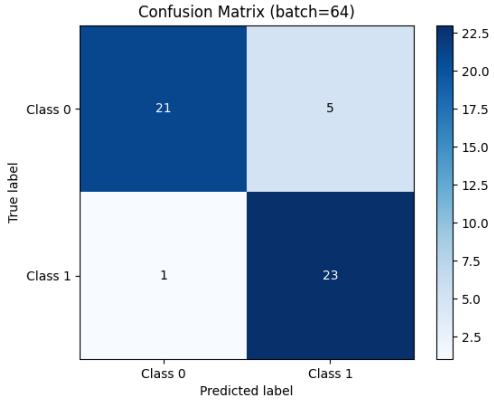


(d) Batch Size = 1024

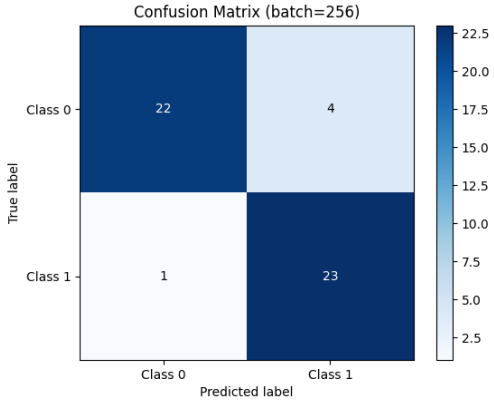
Figure 2: Comparison of Decision Boundaries for selected batch sizes.



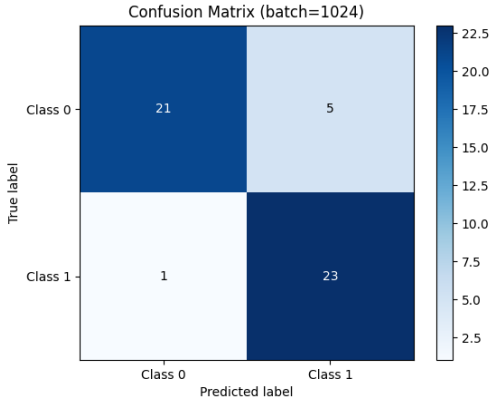
(a) Batch Size = 1



(b) Batch Size = 64



(c) Batch Size = 256



(d) Batch Size = 1024

Figure 3: Comparison of Confusion Matrices for selected batch sizes.