Particle Swarm Optimization to Train Neural Networks

Swaroop Thamimeni, Alan Wang and Nikhil Sobanbabu

April 2025

Abstract

Particle Swarm Optimization (PSO) offers a gradient-free, population-based approach inspired by natural swarming behaviors, applied here to train neural networks. We implemented base PSO and three advanced variants—PPSO, SGPSO, and PSOGSA—to optimize multilayer perceptrons on Iris, Wine, and Breast Cancer datasets from UCI. Comparative experiments against SGD and Adam demonstrate that PSO variants achieve competitive accuracy and robustness, while a hyperparameter sensitivity study highlights trade-offs in convergence speed and stability.

1 Introduction and Motivation

Training neural networks is fundamentally an optimization problem that is often high-dimensional, non-convex, and characterized by numerous local minima and saddle points in the landscape of the objective function. The traditional approach to solving this problem relies on gradient-based optimizers like Stochastic Gradient Descent (SGD) and Adam, which are efficient and scalable but also suffer from critical limitations: sensitivity to initialization, vanishing/exploding gradients, difficulty escaping poor local optima, and reliance on hyperparameters. These issues are exacerbated in low-data settings or when the loss landscape is highly irregular, raising the need for alternatives that are more robust.

Particle Swarm Optimization (PSO), a population-based metaheuristic with gradient-free optimization, offers a different approach. Inspired by the collective behavior of bird flocks and fish schools, PSO explores the parameter search space through a swarm of particles that balance individual experiences and social collaboration. The balance between collective and individual movement allows for a balance between an exploration and exploitation strategy. Unlike gradient-based methods, PSO is inherently global, requires no gradient computations, and is less sensitive to initialization, making it suitable for optimizing neural networks with rugged loss landscapes.

However, classical PSO suffers from issues of premature convergence and poor fine-tuning, limiting its effectiveness in practice. Recent variants, such as PPSO, aim to resolve this by adaptively controlling inertia weights to better balance exploration and exploitation during the optimization process.

In this project, we work off of a framework outlined in the work from ”Accuracy Improvement of Neural Network Training using Particle Swarm Optimization and its Stability Analysis for Classification”. We implement and evaluate multiple PSO variants: Basic PSO, Second Generation PSO, and PSOGSA. Alongside these, we also explore PPSO, which adaptively modulates inertia to balance exploration and exploitation during training.

To evaluate these methods, we compare them against standard gradient-based optimizers such as SGD and Adam on common classification benchmarks. Beyond performance comparisons, we conduct studies on hyperparameter sensitivity and model robustness, aiming to identify the settings under which PSO-based optimizers offer meaningful advantages and insights into their practical viability for training neural networks.

2 Background and Related Work

The PSO algorithm was introduced by Kennedy and Eberhart [1], and it was inspired by the collective foraging and movement patterns observed bird flocks and fish schools. In PSO, a set of candidate solutions navigate a search space, while each particle adjusts its position and velocity based on its personal best and the global best position across the swarm. The design of PSO was shaped in part by swarm intelligence theory, which was developed by Millonas [2], who proposed a spatial model for collective behavior in biological systems. Moreover, Millonas emphasized three fundamental implications for artificial life: (1) emergence of complex adaptive behavior, (2) criticality and phase transitions in biological collectives, and (3) behavioral criteria for the evolution of cooperations. These ideas were adopted by Kennedy and Eberhart as criteria satisfied by the PSO framework. Despite its effectiveness, standard PSO (which we'll refer to as BPSO) may suffer from premature convergence, especially when particles become overly attracted to early-found optima, causing the swarm to stagnate before reaching the global minimum.

Second generation PSO (SGPSO) improves the classic PSO algorithm by incorporating a third term into the velocity update rule: the geometric center of the swarm's top-performing particles. The mechanism was adapted from the idea that individual particles adjust their positions relative not only to immediate neighbors but also to the perceived centroid of the group [3]. By introducing a new acceleration component toward this geometric center, the algorithm balances exploration with group cohesion, thereby reducing the risk of early stagnation and enhancing convergence in dynamic environments.

