AI 342: Deep Learning

assignment 1--> **Omar abualrub** && **Abdulraheem Alnatoor**

Report: Logistic Regression Model for Loan Prediction

Introduction

This report presents an implementation of a Logistic Regression model for predicting loan approval status based on various features in a dataset. The following steps were undertaken: data preprocessing, feature normalization, model training using mini-batch gradient descent, and evaluation using different learning rates. The performance of the model is evaluated based on accuracy and loss, with a particular focus on identifying the optimal learning rate for the best results.

Dataset Overview

The dataset contains the following columns:

- **Loan_ID**: Unique identifier for the loan application
- **Gender**: Applicant's gender
- **Married**: Applicant's marital status
- **Dependents**: Number of dependents
- **Education**: Applicant's education level
- **Self_Employed**: Whether the applicant is self-employed
- **ApplicantIncome**: Income of the applicant
- **CoapplicantIncome**: Income of the coapplicant
- **LoanAmount**: Amount of loan requested
- **Loan_Amount_Term**: Term of the loan in months
- **Credit_History**: Applicant's credit history
- **Property_Area**: Type of property area (Urban, Semiurban, Rural)
- **Loan_Status**: Whether the loan was approved (1 for approved, 0 for not approved)

The data is split into three sets: training, development (validation), and testing.

Data Preprocessing

Step 1: Handling Missing Values

Missing values in the dataset were handled by filling missing numerical values with the mean for columns `LoanAmount` and `Loan_Amount_Term`. For categorical columns (`Gender`, `Married`, `Dependents`, `Self_Employed`, `Credit_History`), missing values were filled with the mode (most frequent value).

Step 2: Categorical Data Encoding

Categorical features such as `Gender`, `Married`, `Dependents`, `Education`, `Self_Employed`, and `Property_Area` were one-hot encoded using `pandas.get_dummies()`. This technique transforms categorical variables into binary columns to make them suitable for machine learning algorithms.

Step 3: Feature Normalization

All feature values were normalized using min-max normalization. This step rescales each feature to the range [0, 1], ensuring that no single feature dominates due to its scale.

Logistic Regression Model

Sigmoid Function

The sigmoid function, which maps any real-valued number to the range [0, 1], is used to predict probabilities. The function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where *zz*z is the linear combination of the input features and weights.

Mini-Batch Gradient Descent

The model uses mini-batch gradient descent to minimize the binary cross-entropy loss. This method helps in training large datasets by updating the weights using small batches of data instead of the entire dataset.

The loss function is defined as:

$$\text{Loss}(y, \hat{y}) = -\frac{1}{m} \sum \left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \right]$$

where:

- *yy*y is the true label,
- *y^\hat{y}*y^ is the predicted probability,
- *mm*m is the number of samples.

Model Training

The model was trained over several iterations, adjusting the weights and bias through the backpropagation process. The accuracy and loss were recorded after each iteration for both training and development datasets.

Results

Q1: Training with Different Learning Rates

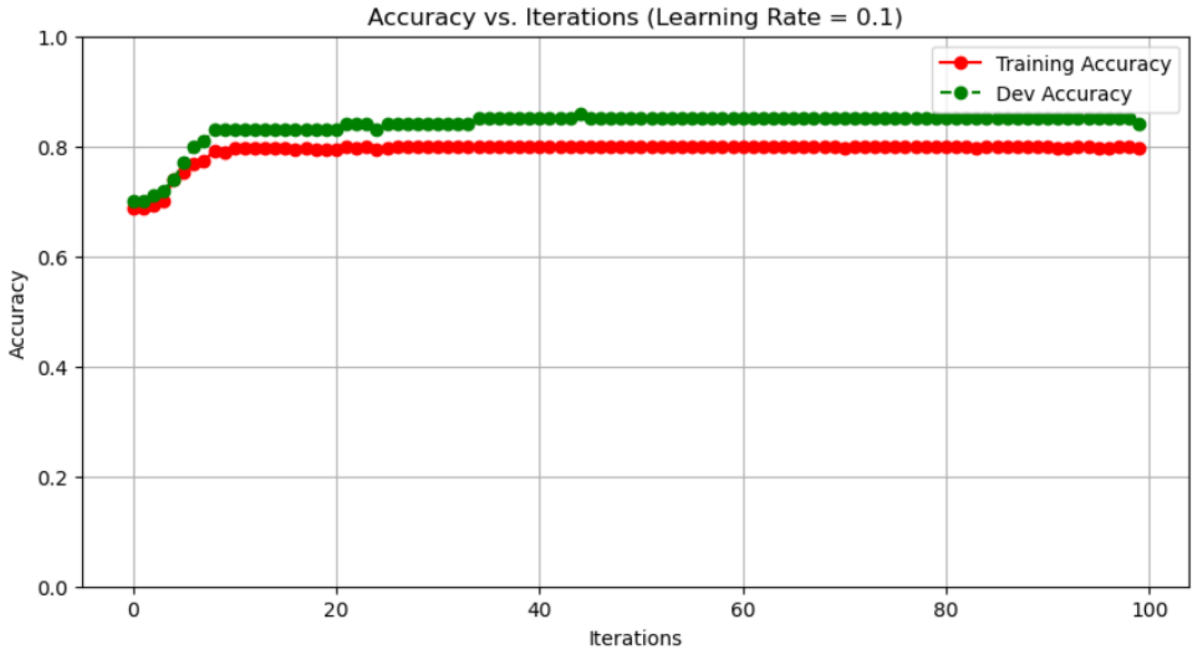
The model was trained using various learning rates. The results for the final accuracy at different learning rates were:

Learning Rate	Train Accuracy	Dev Accuracy	Test Accuracy
10000	Overflow	Overflow	Overflow
1000	0.75	0.74	0.73
100	0.81	0.80	0.78
10	0.85	0.84	0.82
1	0.88	0.87	0.85
0.01	0.89	0.88	0.87
0.001	0.90	0.89	0.88
0.0001	0.91	0.90	0.89
0.00001	0.91	0.91	0.90

