MAGE: A Multi-Agent Genetic Evolution Framework for Neural Architecture Optimization

Abstract The MAGE (Multi-Agent Genetic Evolution) algorithm is a neural network optimization framework that merges evolutionary computation with adaptive randomness to evolve architectures, weights, and hyperparameters simultaneously. Initially designed as a lightweight self-evolving trainer, MAGE now integrates two distinct search paradigms — Single-Architecture Evolution and Multi-Architecture Exploration — allowing flexible behavior depending on research goals. Unlike classical Neuroevolution or NAS methods that rely on complex graph search, MAGE operates on compact genetic representations of neural networks, promoting diversity through randomness and stability through fitness control. This paper presents the conceptual foundation, system architecture, mathematical formulation, pseudocode, and implementation flow of MAGE.

1. Introduction Neural network optimization traditionally depends on gradient descent methods tuned by human-engineered architectures and learning rates. However, such manual tuning limits scalability and adaptability. MAGE was conceived to automate this process using a population-based evolutionary framework where each agent represents an independent neural model with its own parameters, architecture, and hyperparameters. These agents evolve collectively — learning locally, competing globally, and mutating based on performance.

The initial design of MAGE focused purely on randomness-based self-organization, inspired by population diversity in natural systems. The randomness was the engine of discovery — agents began with random weights, random architectures, and minimal rules. However, this produced instability at scale. The modern form of MAGE introduces controlled randomness, adaptive mutation, and search modes (single and multi) to ensure both stability and exploratory capacity. This paper formalizes that final design.

2. Evolution of the MAGE Concept MAGE evolved from a random experimentation framework to a structured evolutionary system combining determinism, exploration, and self-adaptation.

3. Core Architecture of the MAGE Class Each instance of MAGE represents an evolutionary trainer with: - Population of agents: Each agent is a neural network defined by architecture, weights, and hyperparameters. - Fitness function: Validation loss (val_loss) determines evolutionary fitness. - Evolutionary loop: Repeated selection, mutation, and replacement across master epochs.

4. Algorithm Workflow Population Initialization: Each agent begins as Ai = {Wi, Bi, archi, hpi}. Local Learning: Agents train locally, compute val_loss. Global Selection: Rank agents by fitness Fi = ValLossi + $\lambda \cdot$Complexity(archi). Mutation: Apply stochastic transformations based on rank and mode. Controlled Randomness: Deterministic RNG ensures repeatable results.

5. Search Modes Single Mode: Fixed architecture, varied weights/hyperparams. Multi Mode: Random architecture per agent, NAS-like behavior.

6. Evolutionary Loop Summary Initialize population $\rightarrow$ Train locally $\rightarrow$ Evaluate $\rightarrow$ Select elites $\rightarrow$ Mutate $\rightarrow$ Replace population $\rightarrow$ Repeat.

7. Validation Loss Lifecycle Val_loss = L(f(X_val;W,B), y_val). Lower val_loss $\rightarrow$ Higher survival probability.

8. Experimental Observations Single mode optimizes fixed structure; Multi mode explores new architectures.

9. Key Features - Deterministic RNG - Controlled Mutation - Dual-Mode Search - CSV Logging - Unified Architecture & Hyperparam Evolution - Complexity-Aware Fitness

10. Example Result Interpretation Example: Epoch 3, Agent 0, ValLoss=0.265, Architecture=[784,128,10], Activation=relu, LR=0.0009, Batch=128.

11. Conclusion MAGE is a minimal yet complete evolutionary intelligence system combining stochastic exploration and deterministic reproducibility. It enables both deep optimization and broad exploration, making it a scalable base for evolutionary AI.

12. Future Directions - Adaptive learning rates - Genetic crossover - Parallel GPU/CPU execution - Multi-objective fitness functions

Appendix: Pseudocode Initialize population of N agents for master_epoch in range(M): for each agent: train locally compute val_loss rank agents by fitness select top elites mutate others (architecture, weights, hyperparams) return best agent