Differential Evolution for Neural Network Weights Optimization

Overview:

This projects implements differential evolution to optimizer weights for neural network weights optimization and compares it to the backpropagation methods.

Neural Network Architecture:

Input layer: 64 unit (8\*8 pixel images flattened)

Hidden layer: 20 unit

Output: 10 (labels from 0 to 9)

Loss function: CrossEntropyLoss

Dataset Used:

MNIST Dataset

Optimizers:

1. Differential Evolution:

Summary:

It works by initializing set of candidate solutions (vectors) randomly, also we are bounding the weights between -1 and 1 for better convergence and performance, and for number of generations every generation it loop on solutions and get three random vectors other than the current vector, we make out of them a trial vector where we cross over with the current vector.

Then we select the one with better fitness to keep. After all generations best solution is chosen.

Parameters used:

* Population size (NP): 50
* Mutation Factor (F): 0.5
* Crossover Rate (CR): 0.9
* Generations: 1000

2. Stochastic Gradient Descent

Summary:

It's an optimization algorithm that is used to find optimal parameters that minimizes the objective function.

Gradient is vector of all partial derivatives of a function where this vector is the direction of the steepest ascent. We take a step in the opposite direction of it (to minimize loss) and scale by a learning rate.

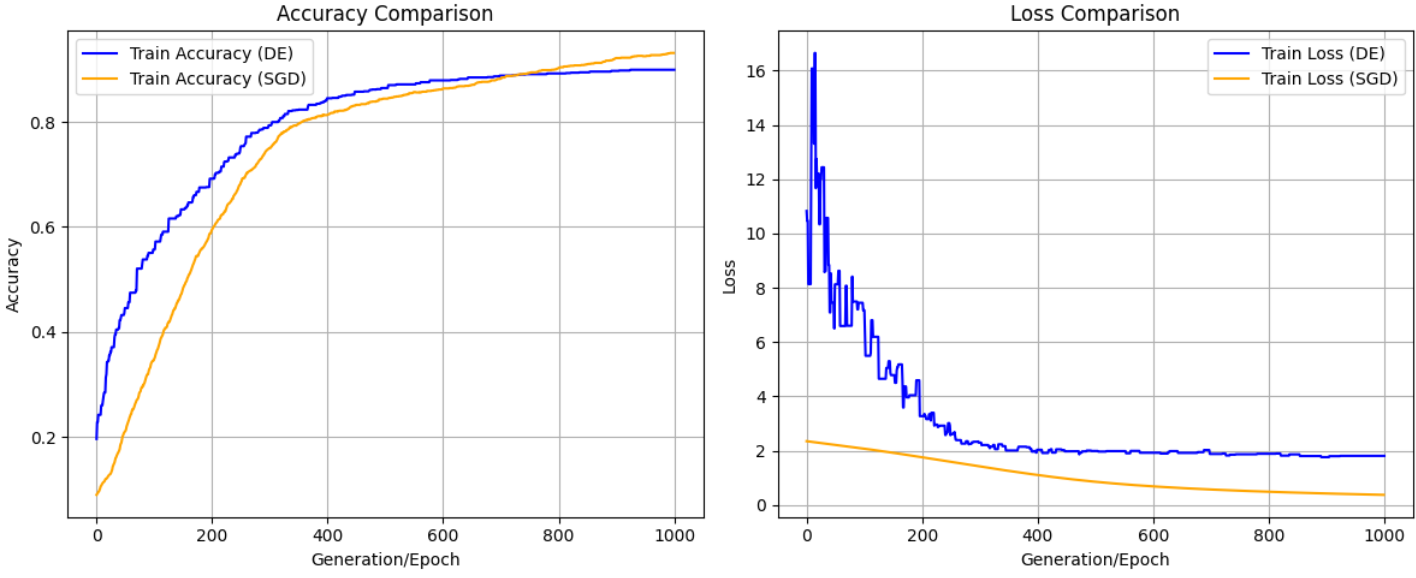
Parameters used:

Epochs: 1000

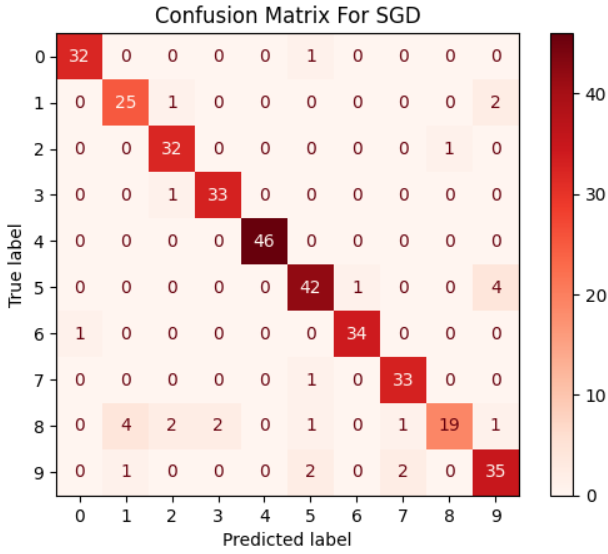
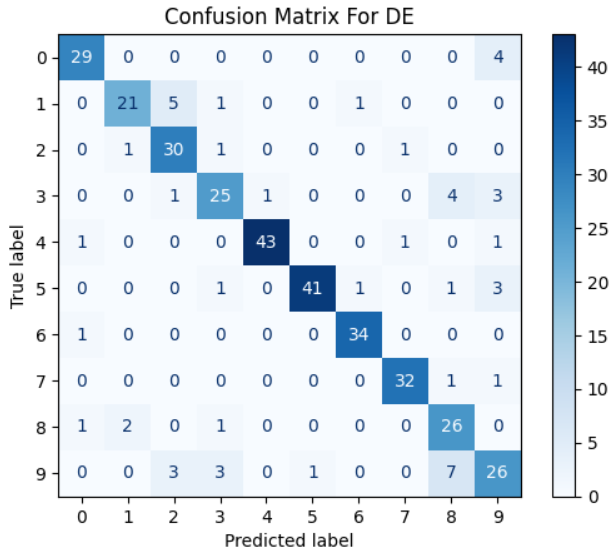
Learning rate (lr): 0.01

Comparison:

|  |  |  |
| --- | --- | --- |
|  | Accuracy | Loss |
| DifferentialEvolution | 0.90 | 1.76 |
| StochasticGradientDescent | 0.93 | 0.37 |



Confusion Matrices:



Summary:

Time for back propagation: 1.05 second

Time for Differential Evolution: 31.8 second

Backpropagation is faster by almost 30x and better by 3% than Differential evolution.