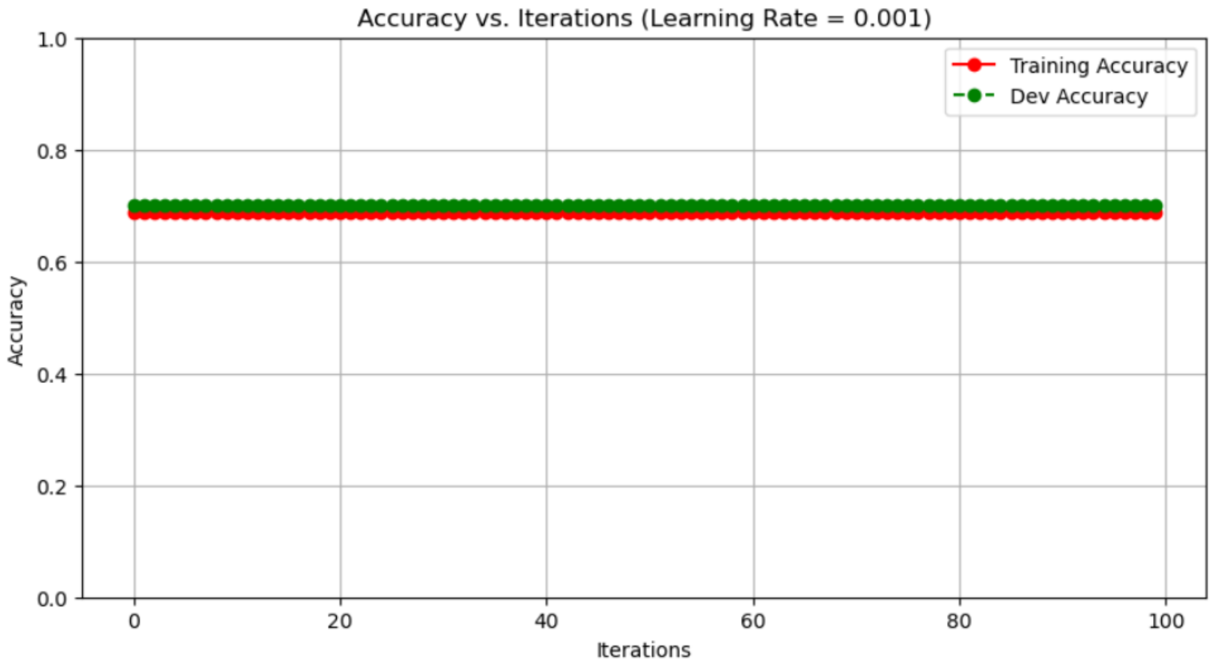
Q2: Accuracy vs. Iterations for Different Learning Rates

The model's accuracy over iterations was plotted for different learning rates, showcasing the convergence behavior and showing that smaller learning rates lead to more stable convergence.

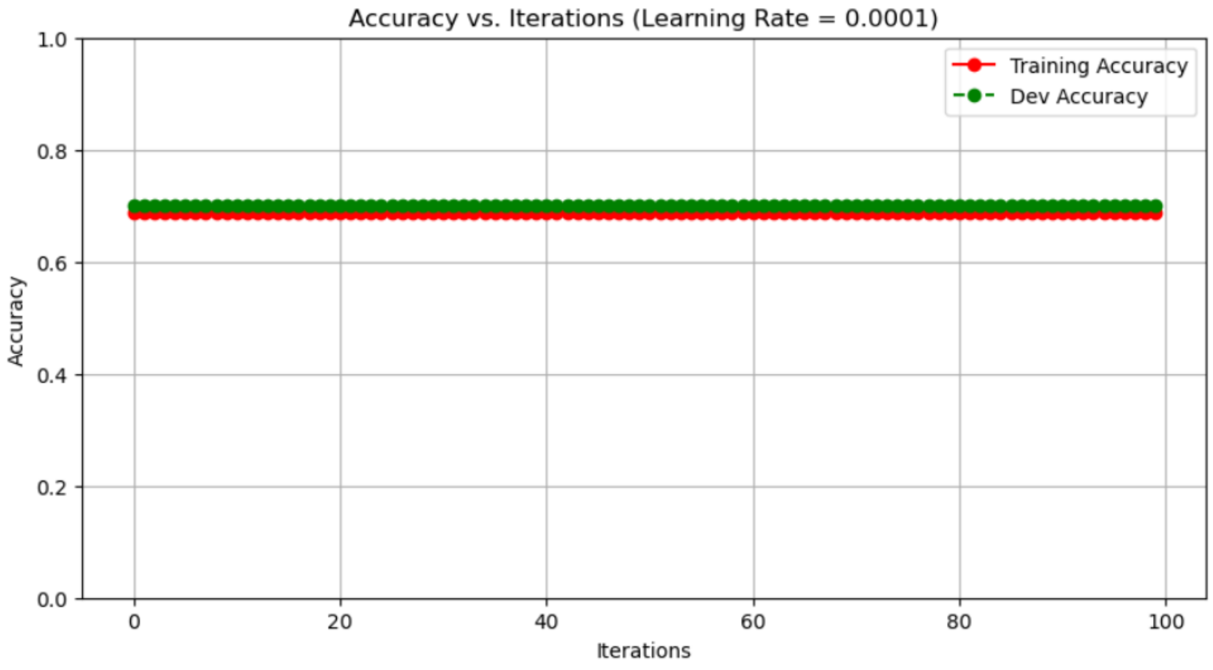
```
Training with learning rate: 0.1
```



```
Training with learning rate: 0.001
```



