1. Title:

ScalarFlow: Implementing Automatic Differentiation

2. Key Idea:

Machine Learning models have forward pass and backward pass, where the forward pass takes inputs, combines weights & biases, and outputs prediction/classification with the help of computational graphs, mathematical operators, and expressions. The loss function tells us whether the output is closer to the ground truth or not, which in turn backpropagates and nudges the gradients of all the inner parameters till the root level weights and biases. This core idea of ML, along with reverse-mode automatic differentiation, chain rule, and gradient descent, is implemented here, some of which I coded and others already existing in the template.

3. Data Need:

The project requires:

* Synthetic datasets for classification tasks, such as the XOR or two-cluster datasets.
* Input features and labels for binary classification problems.
* Randomly initialized weights for the logistic regression and multi-layer perceptron (MLP) models to demonstrate learning through gradient descent.

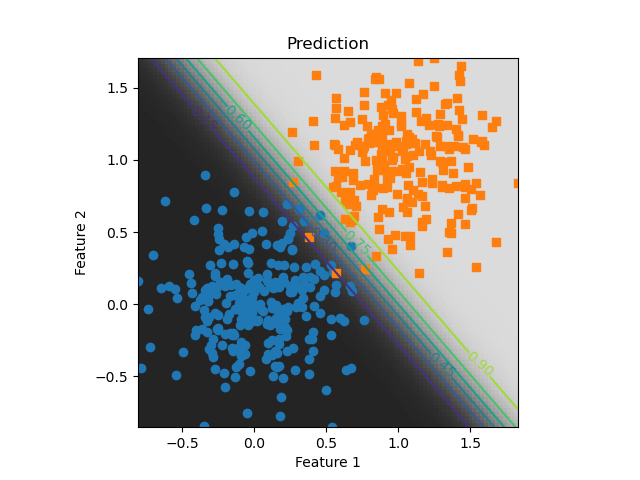
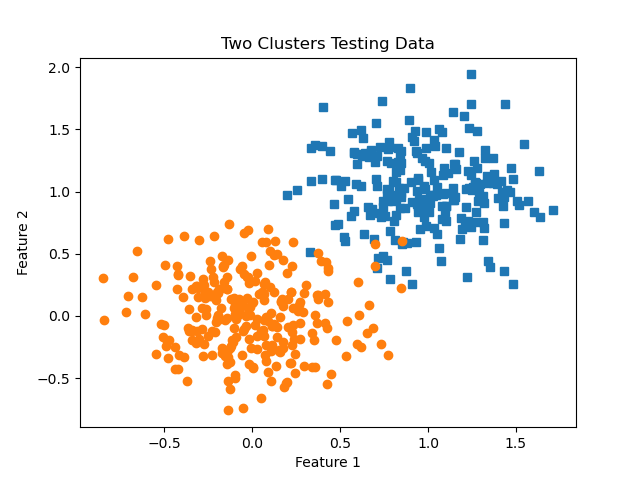
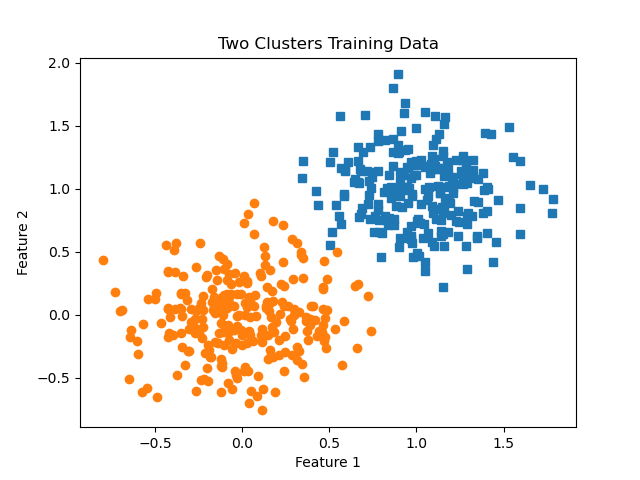
4. Methods:

I implemented computation nodes, such as constants, variables, placeholders, and basic operations (e.g., addition, multiplication, division, exponentiation). Each node supports forward computation (computing its value) and backward computation (computing its derivative for backpropagation). Automatic differentiation was implemented through a directed acyclic graph (DAG), where gradients are calculated in reverse order using backpropagation.