PSO-Gravitational Search Algorithm (PSOGSA) [4] is a hybrid algorithm that combines the social learning behavior of PSO with the physics-inspired attraction model of the Gravitational Search Algorithm (GSA) [5]. GSA is solely based on Newtonian gravity, and it models candidate solutions as objects with masses, where heavier/fitter agents exert stronger attractive forces. In PSOGSA, particles are influenced by both the swarm's memory and gravitational forces that scale with particle fitness. In benchmark experiments, PSOGSA has demonstrated improved convergence and better avoidance of local minima compared to standalone PSO and GSA, particularly in high-dimensional neural network training tasks.

Lastly, the concept of our paper stems from the work "Accuracy Improvement of Neural Network Training using Particle Swarm Optimization and its Stability Analysis for Classification", which proposes an enhanced PSO algorithm (PPSO) that addresses key shortcomings in standard neural network training as we've mentioned [6].The algorithm–which we will delve into–is applied to train feedforward neural networks (FFNNs) for a variety of classification tasks.

3 Methodology

In this section, we discuss the algorithmic details of PSO and its variants. Further, we also discuss how these algorithms are used for FFNN training.

3.1 Base-PSO(BPSO)

This is the classical PSO that the other variants build on. Suppose the size of the swarm or the number of particles in noP and the search space is D dimensional (in our use case, the total number of NN weights and biases is equal to D. Let the position of the i^{th} particle is represented as:

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iD}), \quad \text{where } x_{id} \in [\mathrm{lb}_d, \mathrm{ub}_d], \quad d \in [1, D]$$

and lb_d and ub_d are the lower and upper bounds for the d^{th} dimension of the search space. The velocity of the i^{th} particle is:

$$\mathbf{v}_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$$

At each time step t, the velocity and position of each particle are updated using the following equations:

$$v_{ij}(t+1) = \omega \cdot v_{ij}(t) + c_1 \cdot r_1 \cdot \left(p_{ij}^{lB}(t) - x_{ij}(t)\right) + c_2 \cdot r_2 \cdot \left(p_j^{gB}(t) - x_{ij}(t)\right) \quad (1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (2)$$

Here:

- r_1, r_2 are independent random variables uniformly distributed in $[0, 1]$,
- c_1, c_2 are acceleration coefficients,
- ω is the inertia weight controlling the exploration capability,
- p_i^{lB} is the best position found by the i^{th} particle (personal best),
- p^{gB} is the best position found by the entire swarm (global best),
- The velocities are constrained to $[v_{\min}, v_{\max}]^D$ for stability.

In Equation (1), the first term $\omega \cdot v_{ij}(t)$ promotes exploration, the second term models the particle's own experience (private thinking), and the third term incorporates the swarm's collective experience (social collaboration).

3.2 FFNN Architecture and Training

We consider the following NN architecture(Figure. 1)

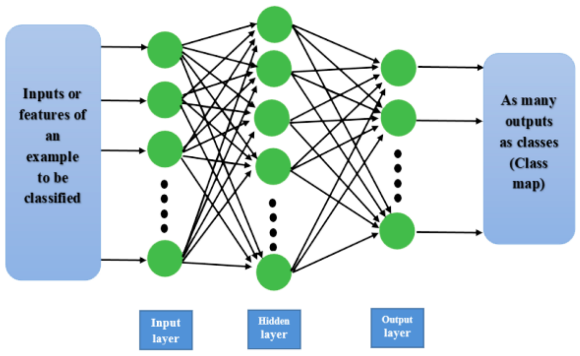
Inputs or features of an example to be classified
As many outputs as classes (Class map)
Input layer
Hidden layer
Output layer

Figure 1: Caption

First the Input Layer(IL), next hidden layers (HL) and an output layer (OL). Let the structure be p-q-r, where the number of nodes in each layer are p,q and r respectively. Usually $D = nw + nb$ where nw is the number of weights and nb is the number of biases. So in p-q-r structure, $nw = pq+qr$

and $nb = q + r$. So $D = pq + qr + q + r$. So in the noP particles considerd to train the NN, in each particle, first pq components are for the weights between IL and HL, then next q components are for the biases of HL, next qr are the weights between HL and OL and the final r are for the biases of OL. The following summarizes the training process of NN with PSO:

First the initial positions of the particles are initialised with Sobol random initializer. Then using each particle parallely, the forward pass of the NN is done and the fitness function is found. Using this, the positions and velocities of each particle is updated based on the PSO algorithm. This is repeated until a stopping criteria is met.