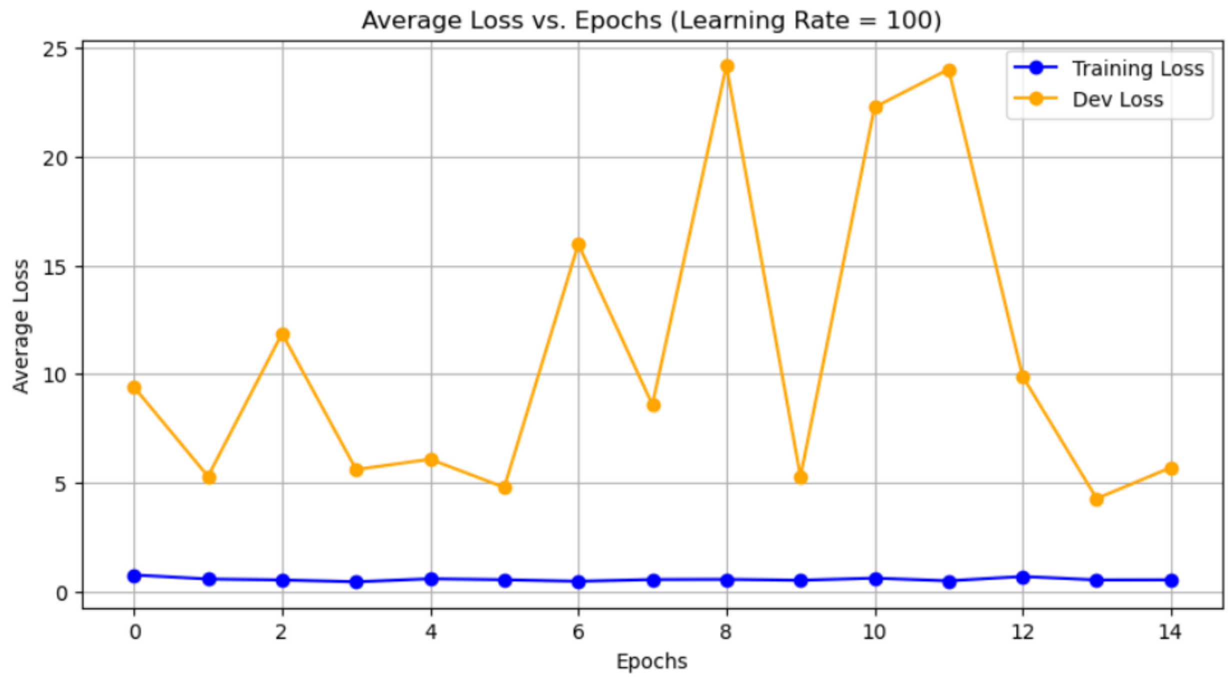
```
Training with learning rate: 0.0001
```



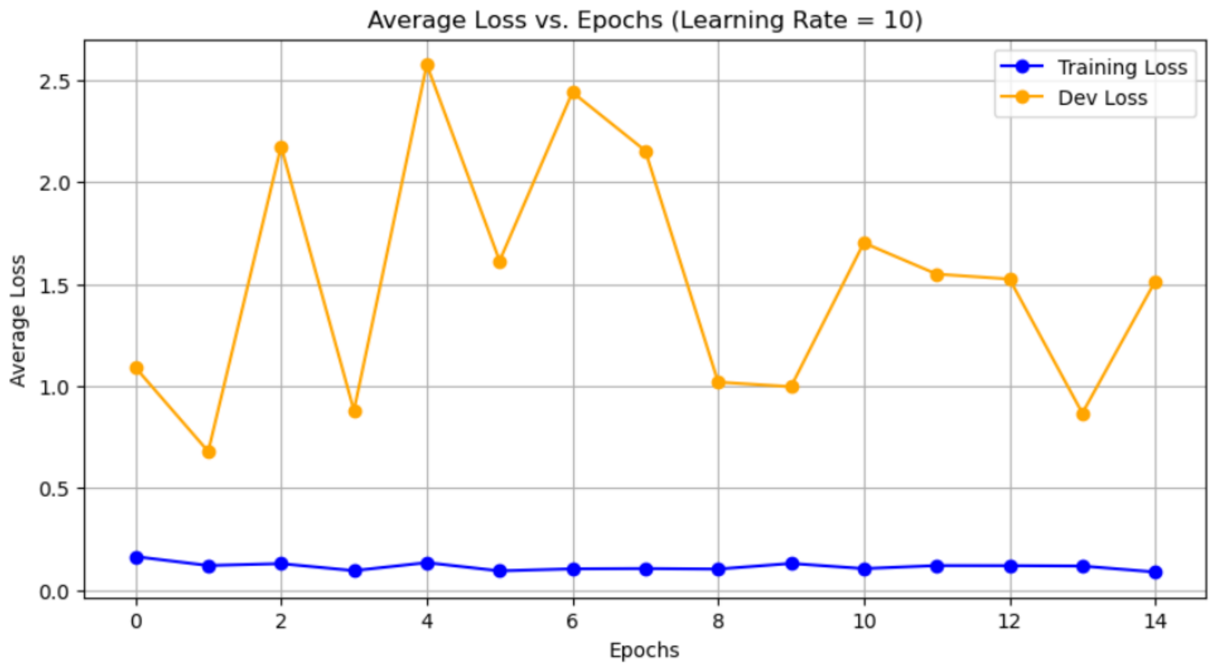
Q3: Loss vs. Epochs for Different Learning Rates

The average loss during training and validation was plotted for learning rates of 100, 10, and 0.1. This shows how the model converges and how quickly the loss decreases over time.

