Neural Network Training Using Particle Swarm Optimization

Poonam Rajan Pawar
pp1549@rit.edu
Rochester Institute of Technology

***Abstract*—The performance of neural networks is determined by both structural parameters (e.g., the number of neurons per layer) and non-structural parameters (e.g., weights and biases). Traditional optimization methods, such as gradient-based approaches, often face challenges like computational inefficiency and susceptibility to local minima. This paper investigates the use of Particle Swarm Optimization (PSO), a non-gradient-based method, to search both structural and non-structural parameters of neural networks. PSO's ability to navigate complex, high-dimensional search spaces makes it a promising alternative for enhancing neural network performance. Through experiments on multiple datasets, we demonstrate that PSO can effectively be used to find the number of neurons per layer and the weights during training, leading to improved accuracy and reduced loss. The results highlight PSO's potential as a complementary approach.**

***Index Terms*—neural networks, particle swarm optimization, machine learning**

I. Introduction

The performance of neural networks depends on its structural parameters (e.g., number of neurons per layer) and non-structural parameters (e.g., weights and biases). Finding effective configuration of both is important for network's performance [1]. This study investigates the effectiveness of Particle Swarm Optimization (PSO) in identifying neural network configurations.

Traditional methods for searching structural parameters, such as grid search or random search have shortcomings, mainly computational inefficiency. To effectively find weights, gradient-based methods such as stochastic gradient descent and its variants like Adam are widely used [4]. However, they face challenges such as local minima and plateaus in the optimization landscape.

PSO, a non-gradient method, offers a possible solution for searching both structural and non-structural parameters. Neural networks learn weights during training, so we are using PSO during this training to effectively find them through search space. PSO's ability to navigate complex, high-dimensional spaces makes it a suitable candidate [1].

II. Related Work

The standard PSO and its variants have demonstrated significant benefits for various commercial applications. Jiao et al. [3] introduced a novel approach for optimizing the location of electric business centers using Center–Decenter Quantum Particle Swarm Optimization (CDQPSO), an enhanced version of PSO. It incorporates quantum mechanics and centralized-decentralized learning; CDQPSO reduces the risk of falling into local optima and improves global search capabilities. The authors compare CDQPSO with standard PSO and its variants, demonstrating its superior performance in terms of accuracy and convergence rate. For example, in high-dimensional problems, CDQPSO achieved a fitness value of 0.033 in a real-world case study, effectively balancing service quality, transportation convenience, and construction costs.

In the field of neural network optimization, Ruder [4] provides a comprehensive overview of gradient descent optimization algorithms, highlighting their strengths and limitations. This work emphasizes the need for alternative approaches like PSO for cases where gradient-based methods struggle with local optima. Additionally, Goodfellow et al. [2] discuss various optimization methods for training deep models, providing context for our exploration of PSO as a complementary approach.

III. Exploitation and Exploration

Exploitation refers to leveraging known information or known search space to refine candidate solutions. It involves structured, local improvements. It focuses on deeply searching in the direction of promising search space.

Exploration focuses on discovering new regions and covering global convergence. It involves sampling new or lesser-known regions of the search space. It uses randomness to introduce diversity.

IV. Particle Swarm Optimization

PSO is a swarm intelligence technique inspired by the collective behavior of birds or fish. It is widely used for optimization problems where a population of candidate solutions, called particles, explores the search space to find optimal or near-optimal solutions [1].

At any iteration t, each particle i is defined by:

- A position $X_i(t)$, representing a candidate solution.
- A velocity $V_i(t)$, determines the direction and step size of the particle's movement.

Particles adjust their trajectories based on:

1) Personal Best (X_i^*): The best solution discovered by particle i so far.
2) Global Best (g^*): The best solution found by any particle in the swarm.