In our implementation the underlying neural network is a fixed MLP with one hidden layer of 16 neurons, matching input dimensions for each dataset, and a cross-entropy loss objective. All notation and hyperparameters (inertia weight schedule, c_1, c_2, population size, max iterations) are defined to ensure reproducibility.

3.3 PSO variants

PPSO: In its vanilla form, this variant is the paralleized version of BPSO. However in the work we followed, a new adaptive hyperbolic tangent based inertia weight strategy has been used:

$$w(t) = w_{min} + tanh(I_{current} \times ((w_{max} - w_{min})/I_{max} \tag{3}$$

where w_{min} is the initial inertia weight, w_{max} is the final inertia weight, $I_{current}$ and I_{max} are the current and the max iteration number respectively. Initially particles moves slowly with low inertia, encouraging more exploration. As time increases, particles become more confident in their state and pick up speed. This Encourages consolidated movement toward better regions found by the swarm.

SGPSO(Second Generation PSO): In this variant, along with the particle trying to move towards the local optimum and global optimum of the particles, there is an extra term corresponding to the geometric center of the swarm and the particle is motivated to move towards it. So the update equation is:

$$P_j(t) = \frac{1}{N} \sum p_{ij} \quad \text{(Geometric center)} \tag{4}$$

$$v_{ij}(t+1) = \omega \cdot v_{ij}(t) + c_1 \cdot r_1 \cdot \left(p_{ij}^{lB}(t) - x_{ij}(t)\right) + c_2 \cdot r_2 \cdot \left(p_j^{gB}(t) - x_{ij}(t)\right) + c_3 \cdot r_3 \cdot (P_j(t) - x_{ij}(t)) \tag{5}$$

$$\tag{6}$$

PSOGSA:

$$M_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \tag{7}$$

$$F_{ij}(t) = G(t) \sum_{\substack{k=1 \\ k \neq i}}^{N} \frac{M_i(t) M_k(t)}{R_{ik}(t) + \epsilon} \left(x_{kj}(t) - x_{ij}(t)\right) \tag{8}$$

$$a_{ij}(t) = \frac{F_{ij}(t)}{M_i(t)} \tag{9}$$

$$v_{ij}(t+1) = \omega \cdot v_{ij}(t) + c_1 \cdot r_1 \cdot a_{ij}(t) + c_2 \cdot r_2 \cdot \left(p_j^{gB}(t) - x_{ij}(t)\right) \tag{10}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{11}$$

The term $a_{ij}(t)$ is the acceleration, computed as the net gravitational force $F_{ij}(t)$ divided by the mass $M_i(t)$ of the particle. The second term incorporates GSA's mass-based attraction (Newtonian gravity), encouraging particles to move toward heavier (fitter) individuals. The mass $M_i(t)$ is calculated based on the fitness value $fit_i(t)$ of particle i at time t, with $best(t)$ and $worst(t)$ representing the best and worst fitness values in the swarm. This ensures that better-performing particles exert stronger attractive forces.

4 Experimental Results

4.1 Model Architecture

A fixed Multi-Layer Perceptron (MLP) architecture was used consistently across all datasets to isolate and evaluate the performance of different optimization algorithms independently from model complexity. The model was trained using the Cross Entropy loss function, appropriate for classification tasks involving two or more classes.

- **Input Layer**: Dimensionality matches the number of features in each dataset.
- **Hidden Layer**: A single hidden layer with 16 neurons was used to minimize the capacity of the model and highlight the impact of the optimizer.
- **Output Layer**: The number of output neurons corresponds to the number of classes in the dataset.
- **Activation Function**: Nonlinear activation functions were used.

Three benchmark datasets were employed in this study taken from Scikit Learn. The details are summarized in Table 1.

Table 1: Summary of UCI classification datasets used in experiments.

Dataset	**# Features**	**# Samples**	**# Classes**
Iris	4	150	3
Wine	13	178	3
Breast Cancer	31	569	2

4.2 Comparison of Optimizers

A diverse suite of optimizers was evaluated, including both classical and PSO-based algorithms. PSO Variants included Base PSO, Second Generation PSO (SGPSO), Gravitational Search Algorithm (GSA), Hybrid PSO + GSA (PSOGSA), and PPSO. We also evaluated with standard optimizers like Stochastic Gradient Descent (SGD) and Adam.