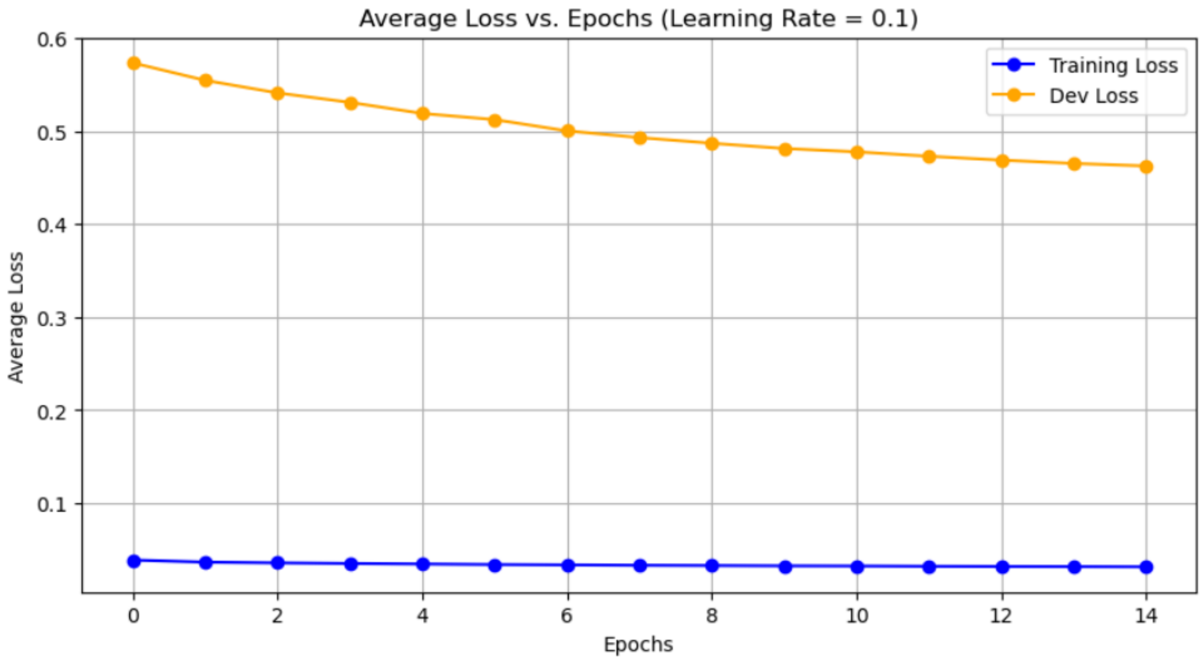
```
Training with learning rate: 100
```



```
Training with learning rate: 10
```

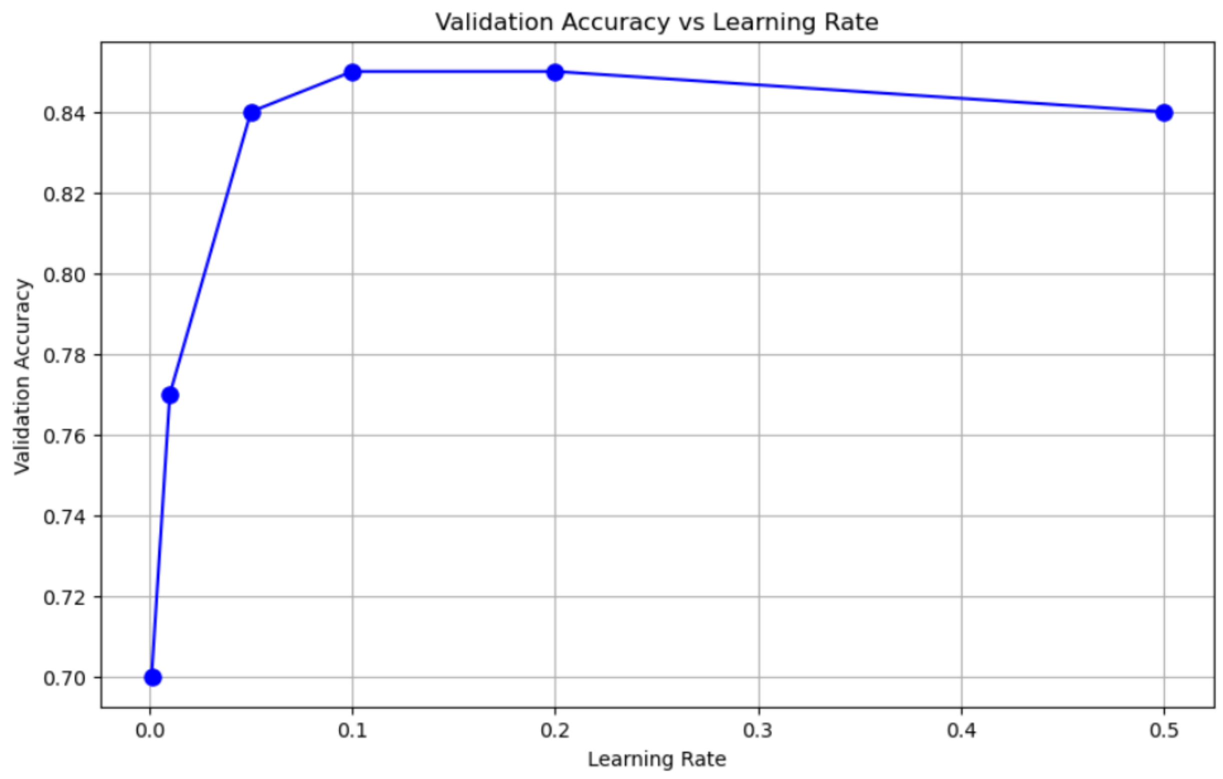


```
Training with learning rate: 0.1
```



Q4: Validation Accuracy vs. Learning Rate

The final validation accuracies were plotted against different learning rates. This helps to visualize the relationship between learning rate and model performance.



```
The best learning rate is: 0.1 with validation accuracy: 0.8500
```

Q5: Test Accuracy for Best Learning Rate

```
The testing accuracy corresponding to the best learning rate (0.1) is: 0.8200
```