PSO balances exploration and exploitation through velocity and position updates:

$$\mathbf{V}_i(t+1) = \theta \mathbf{V}_i(t) + \alpha \epsilon_1 (\mathbf{X}_i^* - \mathbf{X}_i(t)) + \beta \epsilon_2 (\mathbf{g}^* - \mathbf{X}_i(t)) \tag{1}$$

$$\mathbf{X}_i(t+1) = \mathbf{X}_i(t) + \mathbf{V}_i(t+1) \tag{2}$$

where:
$\mathbf{V}_i(t)$ is the velocity of particle i at iteration t
$\mathbf{X}_i(t)$ is the position of particle i at iteration t
θ is the inertia weight
α, β are acceleration coefficients
ϵ_1, ϵ_2 are random numbers uniformly distributed in $[0, 1]$

The above update equations modify each particle's position vector at every iteration, collectively steering the swarm toward high-fitness regions of the search landscape. The procedure repeats these updates until the convergence criterion—or any other predefined termination condition is satisfied. The following pseudo code formalizes this process.

Algorithm 1

```
1: Initialize particles with random positions ($X_i$) and velocities ($V_i$).
2: Evaluate the fitness of each particle.
3: Update local best ($X_i^*$) and global best ($g^*$).
4: for each particle do
5:     Update velocity using Equation 1
6:     Update position using Equation 2
7: end for
8: Repeat steps 2–6 until convergence or a maximum number of iterations is reached.
9: Output the global best solution ($g^*$).
```

V. Genetic Algorithms

Genetic Algorithms (GA) operate with a population of candidate solutions, often represented as particles or configurations. These algorithms rely on two core evolutionary operations: crossover and mutation, which drive exploration and diversity in the search space [1].

- Crossover: This operation combines genetic information from two parent solutions (particles/genes) to produce new offspring. For instance, if solutions are encoded as vectors, crossover remixes or swaps components (e.g., segments or individual indices) between the parents, creating novel configurations (Figure 1). This facilitates the exchange of beneficial traits across the population.
- Mutation: In contrast, mutation acts on a single solution by introducing random perturbations, such as flipping or altering values at specific indices (Figure 1). This injects new variability, helping the algorithm escape local optima and explore uncharted regions of the search space.

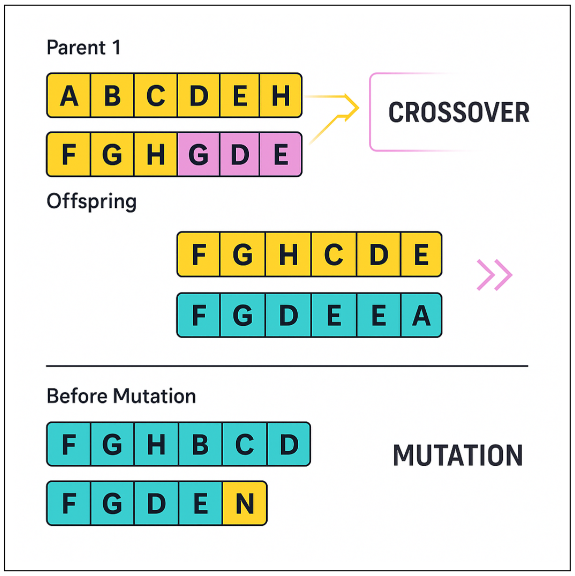

Fig. 1: GA operations

Together, these mechanisms prioritize exploration, continuously diversifying the solution pool to navigate toward optimal or near-optimal configurations. The strength of GAs lies in their ability to balance broad search-space exploration with the refinement of promising solutions.

VI. Evaluation

Throughout this experiment, loss serves as the primary evaluation metric. Since neural network training fundamentally involves minimizing a loss function, the loss value directly quantifies the discrepancy between the model's predicted outputs and the actual labels. A lower loss indicates that the predictions align more closely with the true values, reflecting greater model confidence. Thus, minimizing loss is critical—it signifies a better-performing configuration [2].

While accuracy is another important metric, it can be misleading in certain scenarios. Models may achieve high accuracy (e.g., 90%) yet still exhibit a substantial loss (e.g., 123.10), revealing significant deviations between predictions and ground truth. Since loss and accuracy are not always correlated, relying solely on accuracy may obscure underlying performance issues. Therefore, this study prioritizes loss minimization as the key optimization objective.

