

Neural Evolution for Handwritten Digit Recognition using Differential Evolution (DE) [Preferably in combination with other EAs/SI approaches of your choice]

Project Idea and Problem Definition:

The core idea of this project is to apply **Neural Evolution techniques** to the problem of handwritten digit recognition using the **MNIST** dataset. Instead of training the neural network using traditional backpropagation, we aim to **evolve** its weights and biases using a set of **Evolutionary Algorithms**, including **Differential Evolution (DE)**, **Genetic Algorithm (GA)**, **JADE**, **DE-Island**, **GA-Island**, and **Hybrid approaches**. This allows us to explore complex, non-convex solution spaces more effectively, leveraging the global search capabilities of EAs.

This task is best categorized as a **Free Optimization Problem (FOP)**, where:

* The **goal** is to find the optimal set of weights and biases that maximize the neural network’s classification accuracy,
* The **solution space** is continuous and unconstrained (i.e., no hard restrictions are placed on the parameters),
* The **fitness** is evaluated based on prediction performance on the dataset.

Main Functionalities:

The main functionalities of this project revolve around applying and evaluating different **Evolutionary Algorithms (EAs)** to evolve neural networks for the task of **handwritten digit recognition**. Instead of using traditional gradient-based training methods, our system uses EAs to optimize the weights and biases of the neural network.

Core Functional Components:

1. **Neural Network Representation:**
   * A feedforward neural network is used as the classifier.
   * The network architecture includes an input layer, one or more hidden layers, and an output layer with softmax activation.
2. **Chromosome Encoding:**
   * Each individual (chromosome) in the population represents a complete set of neural network weights and biases flattened into a vector.
3. **Fitness Evaluation:**
   * The fitness of each individual is calculated based on the network's classification accuracy (or loss) on the MNIST dataset.
4. **Evolutionary Algorithms Applied:**
   * we experimented with a wide range of algorithms including:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Differential Evolution (DE)** | Genetic Algorithm (GA) | JADE (an adaptive version of DE) | **Island-based versions** of both DE and GA | A **Hybrid approach** | Lbest\_PSO |

**5. Training and Evolution Process:**

* + Populations evolve over multiple generations using operations like mutation, crossover, and selection.
  + Each evolutionary run aims to find a set of network parameters that maximize classification performance.

**6.Evaluation and Comparison:**

* Each algorithm’s performance is evaluated and compared based on:
  + Classification accuracy on the test dataset
  + Convergence speed
  + Stability of results over multiple runs

**7.Result Visualization:**

* + Accuracy curves, fitness progress, and performance comparisons are visualized to support experimental findings.

Similar applications in the market:

**Google Cloud Vision AI**

* **Algorithms Used:** Deep Learning, Transfer Learning (possibly optimized with Evolutionary Algorithms)
* **Description:** Google Cloud Vision offers a powerful AI service for image recognition tasks, including handwritten digit recognition. While it doesn’t specifically mention DE or GA, Google’s Vision AI uses neural networks, which can be optimized using these evolutionary techniques for improving performance.
* **Relevance:** Their deep learning models can benefit from optimization methods such as DE or GA to enhance their recognition capabilities, especially for complex tasks.
* **Link:** Google Cloud Vision AI

Literature Review:

1. JADE: Adaptive Differential Evolution with Optional External Archive

Link :<https://ieeexplore.ieee.org/document/5208221>

2.The Particle Swarm—Explosion, Stability, and Convergence

3.Island Model based Differential Evolution Algorithm for Neural Network Training

4.Particle Swarm Optimization (original)

**link:** [**https://ieeexplore.ieee.org/document/488968**](https://ieeexplore.ieee.org/document/488968)

5.A Modified Particle Swarm Optimizer (canonical)

**link :** [**(PDF) A Modified Particle Swarm Optimizer**](https://www.researchgate.net/publication/3755900_A_Modified_Particle_Swarm_Optimizer)