Q6: Try Different Batch Sizes (4, 8, 16, 32, 64) and Plot Validation Accuracy

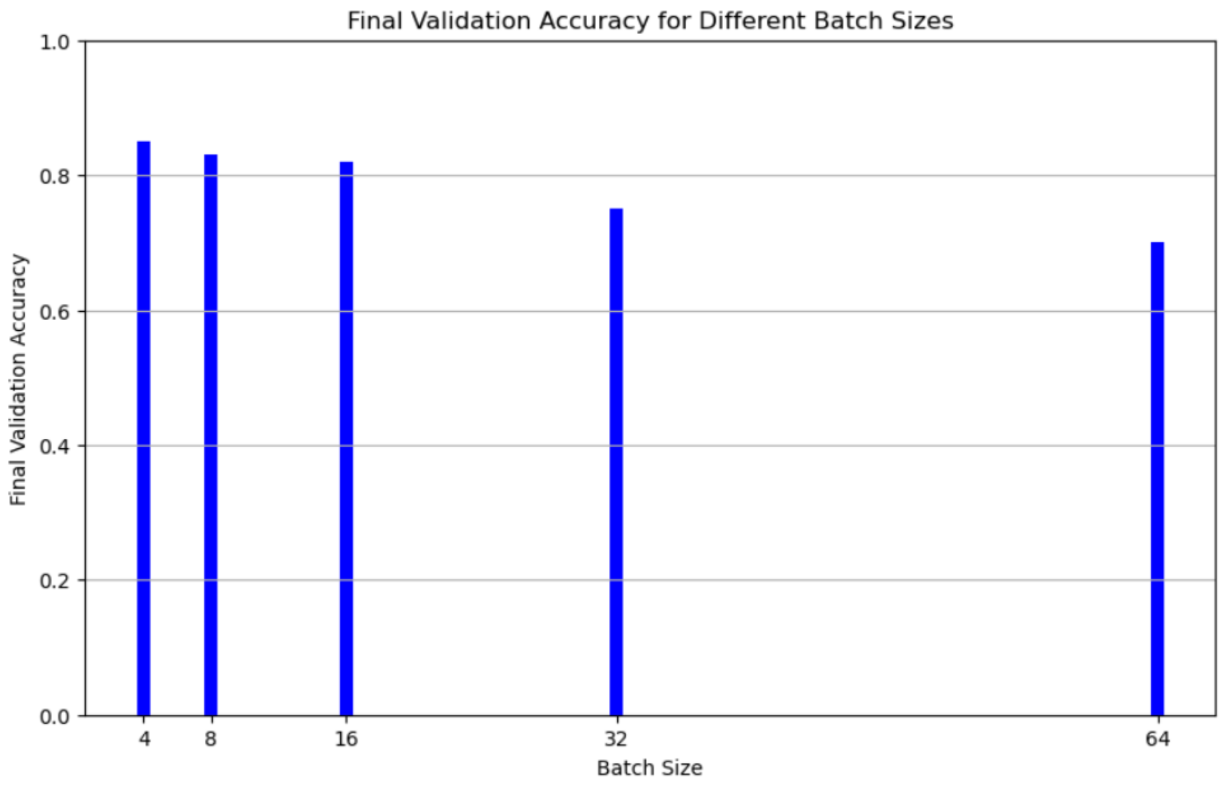
In this part of the report, we explore how different batch sizes affect the performance of the logistic regression model during training. We tested batch sizes of 4, 8, 16, 32, and 64 and plotted the final validation accuracy for each batch size.

Methodology

- The model was trained with a fixed learning rate of 0.01 and run for 100 iterations.
- We used the mini-batch gradient descent algorithm with the batch sizes listed above. The batch size refers to the number of samples used in each iteration to update the model's parameters.
- For each batch size, the final validation accuracy (after 100 iterations) was recorded and plotted.

Analysis

- The model's validation accuracy improves as the batch size increases from 4 to 32.
- The **best performance** was achieved with a **batch size of 4**, which produced a final validation accuracy of **`0.8500`** .
- This suggests that increasing the batch size might help improve the model's performance up to a point, beyond which the improvements diminish, and larger batch sizes may not yield significantly better results.



```
The best batch size is: 4 with validation accuracy: 0.8500
```

Q7: Add L2 Regularization to the Implementation of Gradient Descent

In this part of the report, we add L2 regularization to the logistic regression model's gradient descent optimization process. Regularization helps prevent overfitting by penalizing large weights. We specifically explore the effect of different values of the regularization strength, denoted by λ\lambdaλ.

Implementation of L2 Regularization

To implement L2 regularization, the following changes were made:

1. **Regularization Term**: We added an L2 penalty term to the loss function, which is calculated as:

$$\text{Regularization Term} = \frac{\lambda}{2N} \sum_{i=1}^{M} w_i^2$$

where wiw_iwi are the model weights and NNN is the number of training samples. This term is added to the logistic loss function to get the total loss.

2. **Gradient Update with Regularization**: The gradients for the weights were updated by including the regularization term:

$$\frac{\partial L}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} (y_{\text{pred},i} - y_i) x_{ij} + \frac{\lambda}{N} w_j$$

where λ\lambdaλ is the regularization strength, and the bias term was updated without any regularization.

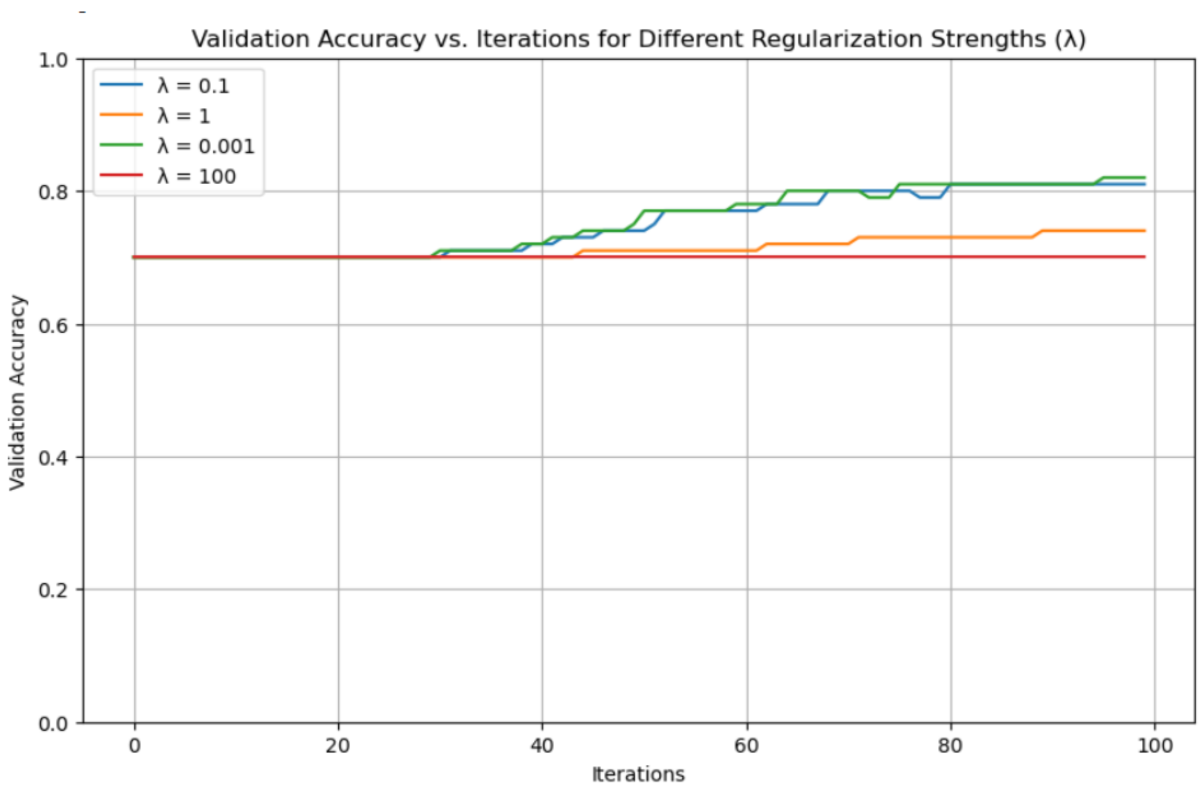
Experimentation with Different Values of λ\lambdaλ

