Exploration of Bio-Inspired Algorithms

James Zhang-ly, Ayesha Rahman

Introduction

Bio-inspired optimisation algorithms have become essential for solving complex, non-convex problems in fields such as engineering, machine learning, bioinformatics, and finance. Inspired by natural phenomena—such as evolution, swarming, and predator-prey dynamics—these algorithms offer stochastic, population-based alternatives to traditional gradient-based methods [1]. Their strength lies in navigating rugged search spaces, avoiding local minima, and working effectively in the absence of reliable gradient information [2].

Over the past two decades, nature-inspired methods like Particle Swarm Optimisation (PSO) [3], Genetic Algorithms [5], and the more recent Marine Predators Algorithm (MPA) [4] have demonstrated strong performance on both benchmark functions and real-world tasks, especially when dealing with noisy, discontinuous, or high-dimensional problems.

In this project, we evaluate four algorithms: PSO, MPA, an LM-enhanced variant of MPA (LM_IMPA), and Adam (as a gradient-based baseline). While Adam is not bio-inspired, it is widely used in deep learning and offers a strong comparative baseline [11]. The algorithms were tested on six standard optimisation functions—Sphere, Rastrigin, Rosenbrock, Griewank, Ackley, and Schwefel [6] designed to represent diverse search landscapes. To extend our evaluation to a real-world application, we also applied these algorithms to a binary classification task on the Breast Cancer Wisconsin (Diagnostic) dataset [7], optimising directly on the weights of the neural network.

This report presents a detailed empirical comparison across three dimensions: convergence behaviour, execution time, and solution quality (fitness or classification accuracy). The motivation is grounded in the increasing relevance of hybrid and bio-inspired algorithms for black-box optimisation problems, particularly in AI applications where explainability, robustness, and global search capabilities are critical.

Background Research

Nature-inspired optimisation algorithms have gained significant traction over the past two decades due to their ability to solve complex, non-linear, and multimodal optimisation problems. Unlike traditional gradient-based methods, which often require smoothness, convexity, or differentiability of the objective function, bio-inspired algorithms rely on stochastic, population-based search strategies. This makes them more robust to noisy gradients, poor initialisation, and non-convex landscapes [8, 9]

One of the most influential is the **Particle Swarm Optimisation (PSO)** [3], introduced by Kennedy and Eberhart (1995). PSO is inspired by the social behaviour of birds and fish, each "particle" in the swarm adjusts its position based on personal experience and the experience of neighbouring particles, resulting in collective intelligence that often leads to effective exploration and exploitation of the search space. PSO has been widely used in areas such as feature selection, parameter tuning, and neural network training. [3]

More recent contributions to the field include the **Marine Predators Algorithm (MPA)** [4], proposed by Faramarzi et al. (2020). Inspired by the foraging behaviour of marine predators, the algorithm uses Brownian motion and Lévy flight to maintain a balance between exploration and exploitation. MPA is particularly noted for its ability to maintain exploration even in high-dimensional or rugged landscapes, and studies have shown it to outperform several classical optimisers on a variety of benchmark problems [4].

Extensions like the **Levenberg–Marquardt Improved MPA** [13], combine the Levenberg-Marquardt (LM) algorithm to find an initial sub-optimal solution, then further refined by the global search ability of the Improved Marine Predator Algorithm (IMPA). It minimises the disadvantages of finding a starting point, while including the strength of escaping local minima. Hybridisation is a common strategy in recent literature, where the strengths of multiple algorithms are combined, often yielding improved convergence rates and greater stability. [10].

Adam, while not a bio-inspired algorithm, is a widely used gradient-based optimiser for neural network training and serves as a strong baseline due to its momentum-based updates and adaptive learning rates. It is especially useful for sparse gradients and is known for its reliable convergence behaviour, making it the optimizer of choice in many deep learning frameworks.

To analyse and compare these algorithms, a standard suite of mathematical test functions was used. These functions simulate diverse optimisation landscapes and are designed to test specific capabilities of

optimisation algorithms. The ones used in this paper are the **Sphere function,** which is unimodal and convex, serving as a baseline function for measuring convergence speed. **Rastrigin** and **Ackley** functions, which are multimodal; they test the algorithm's ability to escape local minima. The **RosenBrock function**, which is a valley, challenges optimisers due to its ill-conditioning and slow convergence toward the global minimum. The **Griewank and Schwefel,** which are large landscapes that the model has to balance exploration-exploitation effectively to find the minimum. These functions are commonly used in literature to evaluate bio-inspired algorithms and to showcase their strengths, weaknesses and robustness. The formulas are detailed in Appendix 1.