Gradient Descent:

Gradient descent was used for optimizing model weights during training by minimizing the loss function (cross-entropy loss). A feed-forward pass was executed to compute model predictions, followed by a backward pass to compute gradients, which were then used to update the weights.

5. Screenshot:



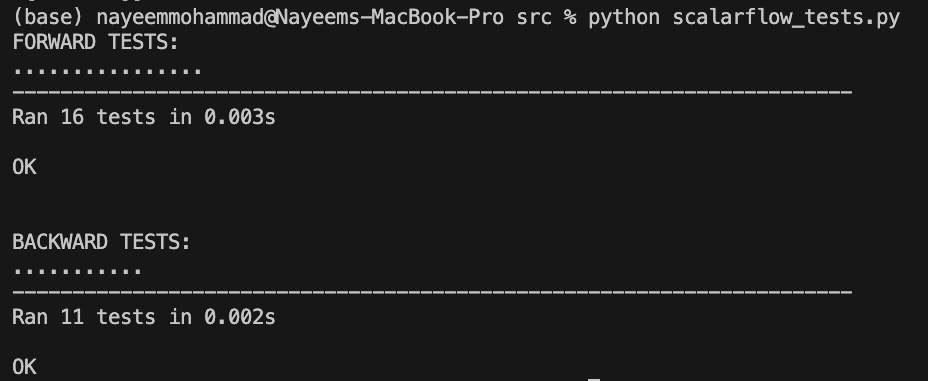
Here is an example of ScalarFlow generating a computation graph and performing gradient descent to minimize the loss in a logistic regression task:

(Insert screenshot showing the terminal output, computation graph, or a training plot)

6. Evaluation:

This project has two evaluation files (scalarflow\_tests.py and sf\_classifier\_examples.py). The first file checks whether all the implementations of add, subtract, multiply, divide, log, etc., are working properly or not. It also checks the backpropagation results by comparing the derivatives of custom inputs. The output is given below:

python scalarflow\_tests.py



The second evaluation generates a synthetic two-cluster dataset containing 500 data points for training and testing separately. The model is trained on the synthetic data, and the training loss is present below where it decreases gradually and achieves an accuracy of Accuracy: 0.98800. Also, the confusion matrix of the classification test result is given as output.