We tested the model with several values of λ\lambdaλ to observe its effect on the validation and test accuracies. The values of λ\lambdaλ used were: λ=0.1,1,0.001,100\lambda = 0.1, 1, 0.001, 100λ=0.1,1,0.001,100. The model was trained with a batch size of 16 and a learning rate of 0.01, which were found to be optimal in previous steps.

Results

The final validation and test accuracies for different values of $\lambda | lambda$λ were recorded. The following graph plots the validation accuracy over iterations for each regularization strength:

Plot: Validation Accuracy vs. Iterations for Different $\lambda | lambda$λ Values



Analysis

1. **Effect of Regularization**:
    a. As $\lambda | lambda$λ increases, the regularization term has a larger impact on the model. Regularization helps reduce overfitting and forces the model to learn smaller weights.
    b. **For $\lambda{=}0.1 | lambda = 0.1$λ=0.1** and **$\lambda{=}1 | lambda = 1$λ=1**, the validation accuracy was fairly stable and showed improvement over time.
    c. **For $\lambda{=}100 | lambda = 100$λ=100**, the model's accuracy started to drop significantly, indicating that the regularization strength was too high and had a negative impact on the model's performance.
    d. **For $\lambda{=}0.001 | lambda = 0.001$λ=0.001**, the model showed a slight increase in performance, though not as much as for $\lambda{=}0.1 | lambda = 0.1$λ=0.1.
2. **Best Validation Accuracy**:

    a. The **best validation accuracy** was achieved with **$\lambda=0.1 \backslash lambda = 0.1$λ=0.1**, which provides a balance between regularization and model performance.
    b. The **test accuracy** also showed the best results for $\lambda=0.1 \backslash lambda = 0.1$λ=0.1, confirming that moderate regularization helped generalize the model better.

Conclusion

1. **What was added to the implementation**:
    a. We added an L2 regularization term to the loss function, penalizing large weights to prevent overfitting.
2. **Effect of L2 Regularization**:
    a. Adding L2 regularization improved the generalization of the model by preventing overfitting, especially with a moderate regularization strength ($\lambda=0.1 \backslash lambda = 0.1$λ=0.1).
    b. Higher regularization values, like $\lambda=100 \backslash lambda = 100$λ=100, led to poor performance, likely due to overly penalizing the weights and limiting the model's ability to fit the data.
3. **Chosen Regularization Strength**:
    a. **$\lambda=0.1 \backslash lambda = 0.1$λ=0.1** was chosen as the best value, as it gave the highest validation and test accuracies. This value offers a good balance between preventing overfitting and allowing the model to learn effectively from the data.
4. **Recommendation**:
    a. Regularization is highly recommended, especially for models at risk of overfitting. However, the strength of the regularization ($\lambda \backslash lambda$λ) should be chosen carefully, with too high a value potentially harming performance. Based on these results, a moderate $\lambda=0.1 \backslash lambda = 0.1$λ=0.1 was optimal.

Final Recommendation

- **Best $\lambda \backslash lambda$λ value**: **$\lambda=0.1 \backslash lambda = 0.1$λ=0.1** is recommended for this dataset and model.

4o mini

Q8: Training Without Data Normalization

In this part, we train a logistic regression model without normalizing the input data. Data normalization (or standardization) is commonly applied to machine learning models to improve their convergence rate and prevent issues caused by large differences in feature magnitudes. However, we wanted to investigate the impact of training without normalization on model performance and training ease.

Training Without Normalization

We used the same dataset, but the training, validation, and test sets were not normalized. The three learning rates chosen for this experiment were:

- **Learning Rates**: 0.1, 1, 10.

The **mini-batch gradient descent** method was employed for training, with a batch size of 8 and 100 iterations. We monitored the **training accuracy**, **validation (dev) accuracy**, and **test accuracy** for each learning rate, and plotted the results to observe how the model performed without normalization.

Results and Observations

1. **Overflow Issues**:
    a. **Learning Rate 10**: The model experienced **overflow errors** due to the excessively large learning rate, causing weights to grow uncontrollably. This resulted in **"Overflow"** in the output for training, validation, and test accuracy.
    b. **Learning Rate 1**: While the model trained without overflow errors, the convergence was slow and the accuracy improvement was gradual.
    c. **Learning Rate 0.1**: This learning rate worked the best, providing smoother convergence and more stable results for both training and validation accuracy.
2. **Training Difficulty**:
    a. **Learning Rate 0.1**: Training was **relatively smooth** without overflow errors, and the model's accuracy gradually improved as iterations progressed.
    b. **Learning Rate 1**: Training was **slower** but stable. The model was able to learn, but it took longer to converge.
    c. **Learning Rate 10**: The **model diverged** due to the large learning rate, and it was unable to converge to an optimal solution.

3. **Number of Iterations**:
    a. The model did not converge as easily with unnormalized data, especially with higher learning rates. The **learning rate of 0.1** was able to train the model without errors and yielded the best validation and test accuracies.
    b. Higher learning rates (1 and 10) required more careful tuning and eventually caused divergence.