In addition to these benchmarking functions, bio-inspired algorithms have been applied to many different areas, including classification, neural network training, protein structure prediction and kinematic calibration. Despite the different domains, the main idea for all of these tasks remains the same: to minimise error. In this study, we selected the **Breast Cancer Wisconsin Diagnostic dataset** as a practical real-world scenario. This binary classification task is a common diagnostic challenge that serves as an effective way of testing bio-inspired algorithms on a real-world scenario and to compare the effectiveness of the algorithms against a well-established gradient descent-based optimiser- Adam.

Methodology

In this study. We have evaluated four algorithms, P**article Swarm Optimisation, Marine Predator Algorithm, Levenberg-Marquardt Improved MPA (LM-IMPA), Adam**. These have been tested on both synthetic benchmark tests and a real-world binary classification task. This setup allowed us to assess each algorithm's convergence behaviour, computational efficiency, and generalisation ability across diverse problem landscapes.

Optimisers:

All four optimisation algorithms were implemented based on their papers in Python:

- pso.py: Particle Swarm Optimisation
- mpa.py: Marine Predators Algorithm
- lm_impa.py: Levy-based MPA hybrid with memory and adaptive coefficients
- adam.py / adam_torch.py: Two implementations, one as a standalone which optimises on the weights directly, whereas the torch version utilises PyTorch for backpropagation to optimise the weights.

All algorithm implementations are supplied with the following variables - A fitness function, Problem dimensionality, Population size, Number of iterations and Bounds. This design allowed seamless integration into a shared benchmarking pipeline.

Benchmark Functions

In addition to six standard continuous optimisation functions, we also evaluated the algorithms on a discrete list-sorting task inspired by Hillis's coevolutionary setup [12], implemented without coevolution. The goal was to sort a shuffled list by minimising the number of out-of-order pairs. This task tests the optimiser's robustness in sparse, combinatorial spaces where gradients or local continuity are absent. These results have been detailed in the Evaluation section.

Function	Properties	Challenge
Sphere	Unimodal, convex, separable	Measures pure convergence speed
Rastrigin	Multimodal, separable	High local minima count
Rosenbrock	Non-separable, curved valley	Ill-conditioning; slow convergence
Ackley	Multimodal, exponential decay	Tests exploration vs. exploitation
Griewank	Oscillatory, non-separable	Complex, repetitive search space
Schwefel	Rugged, deceptive global optimum	Penalises overshooting

Execution Pipeline

The experiments were executed from a shared file, which managed configurations. Each algorithm was executed independently on each benchmark function for 10 trials to account for stochasticity. The fitness values at every iteration were logged, and the average fitness and variance were plotted. This enabled visual comparison of both convergence speed and stability.

Assumptions and Design Decisions

To ensure fair comparison and reproducibility, we made the following experimental assumptions:

- Fixed dimensionality (30): All benchmarks were tested in 30D space to balance problem complexity and runtime.
- Population size of 50: Each algorithm operated with 50 agents/particles/predators to ensure equal exploration capability.
- Max iterations = 500: All algorithms were capped at 500 iterations per run, regardless of early convergence.
- Black-box assumption: Apart from the Adam_torch, no gradient information was supplied, each optimiser tried to minimise the fitness function by directly tuning the weights.
- Stochastic initialisation: Each run used a unique random seed to simulate real-world variance, but seeds were fixed to 42 for repeatability.
- No local tuning: No algorithm-specific hyperparameter tuning was applied beyond the standard or published defaults, ensuring neutrality in evaluation.

These assumptions created a level playing field, allowing us to attribute observed differences in performance solely to algorithmic design rather than experimental bias.

Discrete Sorting Task: Hillis-Style ListSort [12]

To evaluate optimiser performance on **discrete, combinatorial landscapes**, we implemented a variant of the **Hillis-style list sorting problem**, inspired by Hillis (1990) without coevolution. This test challenges algorithms to evolve a sorting strategy capable of reducing disorder in randomly shuffled lists using only black-box, population-based search.

Each candidate solution encodes a **swap decision matrix**: a flattened binary matrix that specifies whether to swap any two adjacent values based on their current positions. The fitness of a candidate is evaluated as the average number of inversions (i.e., out-of-order pairs) remaining after attempting to sort five randomly shuffled lists using the candidate's strategy.