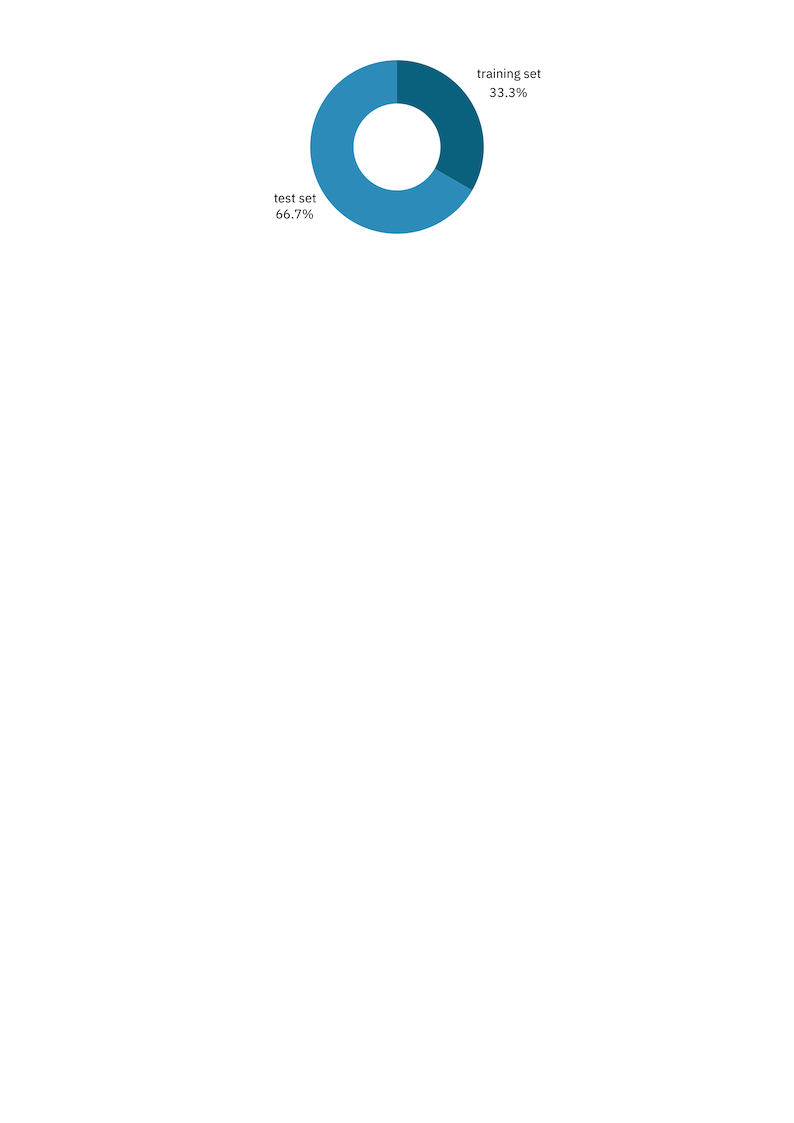
6.Differential Evolution for Neural Networks Optimization(SHADE)

Dataset Used :

In this project, we used the **MNIST (Modified National Institute of Standards and Technology) dataset**, which is a widely recognized benchmark dataset for image classification and machine learning tasks. It contains **grayscale images of handwritten digits** from 0 to 9.

**Dataset Overview:**

* **Original Dataset Size:**
  + **Training Set:** 60,000 images
  + **Test Set:** 10,000 images
* **Subset Used in the Project:**
  + 5,000 images **from the training set**
  + All 10,000 images **from the test set**
* **Image Size:** 28 x 28 pixels (784 input features per image)
* **Format:** Grayscale (pixel values range from 0 to 255)
* **Number of Classes:** 10 (digits 0 through 9)