In terms of evaluation metrics, the optimizers were evaluating using loss (as measured by cross-entropy objective), accuracy (classification accuracy on the test set), and confusion matrices (to analyze per-class performance).

PSO variants match or exceed Adam and SGD accuracy on Iris and Wine, with PPSO showing the fastest convergence in early iterations. On Breast Cancer, SGPSO achieved a peak accuracy of 97.2%, outperforming SGD's 96.5%.

4.3 Loss Curves

Figure 2 shows the cross-entropy training losses of various PSO and standard optimizers on different datasets. The top left shows the losses on the Iris dataset, the top right shows the loss performance on the Wine dataset, the bottom left shows the loss performance on the Breast Cancer dataset, and the bottom right shows the loss performance in the original paper on a different dataset.

As seen in both the original papers' loss curve and our curve, PPSO converges relatively quicker than the other optimizer variants and achieves the lowest overall loss compared to other optimizers. This highlights how PPSO has both the property to not be trapped in local optima like other PSO variants, but also has the convergence speed performance of adaptive optimizers like Adam.

Other PSO optimizers, like BPSO and SGPSO, also converge without reaching local optima as easily, but seem not to further decrease their loss after a certain number of iterations. This is likely due to these PSO variants not having adaptive weight strategies that are shown in PPSO or Adam.

PPSO in particular also achieves relatively similar, if not quicker, convergence speeds to traditional optimizers like SGD and Adam while sometimes achieving an even lower loss than these two, highlighting the effectiveness of PSO with adaptive weighting.

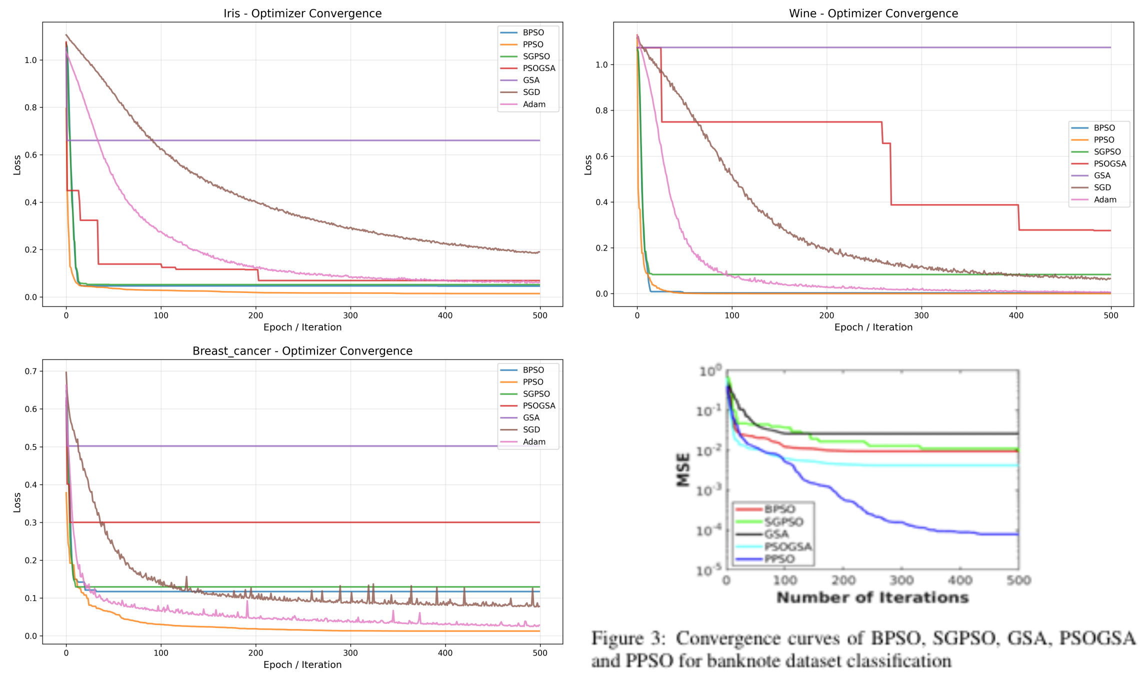

Figure 2: Loss curves across datasets and optimizers: (Top Left) Iris, (Top Right) Wine, (Bottom Left) Breast Cancer, (Bottom Right) Original Paper Loss

4.4 Accuracy Results

Figure 3 highlights the accuracy results of various optimizers and standard optimizers on the Breast Cancer test dataset, which includes all of the aforementioned PSO and standard optimizers in the loss section. Here, it is shown that while all other optimizers struggle to consistently score high on accuracy, PPSO maintains a near-perfect accuracy on the test dataset. This further highlights the strength of PPSO in that it both does not get trapped in local optima easily but also uses adaptive weighting to better find optimal weights.