VII. Datasets

The experiment spans seven diverse datasets, from tabular collections like Iris and Wine to extensive image repositories such as MNIST and CIFAR-10, encompassing 4-3072 features and 150-70,000 samples. Tabular datasets (Iris, Wine, Digits, Breast Cancer, and Spam Base [7]) employ Multilayer Perceptron (MLP) architectures optimized for numeric feature vectors, while image datasets leverage specialized Convolutional Neural Network (CNN) implementations—LeNet5 for MNIST and AlexNet for CIFAR-10—designed for pixel-based inputs. This methodological diversity enables a comprehensive evaluation of our approach across varied neural network architectures and data complexities.

VIII. Methodology And Results

A. Structural Parameters

During this setup, each particle encodes a candidate neuron count $n \in [n_{\min}, n_{\max}]$ for the hidden layer(s). For every candidate, the network is freshly initialized, trained to convergence, and its validation loss is logged. At every iteration the personal- and global-best topologies redirect the swarm toward higher-performing regions of the search space; optimization halts once no further improvement is observed.

TABLE I: Neural Network Architecture Optimization Results

Dataset	Loss	Accuracy	Selected Neurons (Range)
IRIS	0.0535	100%	32 (32–512)
WINE	0.2011	97%	56 (5–100)
DIGITS	0.0522	98%	100 (5–100)

*1) **Results & Discussion**:* Across all evaluated datasets (Table I), the procedure consistently reduced validation loss and preserved—or slightly improved—classification accuracy, indicating that the search converged on compact yet expressive architectures.

Key Observations:

- Each fitness evaluation demands a full re-initialization and training run, driving compute cost sharply upward with swarm size and iteration count;
- Rigorous benchmarking of structural hyper-parameters requiring controlled random seeds, identical training schedules, and multiple repeats lies beyond the present scope [8].

B. Non-Structural Parameters

Each experimental configuration spanning several distinct model architectures is first pretrained with a gradient-based optimizer, yielding an initial set of weight parameters for subsequent analysis. This pretrained model with its weights provides an excellent starting point and a well-initialized set of parameters for the PSO algorithm.

A core element of the experimental pipeline is to (i) extract the network's weight and encode it as particle representations, (ii) iteratively optimize those particles within the proposed framework, and (iii) decode the refined particles back into the model's layers as weights.

*1) **Encoding of Weights in the Particle**:* The weight encoding process primarily targets dense layers, which are fully connected neural network layers (though this could vary depending on the network architecture employed).

From each layer:

- Trainable layers are selected for processing. The weight matrices (present at index 0 in the layer's parameters) are extracted.
- These weights are then flattened, which converts multi-dimensional weight matrices into a single 1D array (Figure 2).
- This flattening of weights into one dimension facilitates their conversion into a particle format.

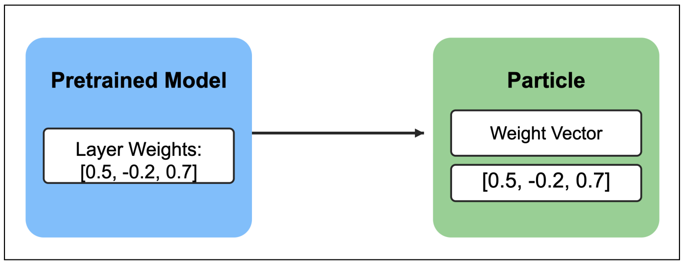

Fig. 2: Encoding Process

After completing the extraction process, a single one-dimensional array containing all weights from the trainable dense layers is generated. This methodical approach ensures that weights are fetched in a consistent order and encoded with precise indexing in the particle representation.

*2) **Decoding of Weights in the Particle**:* During the decoding process, weights are extracted from the global best position and reshaped to match the model's original architecture. Due to the deterministic flattening of weights into a one-dimensional array, the correct indices can be precisely selected and assigned to their corresponding layers. This process is systematically repeated for all remaining layers (Figure 3).

In both the encoding to the particle and decoding back to the model, layers are processed in exactly the same order each time. This consistency ensures that weights are reliably mapped to the same positions, maintaining the structural integrity of the neural network throughout the optimization process.