**Preprocessing Performed:**

To prepare the data for neural network training using evolutionary algorithms, the following preprocessing steps were applied:

|  |  |
| --- | --- |
| de30d156-31e7-4c7d-ac80-348e1b8e0cc1.png | **Reshaping:** Each 28×28 image was converted into a flat vector of 784 values to serve as input to the neural network. |

|  |  |
| --- | --- |
| de30d156-31e7-4c7d-ac80-348e1b8e0cc1.png | **Normalization:** Pixel values were scaled to the range [0, 1] to improve training stability and performance. |

|  |  |
| --- | --- |
| de30d156-31e7-4c7d-ac80-348e1b8e0cc1.png | **One-Hot Encoding:** Target labels were converted into one-hot encoded vectors to enable multi-class classification. |

|  |  |
| --- | --- |
| de30d156-31e7-4c7d-ac80-348e1b8e0cc1.png | **Subset Selection:** A subset of 5,000 samples from the training data was used to reduce computational load, while the full test set was retained for evaluation. |

Neural Network Architecture:

We use a simple 2-layer neural network (one hidden layer) to classify MNIST digits. Instead of training via gradient descent, we evolve the weights using Differential Evolution. Here's how the network is structured and how its operations are computed:

**Architecture Summary:**

* **Input Layer**: 784 neurons (flattened 28x28 image)
* **Hidden Layer**: 32 neurons, activated using ReLU.
* **Output Layer**: 10 neurons, activated using Softmax.

**Key Functions & Equations:**

**1. decode\_vector(vector)**

* + Decodes the flat solution vector into neural network parameters

**2. forward(X, W1, b1, W2, b2)**

* Performs a forward pass through the network:
  + Z1​=X⋅W1​+b1​
  + A1=ReLU(Z1)=max⁡(0,Z1)
  + Z2=A1⋅W2+b2
  + A2​=Softmax(Z2​)

**3. cross\_entropy(preds, targets)**

* Calculates the model loss:
  + L=−(1/n)∑yi⋅log⁡(y^i)

**4. fitness\_function(vector)**

* Evaluates the "fitness" (loss) of a solution vector:
  + Decodes weights from the vector.
  + Performs forward pass on training data.
  + Returns the cross-entropy loss.

*Algorithms and Approaches*

Differential Evolution (DE)

* **Goal:**

Optimize neural network weights by minimizing the **cross-entropy loss** using DE — an Evolutionary Algorithm (EA) that evolves a population of solutions over generations.

* **Algorithm Components:** *(Evolutionary loop for 5000 gen)*

**1. Population Initialization:**

* + Population of size P: each individual is a vector of network weights, initialized using ***He-initialization*** (Smart initialization)*.*
  + Vector size = 25,450 (total parameters in the NN).

**2. Mutation:**

* + For each individual xi​, select three distinct individuals xa,xb​,xc​, and generate a **mutant vector**: *v\_i=x\_a + F ⋅ (x\_b − x\_c)*
  + *F* ∈ ]0,2]: **mutation factor** (0.5)

**3. Crossover:**

* + Mix the parent *x\_i* and mutant *v\_i* to create a **trial vector** *u\_i*​:
  + *u\_i* = 𝑓(𝑀𝑎𝑡ℎ.𝑟𝑎𝑛𝑑𝑜𝑚() > 𝐶𝑅) *v\_i* , otherwise = *x\_i*
  + CR ∈ [0,1]: **crossover rate** (0.7)