The other optimizers are likely trapped in a local optima, as their accuracy either oscillates heavily or maintains the same at a very low rate.

4.5 Confusion Matrices

Figure 4 highlights the confusion matrices of the various PSO and GSA optimizers that we evaluated on the breast cancer dataset. We used the best optimizer performance of each optimizer for these confusion matrices. In terms of classifying each value correctly, we can see that the best at doing so are BPSO and PPSO. This follows a similar trend to the accuracy metrics shown in Figure 3, since PSO and PPSO are able to navigate past local optima and achieve better accuracy results.

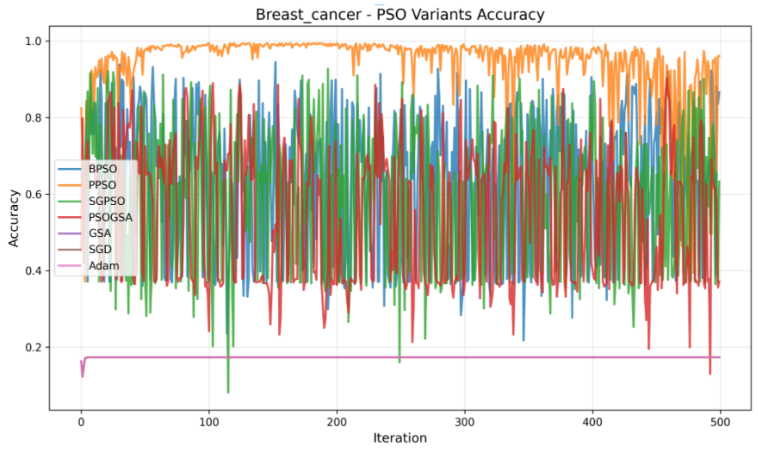

Figure 3: Accuracy comparison across optimizers and datasets on Breast Cancer Dataset

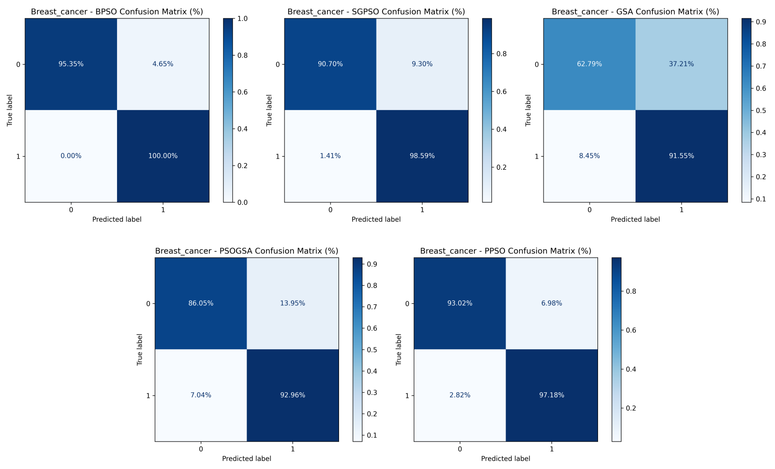

Figure 4: Confusion matrix comparisons across different optimizers on Breast Cancer Dataset

4.6 PPSO Hyperparameter Sensitivity

We conducted a sensitivity analysis on key hyperparameters of PPSO to understand their impact on convergence behavior and generalization. All experiments were performed on the Wine dataset using the small model, which contains a hidden layer with 32 neurons. The accuracy of the validation and the loss of training were tracked throughout the runs. The inertia strategy remained the same, in which we use the hyperbolic tangent scheme. We perform a study on the number of particles, the social coefficient, and the cognitive coefficient.

Number of Particles. In Figure 5, we varied the swarm size and observed that increasing the number of particles led to faster convergence due to broader exploration of the loss landscape. However, this came at the cost of increased computational time per iteration. Interestingly, validation accuracy peaked around 60 particles, suggesting diminishing returns beyond this point, likely due to over exploration or particle redundancy.