This task is non-differentiable, sparse, and lacks continuity, making it a useful benchmark for assessing generalisation and exploration in discrete search spaces. Unlike continuous functions where smooth gradients guide optimisation, the ListSort landscape is dominated by flat regions punctuated by abrupt improvements, highlighting the need for effective global search.

Neural Network Weight Optimisation on Real-World Dataset

After establishing baseline performance on synthetic benchmarks, the second part of our evaluation focused on applying the optimisation algorithms to a real-world classification task: training a neural network on the Breast Cancer Wisconsin (Diagnostic) dataset. This was designed to assess each algorithm's performance in a data-driven scenario that mimics practical machine-learning workflows.

Instead of relying on traditional gradient-based training via backpropagation, each optimiser was tasked with directly tuning the weights of a feedforward neural network. The goal was to minimise binary cross-entropy loss on the training set while maximising classification accuracy on a held-out test set. This setup allowed us to benchmark not just convergence speed, but also generalisation performance.

Dataset Description

We used the load_breast_cancer() dataset from scikit-learn [7], which contains:

- 569 instances
- 30 numerical features per instance
- Binary classification target (Malignant = 0, Benign = 1)

The dataset was standardised using StandardScaler, and then split into 80% training and 20% test subsets.

Neural Network Architecture

A single hidden-layer feedforward neural network was implemented inside the Breast Cancer pipeline. The architecture was deliberately kept simple to focus on optimisation quality rather than network depth:

- Input layer: 30 neurons (1 per feature)

- Hidden layer: 1 layer with 5 neurons (ReLU activation)
- Output layer: 1 neuron (sigmoid activation)
- Activation function: Sigmoid (clipped between -100,100 to stop vanishing or exploding gradients)

The model was manually implemented using NumPy—no automatic differentiation or backpropagation was used. All weight matrices and biases were flattened into a 1D vector, forming the solution vector for the optimiser. The binary cross-entropy loss for a given weight vector and computed accuracy on the test set using the best weights was noted.
Extending this problem, we aimed to enable the algorithm to autonomously derive its own neural network architecture. To achieve this we designed a custom genome representation defined as:

[hidden nodes1, hidden nodes2, weights], bounds = [(1,20),(0,20),(-5,5)]

Since the genome structure is dependent on the neural architecture, its length varies dynamically based on the number of hidden layers and corresponding weights and biases. To ensure compatibility with the optimisation algorithms which require fixed dimensional search space, the dimensions of the problem were set to the maximum genome size (1063) calculated using the largest allowed architecture (2 hidden layers within 20 neurons each).
By using this genome, the algorithms can change the number of neurons in each layer between 1 and 20 and for the second hidden layer, if set to 0, the layer is removed. The evaluation function included clipping and rounding the hidden nodes to a discrete space; if the number of weights exceeded the number required by the chosen architecture values, these were ignored and dropped.

Algorithm Integration

Each algorithm was adapted to handle this real-world fitness function by directly optimising over the weight vector. Modifications included:

- Fixing the dimensionality of the search space to match the total number of trainable weights (calculated based on the architecture).
- Updating bounds to fit typical weight initialization ranges ([-5,5])
- Constraining runtime to 10 runs × 500 iterations × population size 50 followed as earlier.

Assumptions

- The dataset was not augmented or rebalanced; we assumed class distribution was representative.
- No early stopping or regularisation was applied to keep conditions equal across optimisers.
- The fitness function did not return gradients; for adam.py, gradients were approximated numerically using finite differences (estimate_gradient()).
- The PyTorch version (adam_torch.py) served as an upper baseline using standard training with backpropagation; this had one hidden layer with 5 nodes.

The real-world example extended the benchmark-only study to a practical learning problem. By bypassing backpropagation, we were able to test how well bio-inspired algorithms could optimise neural networks in black-box conditions. This also revealed how each algorithm handled real-world challenges such as:

- Generalisation to unseen test data
- Sensitivity to rugged or non-convex loss surfaces
- Scalability with respect to parameter dimensionality

The results from this phase are detailed in the next section, where we compare convergence patterns, fitness values, and final classification accuracy across all optimisers.