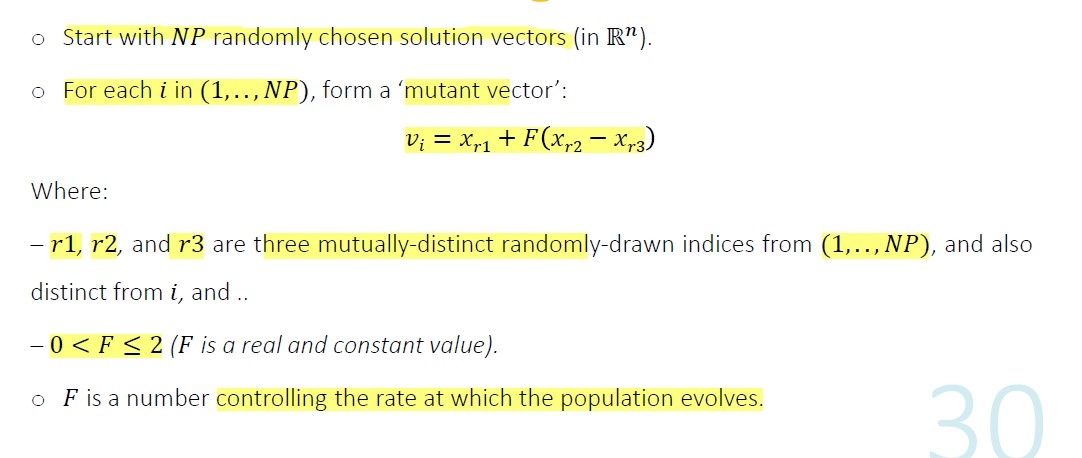
**4. Selection**

* + Evaluate both *x\_i*​ and *u\_i* using the fitness function. Keep the one with **lower loss**:
    - *x(next) = u\_i* , if *f(u\_i​)* ≤ *f(x\_i​)*, otherwise = *x\_i*

* **Fitness Function:**
  + The fitness of each vector is the **cross-entropy loss** computed by:
    - Decoding the vector into NN parameters,
    - Running a forward pass on training data.

* **logging & Results:**
  + Best loss recorded per generation.
  + Logs printed every 100 generations.

* **Pseudocode:**



* **Final Output & Visualization:**
  + *Train accuracy:* **92.62%**
  + *Test accuracy:* **87.79%**

* **Seeds Comparison:** *(Test accuracy and Best Loss)*

|  |  |  |
| --- | --- | --- |
| **Seeds** | **Test Accuracy** | **Best Loss** |
| **10** | **0.8590** | **0.470** |
| **20** | **0.8585** | **0.483** |
| **30** | **0.8590** | **0.4708** |
| **40** | **0.8580** | **0.4580** |
| **50** | **0.8643** | **0.4568** |

Adaptive Differential Evolution (JADE)

* **Goal:**
  + An improved version of Differential Evolution that adapts control parameters (mutation factor & crossover rate) and maintains population diversity using an archive.
  + **JADE** is used as an **optimization algorithm** to **find the best weights and biases** for a neural network classifier **without using traditional training methods** like gradient descent.

* **Algorithm Components:**

**1. Population Initialization:**

* + Population of size P: each individual is a vector of network weights, initialized using ***He-initialization*** (Smart initialization)*.*
  + Fitness is calculated for each individual using the *fitness\_function*.

**2. Mutation Strategy:**

* + Uses **current-to-pbest** mutation
    - *v = x\_i ​+ F ⋅ (p\_best​ − x\_i​) + F ⋅ (x\_r1 ​− x\_r2​)*
  + *p\_best*​: a random individual from the top p% best performers.
  + *x\_r1*, *x\_r2*​: two randomly selected individuals (with *x\_r2x\_{r2}x\_r2*​ possibly from the archive)

**3. Crossover:**

* + **Binomial crossover**: a trial vector *u* is created by mixing the target vector *x\_i* and the mutant vector *v* based on the crossover rate *CR*.

**4. Selection:**

* + If the **trial vector** u has **better or equal fitness** than the current individual *x\_i*​, it **replaces** it in the population.
  + The original vector is stored in the **archive** for future mutation use.

**5. Parameter Adaptation:**

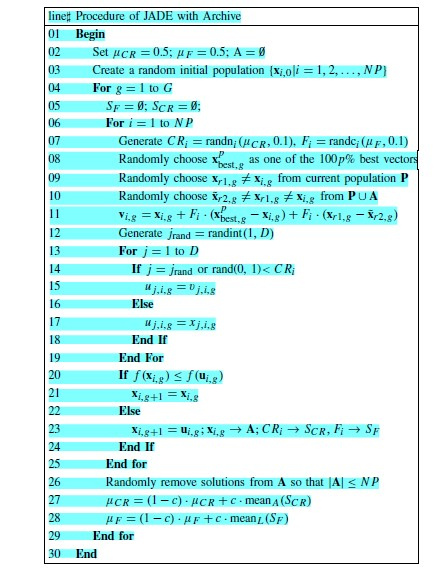
* + The means of F and CR (***mu\_F***, ***mu\_CR***) are updated after each generation using **successful values** from the generation.
  + This helps the algorithm **adaptively learn good parameter settings** over time.