Cognitive Coefficient (c_1). We fixed the social coefficient to $c_2 = 1.7$ and swept c_1 in diverse increments. As shown in Figure 6, validation accuracy remained stable across the range $c_1 \in [0, 1.5]$, with performance degradation at higher values. This suggests that excessive reliance on personal best positions reduces swarm cohesion. The training loss curves were similar across all settings, indicating that convergence was not affected, but the generalization varied, as seen by the validation accuracy.

Social Coefficient (c_2). We fixed $c_1 = 1.5$ and varied c_2 within a specified range. We found that a purely social coefficient of zero yielded the worst validation accuracy, confirming the importance of information sharing among particles. As we can see in Figure 7, the accuracy improved to $c_2 = 2.0$, beyond which the performance decreased. For training loss, extremes (e.g. $c_2 = 0$, 0.01, 0.1, 4.0, 6.0) often led to poor convergence or local minima, while $c_2 = 1.6$ and 2.0 consistently produced smooth and stable descent.

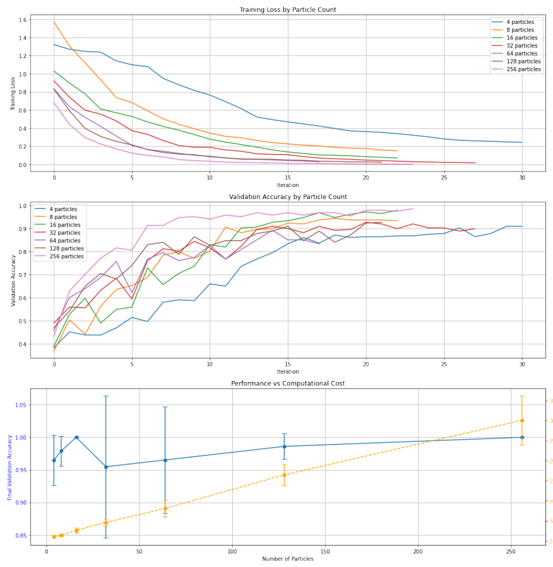

Figure 5: Hyperparameter sensitivity analysis for the Particles

4.7 Model Robustness

To evaluate robustness across model capacity, we trained FFNNs of increasing hidden layer sizes–small [16], medium [16, 32, 16], and large [16, 32, 64, 32, 16]–using PPSO, SGD, Adam. These experiments were performed without hyperparameter finetuning: all optimizer used their standard default settings. Moreover, the inertia strategy remained the same for PPSO, SGD had the PyTorch default step size, and Adam was configured to have PyTorch's default settings as well. We trained the different model sizes on the Wine dataset, under the same conditions: batch size of 32 and maximum iterations of 350.

Table 2 summarizes final accuracy and convergence behavior. On the small model, Adam and PPSO both reached perfect accuracy, while SGD failed to converge reliably. Moreover, PPSO was able to reach the global optimum much faster than the Adam. For the medium model, PPSO and Adam again achieved 100% accuracy, though PPSO required considerably more epochs to converge. However, unlike the small model, PPSO required slightly more iterations than Adam. For larger models, PPSO required considerably more iterations than Adam to converge properly. From the results, we can see that PPSO tends to take longer to converge on larger models because the dimensionality of the parameter space increases. PPSO particles must explore a significant more

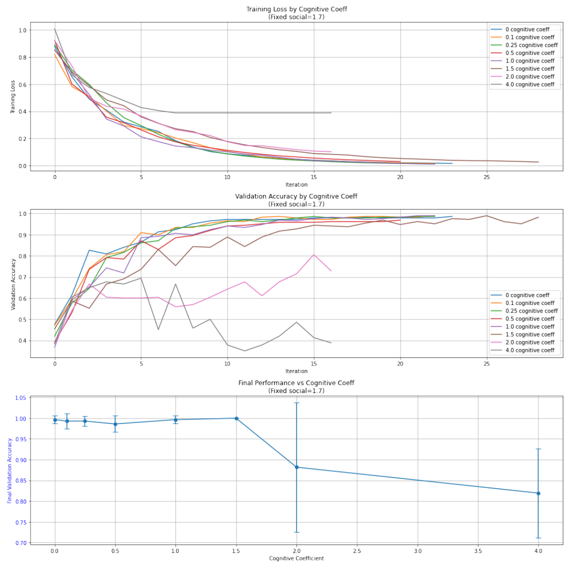

Figure 6: Hyperparameter sensitivity analysis for the Cognitive Coefficient

complex and higher-dimensional loss landscape, requiring more iterations to sufficiently sample promising regions. Moreover, PPSO relies on a stochastic, population-driven search, which is inherently slower in fine-tuning weights-—especially when precise adjustments are needed in deep architectures.

