Assignment 3: Mini-ML Library and Autograd Engine

Vishnupraneeswar P (co23btech11024)

1 Problem 1: Mini ML Library

For this problem, I created the Python package structure for `my_ml_lib` as specified. All required modules, including preprocessing transformers (`PolynomialFeatures`, `GaussianBasisFeatures`, `StandardScaler`) and dataset loaders (`load_spambase`, `load_fashion_mnist`), were implemented. All classes follow the `scikit-learn` interface (`fit`, `transform`, `predict`) as required. The complete library is included in the `.zip` submission.

2 Problem 2: Logistic Regression (IRLS)

Here, I implemented L2-Regularized Logistic Regression using the Iterative Reweighted Least Squares (IRLS) algorithm. The `LogisticRegression` class was created, following the `scikit-learn` interface (`__init__`, `fit`, `predict`, `predict_proba`).

3 Problem 3: Spam Classification Experiment

In this experiment, I analysed the impact of feature scaling on the performance of the L2-Regularized Logistic Regression model.

3.1 Hyperparameter Tuning Results

I performed a 5-fold cross-validation on the training set to find the optimal L2 regularization strength `alpha` for both raw and standardized data. The values tested were $\alpha \in \{0.01, 0.1, 1, 10, 100\}$.

- **For Raw Data:** The best `alpha` found was **0.1**, with a mean cross-validation error of 7.20%.
- **For Standardized Data:** The best `alpha` was **0.01**, with a mean cross-validation error of 7.39%.

3.2 Final Model Performance

Using the best `alpha` from cross-validation, the `LogisticRegression` model was trained on the full 80% training set and evaluated on the 20% test set.

3.3 Analysis

Observation on Standardization: The results in Table show the impact from standardization. Clearly Scaled data has better final test performance with its optimal $\alpha = 0.1$.

Table 1: Final Model Error (Best α from CV)

Preprocessing	Best Alpha	Train Error (%)	Test Error (%)
Raw Data	0.1	6.68%	8.94%
Standardized Data	0.01	6.54%	8.59%

4 Problem 4: The Autograd Engine & Visualization

This included the implementation of backward methods for Value class, as well as nn.Module subclasses for Linear, ReLU, Sigmoid, Sequential, CrossEntropyLoss, and the SGD optimizer.

4.1 Computation Graph Visualization

To check the autograd engine, I visualized a simple MLP (Sequential(Linear(2, 3), ReLU())) followed by .sum() operation. get_all_nodes_and_edges function was used to build the node and edge sets, which were then plotted.

4.2 Graph Explanation

The generated graph shows the complete forward and backward computation of a simple two-layer MLP defined as Sequential(Linear(2, 3), ReLU()), followed by .sum() operation that gives a single scalar loss.

At the top of the graph are the basic components, which include x (shape (1, 2)), weight (shape (2, 3)), and bias (shape (3,)). The input x and weight first go through a matrix multiplication (@), giving an intermediate result of shape (1, 3). This is then added to the bias using the addition operation (+), completing the linear transformation (x @ W) + b.

The output from this linear layer (shape (1, 3)) is then passed through a ReLU activation, producing the model's final output (shape (1, 3)). This output is then summed using .sum(), which gives a single scalar value as the loss, shown as the final node at the bottom of the graph.

After calling loss.backward(), the graph also shows how the gradients flow backward through each step of the computation. The gradient values, such as grad = 1.0000 at the final node, can be seen propagating back through the network, confirming that the autograd system is working properly and computing gradients for every operation.

5 Problem 5: Capstone Showdown Analysis

For this final problem, I ran a full experiment on the Fashion-MNIST dataset to compare five different model architectures from the my_ml_lib library:

1. OvR Logistic Regression
2. Softmax Regression (Raw)

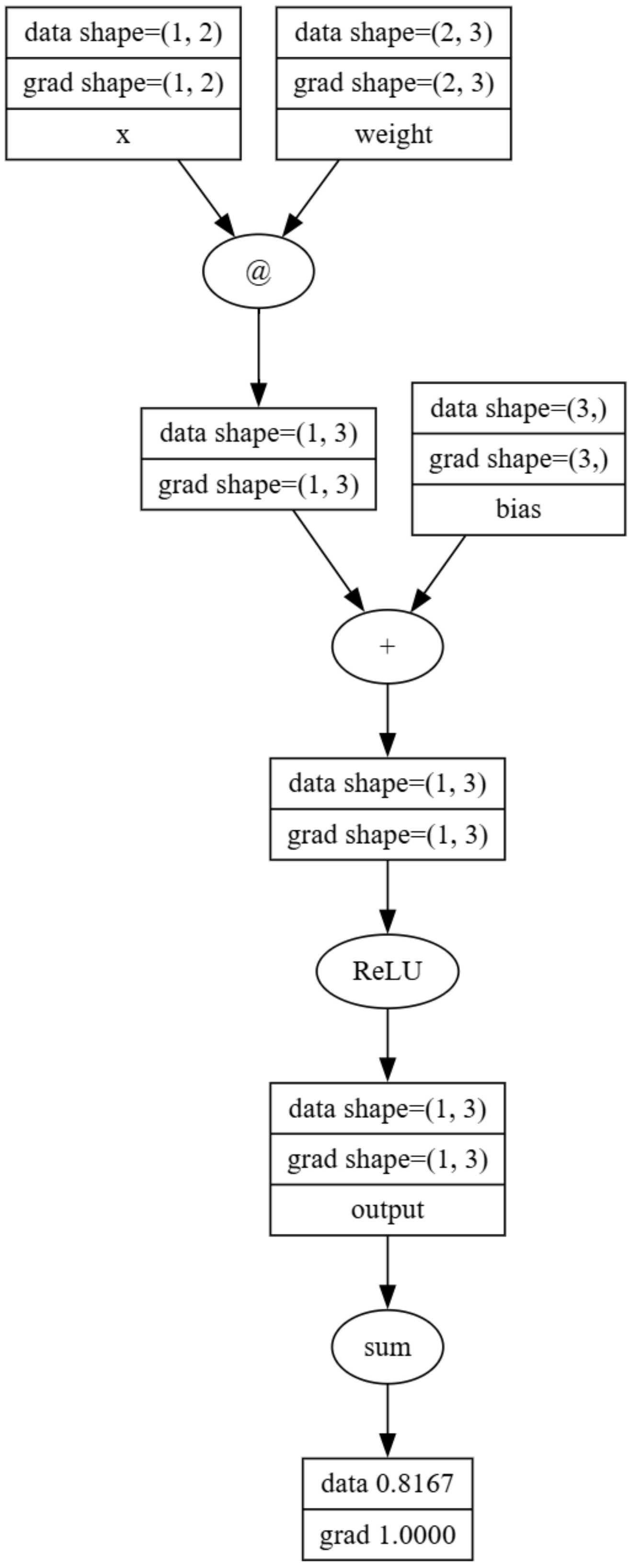

Figure 1: Computation graph for `Sequential(Linear(2, 3), ReLU()).sum()`

3. `Softmax + Gaussian Basis Features`

4. `Multi-Layer Perceptron (MLP)`

5.1 Training Loss Curves

The plot below shows the training loss curves (loss vs. epochs) for the best-performing autograd-based models.

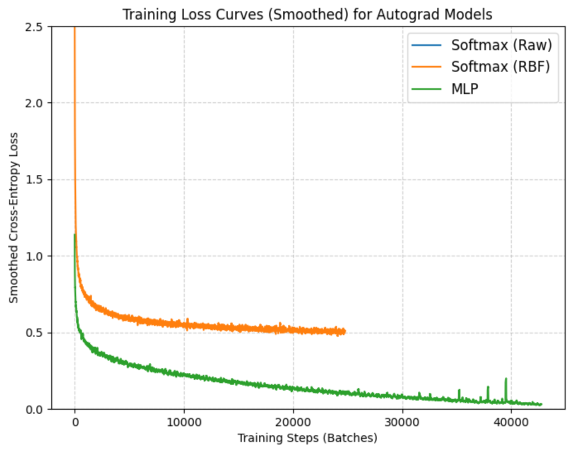

Figure 2: Training loss curves for autograd-based models on Fashion-MNIST.

5.2 Final Performance Results

The table below summarizes the best test accuracy achieved by each of the five model types on the held-out test set, after tuning.

Table 2: Best Test Accuracy on Fashion-MNIST

Model Configuration	**Best Test Accuracy (%)**
OvR Logistic Regression	81.26%
Softmax Regression (Raw)	73.50%
Softmax + Gaussian Basis Features	81.81%
MLP (784-256-128-10)	88.53%

5.3 Discussion

(a) OvR Logistic vs. Softmax (Raw):
The One-vs-Rest (OvR) Logistic Regression and the raw Softmax Regression models showed different generalization capabilities. The OvR model achieved 81.26% test accuracy, while the raw Softmax model performed poorly, achieving only 73.50%. This is interesting because the Softmax model achieved a higher validation accuracy (84.3%) than the OvR model (84.0%). This suggests the Softmax (Raw) model overfit significantly to the validation set. The OvR approach, by breaking the problem into 10 simpler binary classifications, proved to be more robust and generalized.

(b) Effectiveness of Basis Features:
The use of basis features had a dramatic impact. The Softmax + Gaussian Basis Features (RBF) model achieved a test accuracy of 81.81%, which is a significant improvement

over the raw Softmax model's 73.50%. This demonstrates that the raw pixel data is not well-suited for a linear classifier and that transforming the feature space using RBF centers makes the problem much more linearly separable. This RBF-based model's performance was competitive with the OvR Logistic Regression model.

(c) Linear Models vs. MLP:
The Multi-Layer Perceptron (MLP) achieved final test accuracy of 88.53%. This is much better than the best linear model (Softmax + RBF at 81.81%). This result is expected and highlights the ability of deep learning to model complex interactions in the data. RBF model relies on predefined features, the MLP learns its own way to represent the features. As implied by the Universal Approximation Theorem, the hidden layers (784-256-128) learn to extract more complex patterns from the data (from edges to shapes to parts of clothing) making it more suitable for real world classification tasks.

(d) Tuning Process Summary:
For the OvR Logistic model, L2 regularization (alpha) was the key hyperparameter, with a relatively high value of 10.0 providing the best validation result. This indicates that OvR is not able generalize good enough (as it treats 1class vs rest (9 classes)), hence slightly heavier penalty is required. For the autograd models, tuning was an iterative process of monitoring validation accuracy, which guided the number of epochs. The significant gap between validation (84.3%) and test (73.5%) accuracy for the raw Softmax model, highlights its poor generalization. It was also notable that the MLP showed signs of overfitting (96.2% validation vs. 88.53% test) but still performed the best overall, indicating its learned features were enough to overcome overfit to some extent.