**6. Archive:**

* + Stores replaced individuals to maintain diversity and improve exploration during mutation.

* **logging & Results:**
  + Best loss recorded per generation.
  + Logs printed every 100 generations.

* **Pseudocode:**



* **Final Output & Visualization:**
  + *Train accuracy:* **93.56%**
  + *Test accuracy:* **88.36%**

* **Seeds Comparison:** *(Test accuracy and Best Loss)*

|  |  |  |
| --- | --- | --- |
| **Seeds** | **Test Accuracy** | **Best Loss** |
| **10** | **0.8723** | **0.448** |
| **20** | **0.8771** | **0.419** |
| **30** | **0.8723** | **0.4493** |
| **40** | **0.8809** | **0.3458** |
| **50** | **0.8782** | **0.4166** |

Island Model (DE)

* **Goal:**

To optimize neural network weights using Differential Evolution (DE) with multiple subpopulations (islands) for improved exploration and faster convergence.

* **Algorithm Components:**

**1. Initialization:**

* + Population initialized using ***He-initialization*** and split across islands.

**2. Mutation Strategies:**

* + Each island uses a different DE strategy: *rand/1*, *best/1*, *current-to-best/1*, *rand/2.*

**3. Crossover:**

* + Binomial crossover applied with *CR* = 0.8.

**4. Selection:**

* + Greedy replacement based on fitness.

**5. Migration:**

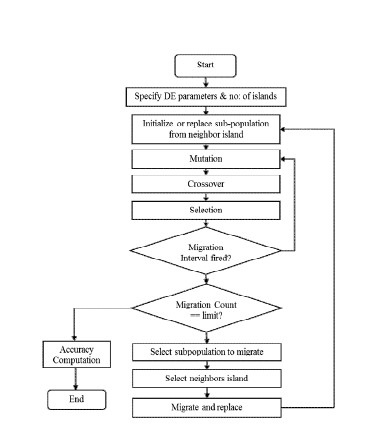
* + Every 50 generations, 2 individuals migrate between neighboring islands (ring topology).

**6. Fitness Evaluation:**

* + Done in parallel using joblib for speed.

* **logging & Results:**
  + Best loss recorded per generation.
  + Logs printed every 100 generations, showing:
    - Global best loss
    - Each island’s best and average fitness

* **Pseudocode:**



* **Final Output & Visualization:**
  + *Train accuracy:* **96.04%**
  + *Test accuracy:* **88.99%**

Genetic Algorithm (GA)

* **Goal:**

Optimize the weights and biases of the neural network to *minimize* the **cross-entropy loss** on the MNIST dataset.

* **Algorithm Components:**

**1. Population Initialization:**

* + Population of size P: each individual is a vector of network weights, initialized using ***He-initialization*** (Smart initialization)*.*
  + Vector size = 25,450 (total parameters in the NN).

**2. Parent Selection:**

* + ***Tournament selection*** is a computationally efficient and scalable method for selecting high-fitness individuals to act as parents for the next generation.
  + **Local Fitness Comparison:**
    - Unlike rank-based or fitness-proportional selection, it only requires comparing individuals within small, randomly sampled groups ("tournaments").
    - No global knowledge of the entire population’s fitness distribution is needed.
  + **Advantages:**
    1. Balances exploration and exploitation: *Larger k increases selection pressure while smaller k maintains diversity.*
    2. Parallel-friendly: *Each tournament is independent, enabling efficient distributed computation.*

