Differential Evolution for Neural Network Optimization

1. Project Idea in Detail

This project presents an optimization approach for training neural networks using Differential Evolution (DE), an evolutionary algorithm. Instead of traditional backpropagation, DE is applied to directly evolve the weights of a neural network for a classification task. The key goal is to compare this technique with gradient-based training methods and investigate its performance, stability, and convergence when applied to the MNIST digit classification dataset.

The DE-based training process is composed of the following major components: initializing a population of weight vectors, applying mutation and crossover operations to generate candidate solutions, evaluating their fitness via model accuracy on a validation set, and finally applying selection mechanisms to retain the most promising individuals.

---

2. Main Functionalities

- Generate initial population using either **random** or **Gaussian** noise strategies
- Evaluate fitness of individuals using model accuracy on validation data
- Apply multiple mutation strategies **(rand_1, best_1)**
- Apply crossover methods **(binomial and exponential)**
- Use selection mechanisms **(greedy, tournament, crowding)**
- Track convergence of the algorithm across generations
- Enable early stopping based on stagnation or fitness variance

---

3. Similar Applications in the Market

- **Neuroevolution of Augmenting Topologies (NEAT)**: A well-known approach for evolving both weights and architectures of neural networks.
- **Google AutoML**: Uses evolutionary strategies and reinforcement learning to automate the search of neural architectures.
- **TPOT (Tree-based Pipeline Optimization Tool)**: Employs genetic programming to optimize machine learning pipelines.
- **Evonet**: A library that implements DE and GA to evolve neural networks for classification problems.

4. Literature Review

1. **X. Yao, Y. Liu, and G. Lin**,
   "Evolutionary programming made faster,"
   IEEE Transactions on Evolutionary Computation, vol. 3, no. 2, pp. 82–102, 1999.
   link Num[1] in last page
   ‣ Introduces improvements in evolutionary strategies, including techniques applicable to Differential Evolution (DE), for accelerating function optimization.

2. **R. Storn and K. Price**,
   "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces,"
   Journal of Global Optimization, vol. 11, pp. 341–359, 1997.
   link Num[2] in last page
   ‣ The foundational paper proposing the Differential Evolution algorithm.

3. **M. Pant, R. Thangaraj, and A. Abraham**,
   "Differential evolution: A review of recent variants and applications,"
   Soft Computing, vol. 15, no. 8, pp. 1575–1584, 2011.
   link Num[3] in last page
   ‣ A review of various DE variants and its applications, especially in areas like training neural networks.

4. **S. Das and P. N. Suganthan**,
   "Differential evolution: A survey of the state-of-the-art,"
   IEEE Transactions on Evolutionary Computation, vol. 15, no. 1, pp. 4–31, 2011.
   link Num[4] in last page
   ‣ A comprehensive survey of DE algorithms, highlighting theoretical advancements and practical applications.

5. **V. Salomon and D. Rodoplu**,
   "Training neural networks using differential evolution,"
   IEEE International Conference on Neural Networks, 2003.
   link Num[5] in last page
   ‣ Explores DE as a method for training feedforward neural networks, including performance on MNIST-like datasets.

6. **E. Mezura-Montes and C. A. Coello Coello**,
   "Constraint-handling in nature-inspired numerical optimization: Past, present and future,"
   Swarm and Evolutionary Computation, vol. 1, no. 4, pp. 173–194, 2011.
   link Num[6] in last page
   ‣ Discusses various techniques for constraint handling in evolutionary algorithms, including how DE deals with constraints.

5. Dataset Employed

- **Dataset**: MNIST Handwritten Digit Dataset
- **Source**: https://www.kaggle.com/datasets/oddrationale/mnist-in-csv
- **Description**: 70,000 grayscale images (60,000 for training, 10,000 for testing), each of size 28x28 pixels, labeled with digits from 0 to 9.
- **Preprocessing**: Normalized pixel values, one-hot encoded labels

---

6. Algorithm and Experimental Results

Algorithm:

Individual and Population Generation

- **Individuals** are flattened arrays of neural network weights.
- Two strategies for population initialization:
    - **Random**: Fully random weight vectors generated using different seeds.
    - **Gaussian**: Perturbed versions of a base individual with Gaussian noise.

Fitness Evaluation

- The fitness function measures **classification accuracy** on validation data.

Mutation Strategies

- **mutation_rand_1**: Classical DE strategy using three randomly chosen individuals.
- **mutation_best_1**: Uses the best-performing individual as the base vector for mutation.