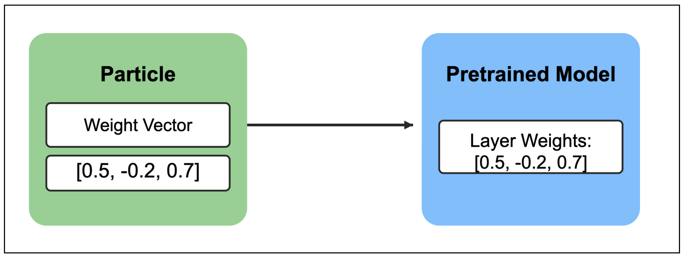

Fig. 3: Decoding process

*3) **PSO Setup**:* After encoding, each particle in the PSO algorithm represents a potential weight configuration of the model. These particles iteratively update their positions based on the velocity equation 1 and position equation 2 until the stopping criteria are met. The process begins with particles initialized at different locations in the search space, gradually converging toward an optimal solution. Finally, the best-performing particle's weights are decoded, and the model's accuracy is evaluated.

Results & Discussion: The PSO algorithm was tested on multiple datasets. Table II summarizes the result:

TABLE II: PSO Setup Results

Dataset	Loss	Accuracy
Iris	0.33 → 0.32 (3 % ↓)	93% → 93% (→ 0 %)
Wine	0.42 → 0.42 (→ 0 %)	97% → 100% (3.1 % ↑)
Digits	0.65 → 0.49 (24.6 % ↓)	84% → 86% (2.4 % ↑)
Breast Cancer	0.05 → 0.05 (→ 0 %)	98% → 99% (1.0 % ↑)
Spam Base	0.24 → 0.24 (→ 0 %)	91% → 91% (→ 0 %)
MNIST	0.49 → 0.49 (→ 0 %)	84% → 84% (→ 0 %)
CIFAR-10	0.20 → 0.07 (65 % ↓)	95% → 98% (3.2 % ↑)

Key Observations:

- Significant loss reduction was observed in the Digits (24.6 %) and CIFAR-10 (65 %) datasets, indicating effective convergence.
- Accuracy improvements were seen in Wine (3.1 %), CIFAR-10 (3.2 %), and Breast Cancer (1.0 %); some datasets (e.g., Iris, Spam Base, MNIST) showed no change.
- Stagnation in optimization occurred in several cases (e.g., Wine, MNIST), likely due to PSO getting trapped in local optima.

Particle Stagnation in PSO and Hybrid PSO–GA:

- **Stagnation of sub-optimal particles:** Figures 4–5 and Figures 6–7 represent the best- and worst-performing particles, respectively, for the Digits and Breast Cancer datasets. Although all particles followed consistent trajectories across iterations, comparative analysis shows that those in Figures 6–7, deemed sub-optimal, had minimal impact on the swarm's convergence.

  The PSO velocity update rule predominantly guides particles toward global or personal best solutions, allowing poorly performing particles to linger without contributing meaningfully to the search. This absence of an explicit elimination mechanism limits swarm diversity and overall efficiency.
- **Implications for high-dimensional search:** This stagnation suggests that standard PSO can struggle when navigating complex, high-dimensional landscapes.
- **Hybrid PSO–GA:** To enhance exploration and address the stagnation issue, we incorporated GA operators—specifically crossover and mutation—into the PSO framework.

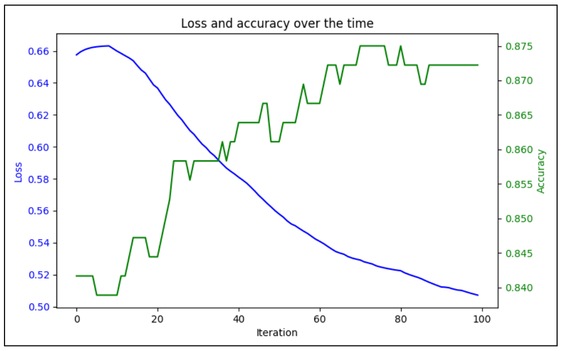

Fig. 4: Convergence behavior of best-performing particle on the Digits dataset

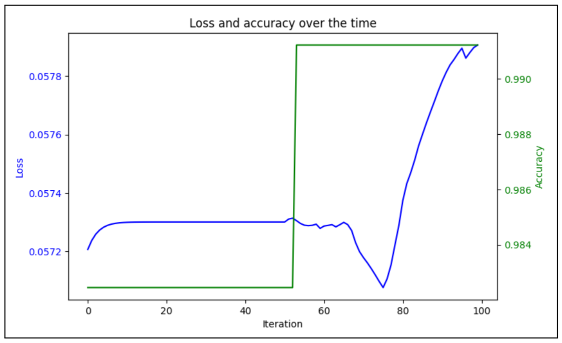

Fig. 5: Convergence behavior of best-performing particle on the Breast Cancer dataset