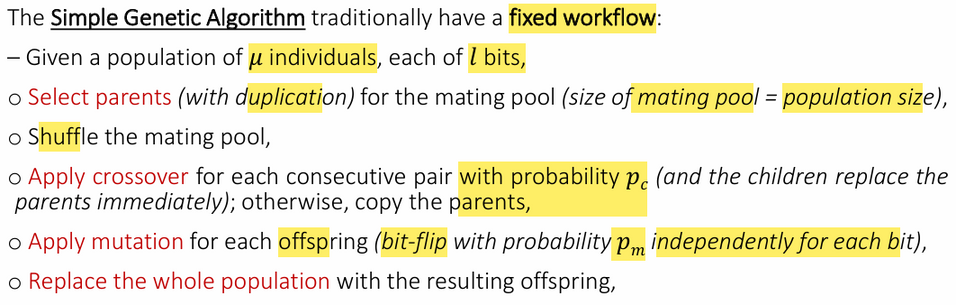
**3. Crossover:**

* + ***BLX-α*** is a recombination operator that creates offspring by blending parameters from two parent solutions within an extended range, promoting diversity while preserving promising traits.

**4. Survivor Selection:**

* + ***Elitism*** is a common survivor selection strategy in evolutionary algorithms (EAs) that ensures the best solutions are preserved across generations.
  + This elitism strategy ensures steady improvement in EAs by safeguarding the best solutions while still allowing evolutionary progress through offspring selection. It’s a simple yet powerful way to enhance algorithm reliability and performance.
* **logging & Results:**
  + Best loss recorded per generation.
  + Logs printed every 100 generations.

* **Pseudocode :**



* **Final Output & Visualization:**
  + *Train accuracy:* **94.92%**
  + *Test accuracy:* **89.03%**

* **Seeds Comparison:** *(Test accuracy and Best Loss)*

|  |  |  |
| --- | --- | --- |
| **Seeds** | **Test Accuracy** | **Best Loss** |
| **10** | **0.8956** | **0.368** |
| **20** | **0.8982** | **0.345** |
| **30** | **0.8907** | **0.3851** |
| **40** | **0.8996** | **0.3458** |
| **50** | **0.8945** | **0.3542** |

Island Model (GA)

* **Goal:**

This Island Model GA implements a parallel evolutionary strategy where multiple subpopulations (islands) evolve independently, with occasional migration between them.

* **Algorithm Components:**

**1. Initialization:**

* + Population initialized using ***He-initialization*** and split across islands.

**2. Parent Selection:**

* + Tournament Selection.

**3. Mutation & Crossover Strategies:**

* + Each island uses combination of (*arithmetic\_crossover* & *blx\_alpha\_crossover*) and (*gaussian\_mutation* & *cauchy\_mutation*).

**4. Survivor Selection:**

* + Elitism.

**5. Migration:**

* + Every 50 generations, 2 individuals migrate between neighboring islands (ring topology).

**6. Fitness Evaluation:**

* + Done in parallel using joblib for speed.

* **logging & Results:**
  + Best loss recorded per generation.
  + Logs printed every 100 generations, showing:
    - Global best loss
    - Each island’s best and average fitness

* **Final Output & Visualization:**
  + *Train accuracy:* **94.70%**
  + *Test accuracy:* **90.08%**

Hybrid (GA + DE)

* **Goals:**

**GA** acts like a global searcher (Exploration), and **DE** acts as a local searcher (Exploitation), they avoid premature convergence.

* **Complementary Strengths:**
  + **GA’s** crossover and mutation operators diversify the population.
  + **DE’s** differential mutation efficiently refines solutions around promising areas.

* This hybridization uses the simple GA algorithm then takes the results to initialize the DE algorithm.

**Results:**

* *Train accuracy:* **96.10%**
* *Test accuracy:* **89.50%**

* **Seeds Comparison:** *(Test accuracy and Best Loss)*

|  |  |  |
| --- | --- | --- |
| **Seeds** | **Test Accuracy** | **Best Loss** |
| **10** | **0.9030** | **0.3514** |
| **20** | **0.9073** | **0.3504** |
| **30** | **0.8907** | **0.3851** |
| **40** | **0.8828** | **0.3870** |
| **50** | **0.8945** | **0.3542** |

PSO (*lbest*)

* **Goal:**

Our ***lbest*** PSO (Local Best PSO) uses a ring topology where each particle is influenced only by its nearest neighbors rather than the entire swarm.