Table 2: Final accuracy and convergence epoch for each optimizer across model sizes.

Model Size	Optimizer	Accuracy	Converged Epoch
Small	SGD	0.6479	None
	Adam	1.0000	304
	PPSO	1.0000	52
Medium	SGD	0.4014	80
	Adam	1.0000	91
	PPSO	1.0000	97
Large	SGD	0.3169	0
	Adam	1.0000	47
	PPSO	1.0000	117

4.8 Reproducibility

The experiments present above can be reproduced with our code base, We open-sourced our code in the following github repo: https://github.com/SwaroopTha/PSO-for-NNs

5 Conclusion and Future Work

This work demonstrates the potential of Particle Swarm Optimization (PSO) and its variants as viable alternatives to traditional gradient-based optimizers for training neural networks, particularly in scenarios where gradient information is unreliable or difficult to compute. By implementing and benchmarking multiple PSO variants—including Basic PSO, Second Generation PSO, PSOGSA, and PPSO—we highlight their strengths in navigating complex, high-dimensional loss landscapes

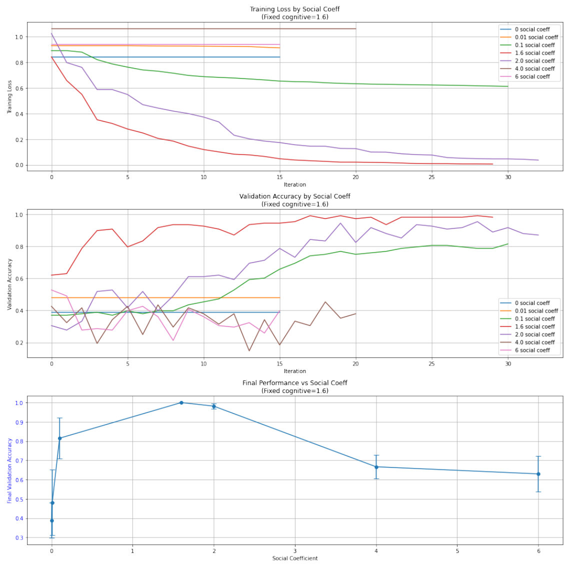

Figure 7: Hyperparameter sensitivity analysis for the Social Coefficient

with improved global search capabilities and robustness to initialization. Our comparative evaluation against SGD and Adam on standard classification tasks reveals that, while gradient-based methods remain efficient for well-behaved settings, PSO variants offer competitive or superior performance in challenging scenarios characterized by noisy gradients or limited data. Furthermore, adaptive strategies like PPSO help mitigate classical PSO's limitations such as premature convergence, enhancing both exploration and fine-tuning. Overall, our findings underscore the promise of PSO-based approaches in neural network optimization and encourage further research into hybrid and adaptive metaheuristic designs.

Contributions

- **Nikhil**: Implemented variant modifications; performed convergence analysis.
- **Swaroop**: Built codebase structure for all PSO variants; ran experiments for hyperparameter tuning.
- **Alan**: Setup neural network architecture; ran experiments on all datasets; compared results with SGD and Adam; generated visualizations.

References

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4, 1995.

[2] M. M. Millonas, "Swarms, phase transitions, and collective intelligence," 1993.

[3] B. A. Garro and R. A. Vázquez, "Designing artificial neural networks using particle swarm optimization algorithms," *Computational Intelligence and Neuroscience*, vol. 2015, no. 1, p. 369298, 2015.

[4] S. Mirjalili, S. Z. Mohd Hashim, and H. Moradian Sardroudi, "Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11125–11137, 2012.

[5] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "Gsa: A gravitational search algorithm," *Information Sciences*, vol. 179, no. 13, pp. 2232–2248, 2009. Special Section on High Order Fuzzy Sets.

[6] A. Nandi and N. D. Jana, "Accuracy improvement of neural network training using particle swarm optimization and its stability analysis for classification," *CoRR*, vol. abs/1905.04522, 2019.