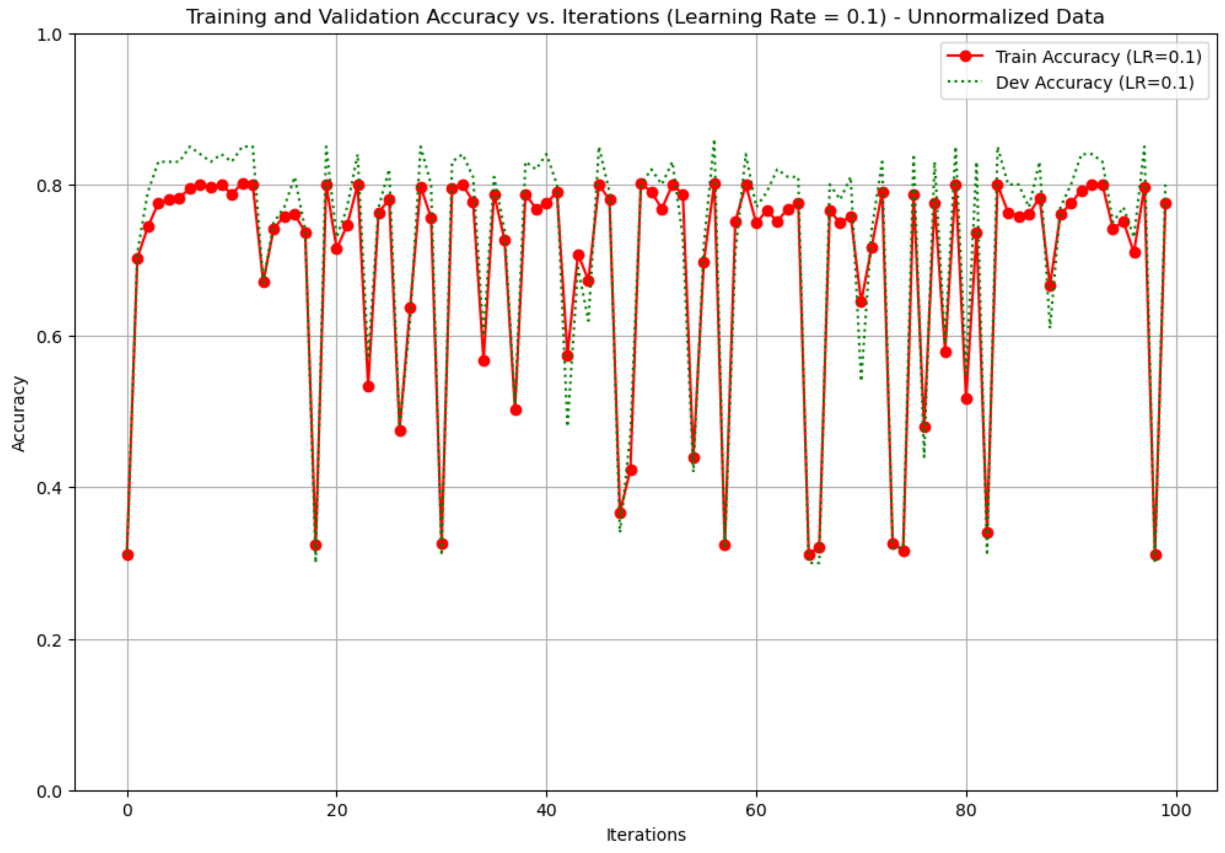
Plotting Accuracy vs. Iterations

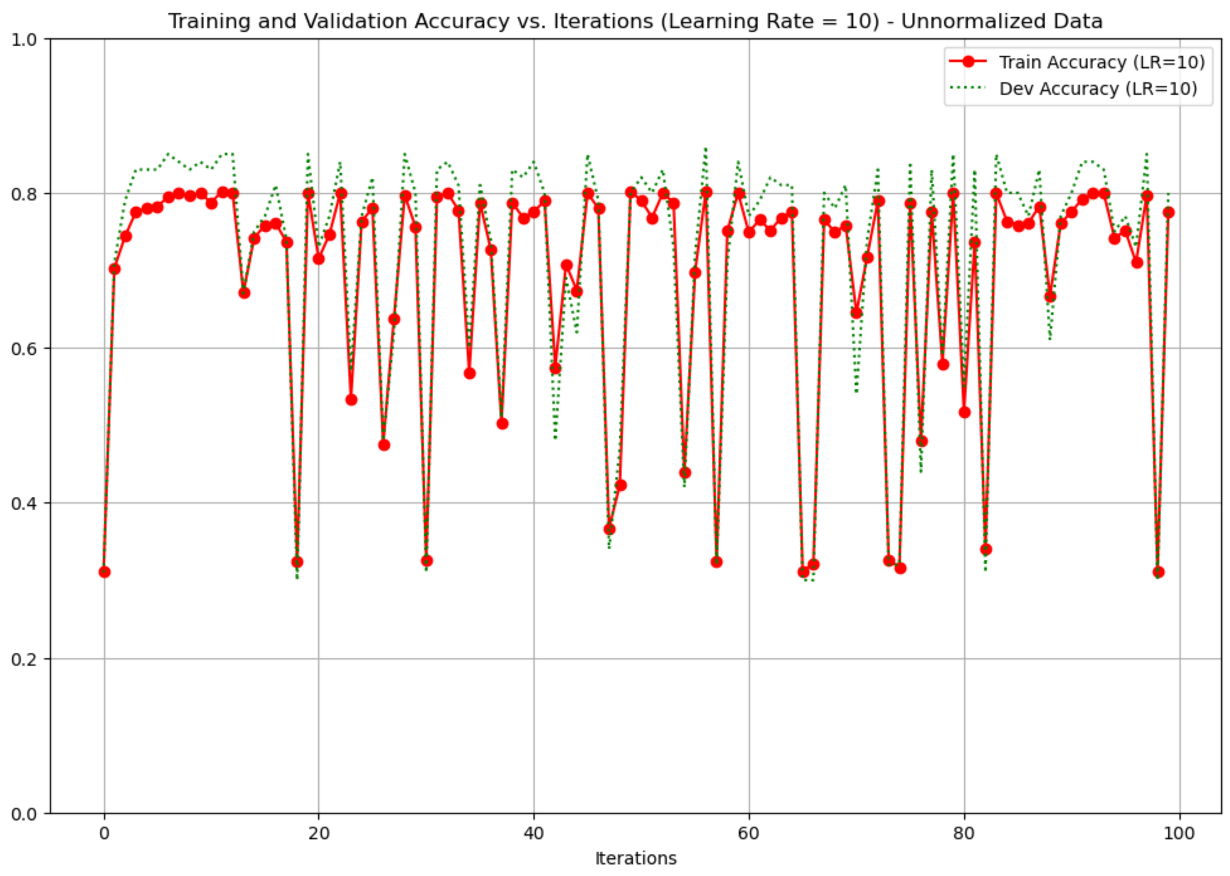
The following plots show how training and validation accuracy evolved during training for each learning rate:

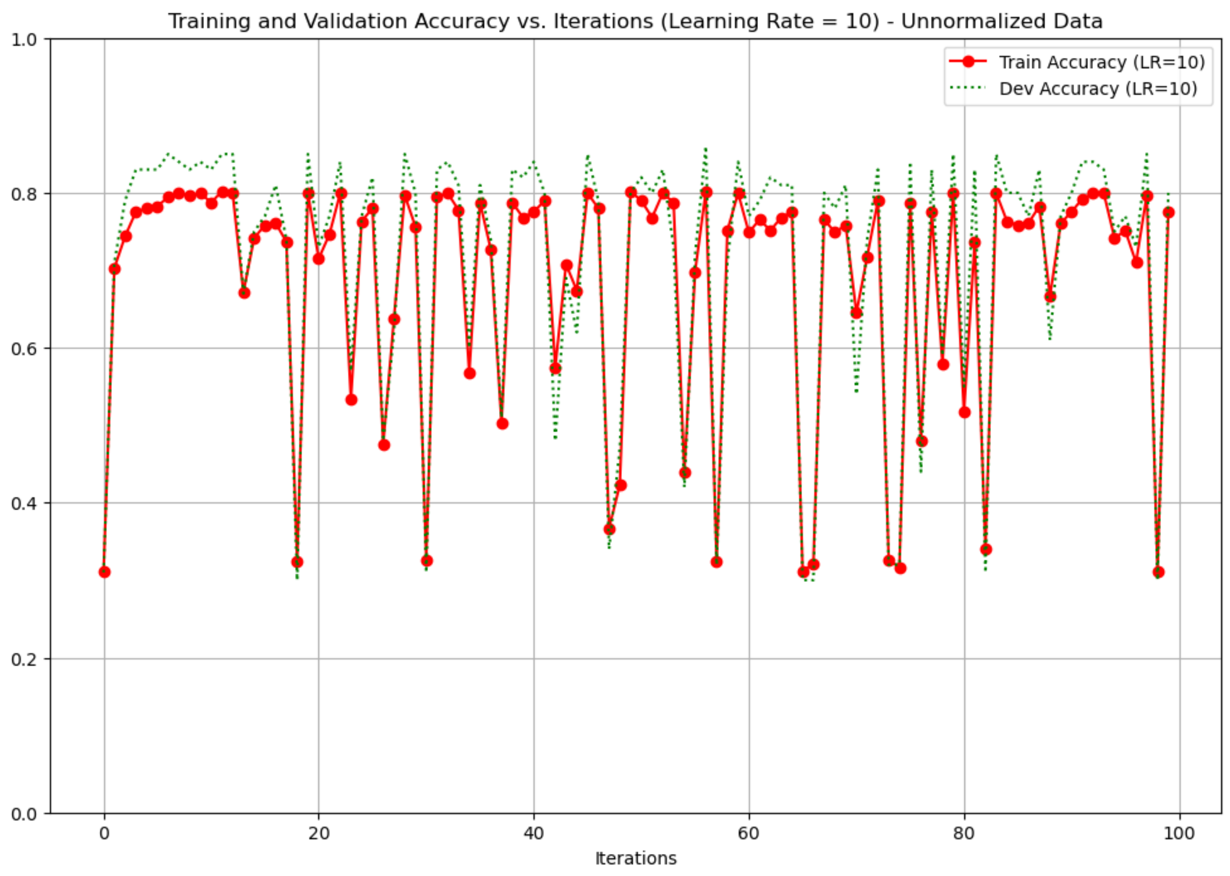
```
Results for Learning Rate 0.1:
{'Train Accuracy': 0.7971014492753623, 'Dev Accuracy': 0.84, 'Test Accuracy': 0.82}

Results for Learning Rate 1:
{'Train Accuracy': 0.782608695652174, 'Dev Accuracy': 0.79, 'Test Accuracy': 0.77}

Results for Learning Rate 10:
{'Train Accuracy': 0.7753623188405797, 'Dev Accuracy': 0.8, 'Test Accuracy': 0.76}
```




Training and Validation Accuracy vs. Iterations (Learning Rate = 10) - Unnormalized Data
Train Accuracy (LR=10)
Dev Accuracy (LR=10)
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Iterations



Conclusion

1. **Effectiveness Without Normalization**:
    a. Training without normalization led to slower convergence and potential overflow issues with higher learning rates (such as 10). The model did not perform as well compared to training with normalized data.
2. **Best Learning Rate**:
    a. **Learning rate of 0.1** was the most effective, as it provided the best convergence without overflow errors.
    b. **Learning rate of 1** showed slower learning but was still usable, whereas **learning rate of 10** caused the model to diverge due to overflow errors.
3. **Training Difficulty**:
    a. Without normalization, the model required more careful tuning of the learning rate to avoid divergence and achieve satisfactory performance. Normalization could have helped accelerate convergence and improved model stability.

Recommendation

- **Normalization** is highly recommended for training models, especially when dealing with features that vary in scale. It helps the model converge faster and avoids issues like overflow during gradient descent, particularly for high learning rates.