**The main goals:**

* + - Prevent Premature Convergence – unlike *gbest (global)*
    - Improve Exploration in Multimodal Landscapes
    - Balance Exploration & Exploitation

* **Algorithm Components:**

**1. Population Initialization:**

* + Population of size P: each individual is a vector of network weights, initialized using ***He-initialization*** (Smart initialization)*.*
  + Vector size = 25,450 (total parameters in the NN).

**2. Personal bests Initialization**

**3. Neighbours Initialization:**

* + Find local best using ring topology, my neighbours are after and before me.

**4. Update velocity:**

* + *v\_i(t+1) = w \* v\_i(t) + c1 \* r1 \* [y\_i(t) - x\_i(t)] + c2 \* r2 \* [y^\_i(t) - x\_i(t)]*

**5. Update position:**

* + *x\_i(t+1) = x\_i(t) + v\_i(t+1)*

**6. Evaluate Fitness**

* **logging & Results:**
  + Best loss recorded per generation.
  + Logs printed every 100 generations.

**Results:**

* *Train accuracy:* **91.80%**
* *Test accuracy:* **87.88%**

* **Seeds Comparison:** *(Test accuracy and Best Loss)*

|  |  |  |
| --- | --- | --- |
| **Seeds** | **Test Accuracy** | **Best Loss** |
| **10** | **0.8769** | **0.414** |
| **20** | **0.8828** | **0.8828** |
| **30** | **0.8769** | **0.4144** |
| **40** | **0.8996** | **0.3458** |
| **50** | **0.8813** | **0.3900** |

* **Back Propagation:**
  + **Seeds Comparison:** *(Test accuracy and Best Loss)*

|  |  |  |
| --- | --- | --- |
| **Seeds** | **Test Accuracy** | **Best Loss** |
| **10** | **0.9117** | **0.3028** |
| **20** | **0.9118** | **0.2964** |
| **30** | **0.9117** | **0.3028** |
| **40** | **0.9130** | **0.3034** |
| **50** | **0.9112** | **0.3039** |

**Comparison Table of Optimization Algorithms (Digit\_MINST):**

|  |  |  |
| --- | --- | --- |
| **Algorithms** | **Average Testing Accuracy** | **Best Loss** |
| **DE** | **0.8598** | **0.4677** |
| **JADE** | **0.8761** | **0.4157** |
| **GA** | **0.8958** | **0.3656** |
| **PSO** | **0.8835** | **0.3019** |
| **Hybrid (GA+DE)** | **0.8956** | **0.3656** |
| **Back Propagation** | **0.9119** | **0.3019** |

**Comparison Table of Optimization Algorithms (Fashion\_MINST):**

|  |  |  |
| --- | --- | --- |
| **Algorithms** | **Testing Accuracy** | **Best Loss** |
| **DE** | **0.7528** | **0.6922** |
| **JADE** | **0.7835** | **0.6121** |

**Conclusion:**

We trained a neural network for MNIST digit recognition using evolutionary algorithms (GA, DE, PSO) and compared them to backpropagation. While backpropagation was faster and more accurate, DE and PSO showed competitive performance, avoiding local optima and offering gradient-free optimization. Evolutionary methods are viable alternatives for non-differentiable or complex problems. Future work could explore hybrid approaches or dynamic architectures.

* Evolutionary algorithms (GA, DE, PSO) successfully optimized neural weights.
* DE balanced exploration well; PSO converged faster.
* Backpropagation remains superior for efficiency but is less robust to local minima.
* Potential for hybrid methods in future research.

**Prepared by:**

1. Abdelrahman Mohamed Hassan
2. Mahmoud Osama Hassan
3. Rehab Hamdy Abdallah
4. Rawan Hesham Mostafa
5. Aya Mohamed Abdelfattah

**Supervised by:**   
 **Amr Ghoneim**