Crossover Strategies

- **Binomial Crossover**: Each gene in the offspring has a probability CR to come from the mutant.
- **Exponential Crossover**: Continuous segment from the mutant replaces genes in the target individual.

Selection Strategies

- **Greedy Replacement**: Offspring replaces parent only if it performs better.
- **Tournament Selection**: Probabilistically selects the better individual.
- **Crowding**: Replaces the most similar individual in the population if the offspring is fitter.

DE Execution

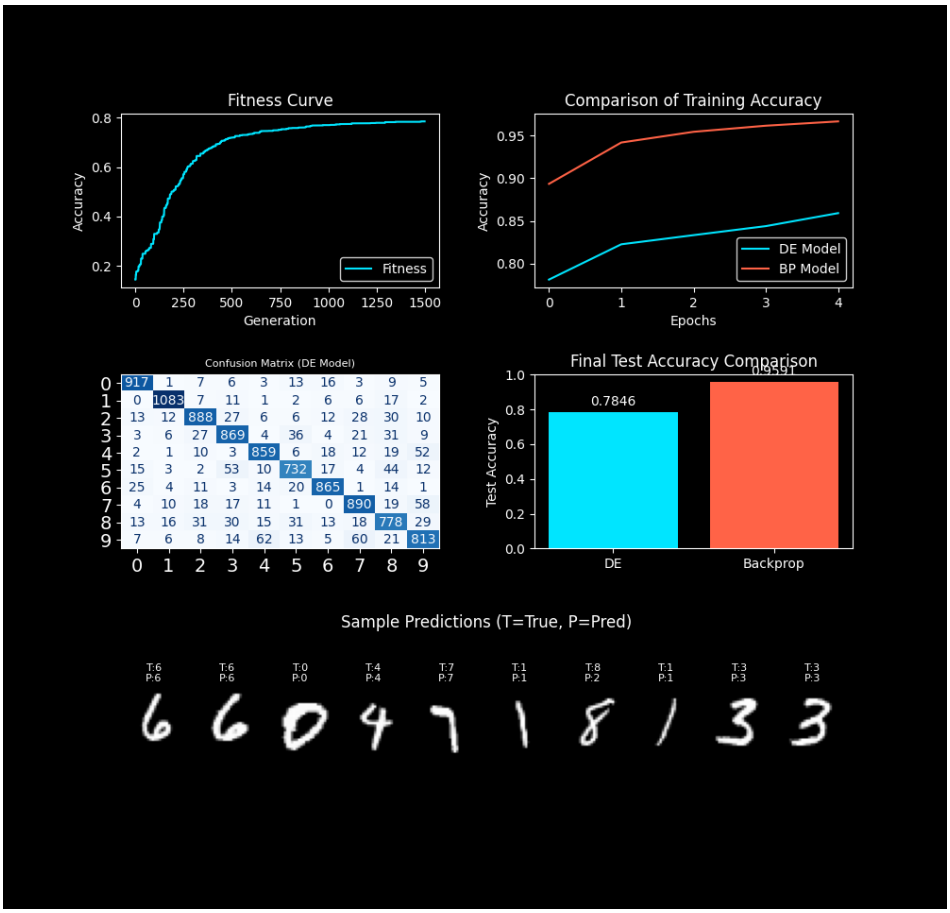
The differential_evolution function performs the evolutionary loop:

- Mutation → Crossover → Evaluation → Selection
- Tracks best fitness per generation and supports progress callbacks.

Experimental Results:

DE Experiment example Detail:

(Random Seed: 115000 , Population Size: 20 ,Mutation Factor (F): 0.5 ,Crossover Rate (CR): 0.7 ,Generations: 1500 ,Mutation Strategy: mutation_rand_1 ,Crossover Strategy: crossover_binomial ,Selection Strategy: select_better ,Population Init Strategy: random)



7. Development Platform

- **Programming Language**: Python
- **Frameworks**:
    - TensorFlow 2.x (GPU enabled)
    - NumPy
    - Keras
- **Platform**: Windows with NVIDIA GPU support
- **Execution Environment**: Visual Studio Code environment with GPU acceleration
- **Project Modules**:
    - model_utils.py: Contains build_model, flatten_weights, set_model_weights
    - main.py: Executes DE logic and experiment loop

[1] https://doi.org/10.1109/4235.771163

[2] https://doi.org/10.1023/A:1008202821328

[3] https://doi.org/10.1007/s00500-010-0641-4

[4] https://doi.org/10.1109/TEVC.2010.2059031

[5] https://doi.org/10.1109/IJCNN.2003.1223514

[6] https://doi.org/10.1016/j.swevo.2011.10.001