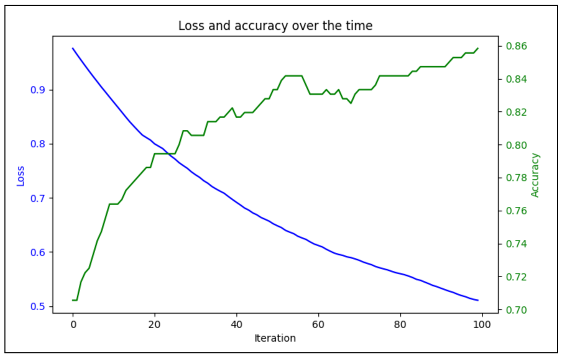

Fig. 6: Convergence behavior of worst-performing particle on the Digits dataset

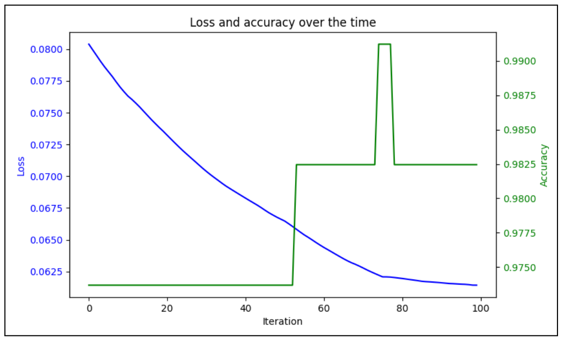

Fig. 7: Convergence behavior of worst-performing particle on the Breast Cancer dataset

*4) **PSO & GA Setup**:* In this framework, each cycle is executed in two complementary stages:

- **Exploitation (PSO phase) –** Particle velocities are updated with the canonical cognitive-social rule, intensifying the search around the swarm's most promising regions.
- **Exploration (GA phase) –** Selection, crossover, and mutation are applied to the current particle pool, synthesizing novel solution vectors and expanding the search frontier.

This interaction delivers three tangible benefits in high-dimensional parameter spaces:

- **Population diversity:** GA perturbations inject previously unseen, potentially higher-fitness candidates, limiting swarm stagnation and premature convergence.
- **Adaptive replacement:** Particles with negligible contribution are discarded and substituted by freshly generated, potentially advantageous individuals, ensuring computational effort is devoted to influential configurations.
- **Progressive refinement:** The surviving population inherits both the directional intelligence of PSO and the structural variation of GA, accelerating convergence toward high-quality search space.

Figure 8 schematically depicts the sequence of PSO updates followed by GA-driven replacement, illustrating how sub-optimal particles are removed and replaced with more promising alternatives to sustain overall performance gains.

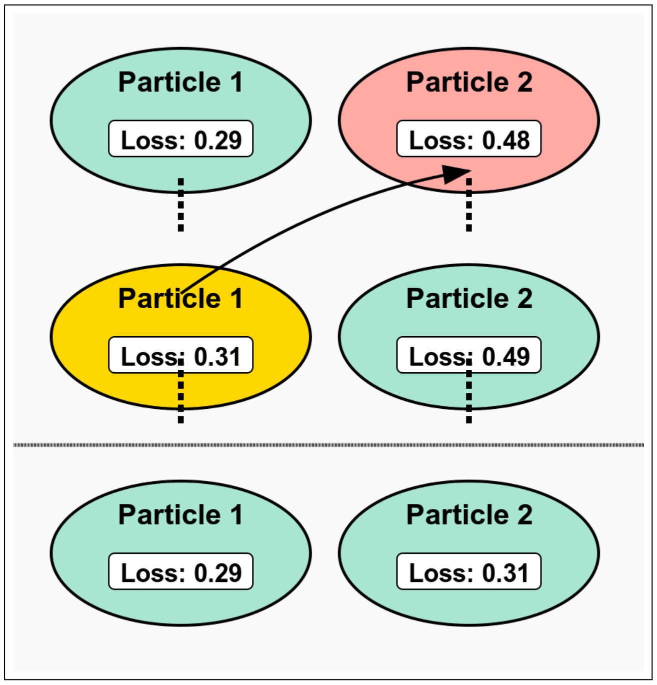

Fig. 8: Selection Mechanism

Results & Discussion:

TABLE III: Hybrid PSO-GA Results

Dataset	Loss	Accuracy
Iris	0.61 → 0.43 (29.5 % ↓)	89% → 93% (4.5 % ↑)
Wine	0.19 → 0.05 (73.7 % ↓)	97% → 100% (3.1 % ↑)
Digits	3.00 → 0.67 (77.7 % ↓)	43% → 84% (95.3 % ↑)
Breast Cancer	0.50 → 0.0006 (99.9 % ↓)	98% → 100% (2.0 % ↑)
Spam Base	0.20 → 0.12 (40.0 % ↓)	92% → 95% (3.3 % ↑)
MNIST	0.44 → 0.33 (25.0 % ↓)	85% → 89% (4.7 % ↑)
CIFAR-10	0.20 → 0.11 (45.0 % ↓)	96% → 97% (1.0 % ↑)

The hybrid approach combines PSO's fine-grained local exploitation with GA-driven global exploration, achieving superior results with significantly lower objective loss compared to standalone PSO.

Key Observations:

- Relative to the baseline figures in Table II, loss on the Digits dataset improvised from 24.6% to 77.7%, while Iris improved from 3.0% to 29.5%.
- Datasets that previously showed no measurable benefit under PSO— Breast Cancer, SpamBase, and MNIST—now register gains of 99%, 40%, and 25%, respectively, demonstrating the hybrid's ability to escape local optima through sustained population diversity. Collectively, these results indicate that the algorithm achieves an effective balance between exploration and exploitation.
- In contrast, CIFAR-10's improvement receded from an early 65% to 45% at convergence. We hypothesize that the high-dimensional convolutional architecture introduces strong inter-parameter dependencies; discrete GA operators may disrupt these weight correlations, limiting the effectiveness of gradient-free updates. This suggests that underlying network topology critically mediates the utility of such hybrid, non-gradient optimizers.

C. Convergence Dynamics

Gradient-based optimization methods, such as Adam, leverage derivatives to navigate the search space more effectively enabling better convergence. Our experiments show that Adam, a gradient-based method, achieves competitive performance with increased epochs, reinforcing the superiority of gradient-driven approaches in neural network training. Future work may explore hybrid approaches combining gradient and evolutionary methods for improved performance.

TABLE IV: Performance Metrics with gradient methods

Dataset	Loss	Accuracy
Iris	0.56	90%
Wine	0.16	97%
Digits	0.67	87%
Breast Cancer	0.11	97%
Spam Base	0.20	92%
MNIST	0.08	97%
CIFAR-10	0.03	99%

IX. Conclusion

Neural networks possess high-dimensional weight spaces, making optimization a challenging task, particularly for non-gradient-based methods. The curse of dimensionality significantly impacts perturbation techniques, as the search space grows exponentially with the number of parameters, leading to inefficient convergence compared to gradient based methods. This paper presented an approach for enhancing neural network performance using Particle Swarm Optimization (PSO) for both structural and non-structural parameter optimization.

References

[1] X. S. Yang, *Nature-Inspired Optimization Algorithms*, Elsevier, 2014.
[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[3] R. Jiao, X. Huang, H. Ouyang, G. Li, Q. Zheng, and Z. Jiang, *Optimal electric business centre location by centre-decentre quantum particle swarm optimization*, Systems Science & Control Engineering, vol. 7, no. 1, pp. 222–233, 2019.

[4] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747, 2016.

[5] R. Eberhart and J. Kennedy, *A new optimizer using particle swarm theory*, in Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995, pp. 39-43.

[6] Y. Shi and R. Eberhart, *A modified particle swarm optimizer*, in IEEE International Conference on Evolutionary Computation, 1998, pp. 69-73.

[7] M. Lichman, *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, 2013.

[8] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, *NAS-Bench-101: Towards Reproducible Neural Architecture Search*, in Proceedings of the 36th International Conference on Machine Learning, 2019