python sf\_classifier\_examples.py

Epoch 0 loss: 157.46204469096585

Epoch 1 loss: 73.17578943740041

Epoch 2 loss: 53.44396043135303

Epoch 3 loss: 43.96066106485995

Epoch 4 loss: 38.33961422908067

Epoch 5 loss: 34.48238332626951

Epoch 6 loss: 31.694944927285313

Epoch 7 loss: 29.55911500384169

Epoch 8 loss: 27.836991220390725

Epoch 9 loss: 26.477637861502924

Epoch 10 loss: 25.28653630187271

Epoch 11 loss: 24.292169374139586

Epoch 12 loss: 23.46761505331837

Epoch 13 loss: 22.664030886897365

Epoch 14 loss: 22.058710157781242

Epoch 15 loss: 21.4745134259346

Epoch 16 loss: 20.954119133983397

Epoch 17 loss: 20.47745565322942

Epoch 18 loss: 20.05365583959138

Epoch 19 loss: 19.664167345775663

Epoch 20 loss: 19.29866444835023

Epoch 21 loss: 18.965188684769547

Epoch 22 loss: 18.662595906640572

Epoch 23 loss: 18.360892669500473

Epoch 24 loss: 18.111028660751188

Epoch 25 loss: 17.86080491958305

Epoch 26 loss: 17.642842281017806

Epoch 27 loss: 17.435186362387235

Epoch 28 loss: 17.22757215798569

Epoch 29 loss: 17.039920959834408

Epoch 30 loss: 16.85343069652421

Epoch 31 loss: 16.666158863370853

Epoch 32 loss: 16.53313704257978

Epoch 33 loss: 16.36639443149953

Epoch 34 loss: 16.221688645068603

Epoch 35 loss: 16.0869314384128

Epoch 36 loss: 15.94086363669948

Epoch 37 loss: 15.841257278883427

Epoch 38 loss: 15.720846374725996

Epoch 39 loss: 15.603798050593468

Epoch 40 loss: 15.493128093928739

Epoch 41 loss: 15.387383027276071

Epoch 42 loss: 15.28809673052319

Epoch 43 loss: 15.176502570875643

Epoch 44 loss: 15.100932800281894

Epoch 45 loss: 15.012453603532773

Epoch 46 loss: 14.912027577933603

Epoch 47 loss: 14.821312107872915

Epoch 48 loss: 14.756896375351168

Epoch 49 loss: 14.674226641140551

Epoch 50 loss: 14.586522629514034

Epoch 51 loss: 14.53283775486413

Epoch 52 loss: 14.450696197696526

Epoch 53 loss: 14.395180975795592

Epoch 54 loss: 14.327510755125015

Epoch 55 loss: 14.279148587242585

Epoch 56 loss: 14.212956686521252

Epoch 57 loss: 14.15606357338339

Epoch 58 loss: 14.096487614249654

Epoch 59 loss: 14.02423634328598

Epoch 60 loss: 13.984120700488875

Epoch 61 loss: 13.935957597331852

Epoch 62 loss: 13.883973580658255

Epoch 63 loss: 13.83367254726498

Epoch 64 loss: 13.785184323402284

Epoch 65 loss: 13.733575284787213

Epoch 66 loss: 13.681333338203746

Epoch 67 loss: 13.635742900407163

Epoch 68 loss: 13.603398325402653

Epoch 69 loss: 13.554221076453704

Epoch 70 loss: 13.514637965587683

Epoch 71 loss: 13.470783025660237

Epoch 72 loss: 13.445958342930371

Epoch 73 loss: 13.38224597608503

Epoch 74 loss: 13.358627822396992

Epoch 75 loss: 13.329904787425999

Epoch 76 loss: 13.289227944691003

Epoch 77 loss: 13.260771612469249

Epoch 78 loss: 13.224282895493012

Epoch 79 loss: 13.195753303356977

Epoch 80 loss: 13.157604234988996

Epoch 81 loss: 13.126053731552794

Epoch 82 loss: 13.078430123877132

Epoch 83 loss: 13.062898183315745

Epoch 84 loss: 13.039269429703637

Epoch 85 loss: 13.011188791165893

Epoch 86 loss: 12.966584902314743

Epoch 87 loss: 12.955253838541738

Epoch 88 loss: 12.922589689232458

Epoch 89 loss: 12.898447741858352

Epoch 90 loss: 12.87490568581963

Epoch 91 loss: 12.844672692603696

Epoch 92 loss: 12.824964745149066

Epoch 93 loss: 12.792847855463396

Epoch 94 loss: 12.769689899864042

Epoch 95 loss: 12.752470479392471

Epoch 96 loss: 12.712193054183365

Epoch 97 loss: 12.698787621673215

Epoch 98 loss: 12.680815814084012

Epoch 99 loss: 12.651013623723713

Accuracy: 0.98800

Confusion Matrix

[[257 2]

[ 4 237]]

For simpler datasets like the two-cluster dataset, the model achieved good performance after training for a sufficient number of epochs.

Gradient Descent Behavior: I observed how the loss decreased over time throughout the training, confirming that the gradients and backpropagation were working correctly. In certain cases, bugs were encountered in the derivative calculations, such as incorrect derivatives for operations like logarithms and subtraction, which were fixed.

Challenges: A key challenge was implementing correct gradients for more complex operations like cross-entropy loss. Debugging backward computation steps and ensuring the accuracy of the gradient calculations required thorough testing.

7. Experience:

I gained a deep understanding of the following concepts:

I learned how to implement forward and backward passes using computation graphs, where nodes represent mathematical operations. Understanding how gradients are propagated backward through the graph to compute derivatives for gradient descent was also key learning from this project. Implementing the gradient descent algorithm from scratch and seeing how updating weights step-by-step minimizes the loss function helped understand the complex relationship between loss and optimizations, which is the key to finding hidden patterns in the data.

One of the most valuable experiences was debugging various issues, like zero gradients during backpropagation, which helped me better understand the subtleties of derivative computations in neural networks and optimization.