Evaluation & Results

Benchmark Based Evaluation

Convergence curves (averaged and shaded by standard deviation) allowed comparison of speed, stability, and robustness. The results are detailed in Appendix 2.
For consistency across the project, function representations were decided as follows:

- LM_IMPA - Black triangle
- PSO - Blue circle
- MPA - Purple cross
- Adam - Orange square
- Adam_Torch - Green diamond

Although Adam and Adam_Torch are plotted, we will be comparing the bio-inspired algorithms first and will focus on the comparison of the gradient-based optimisers in the key insights.

Sphere *(Unimodal, Fig. 1)*: LM_IMPA achieved fitness of zero from the first iteration. PSO converged faster than MPA, though eventually, all three achieved optimal fitness.

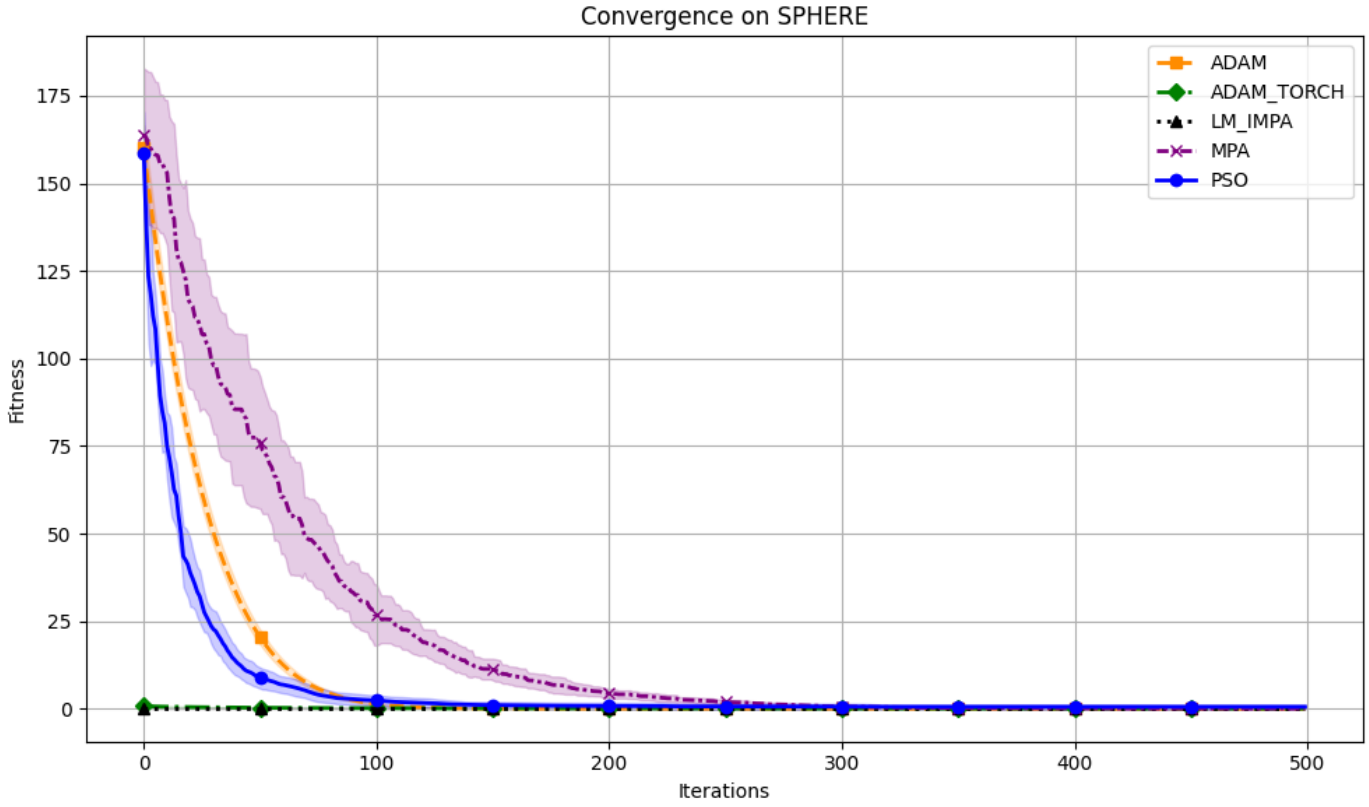

Figure 1 - Sphere function

Rastrigin *(Multimodal, Fig. 2)*: LM_IMPA achieved the lowest fitness; PSO plateaued early; MPA has a high variance and seems to struggle to converge.

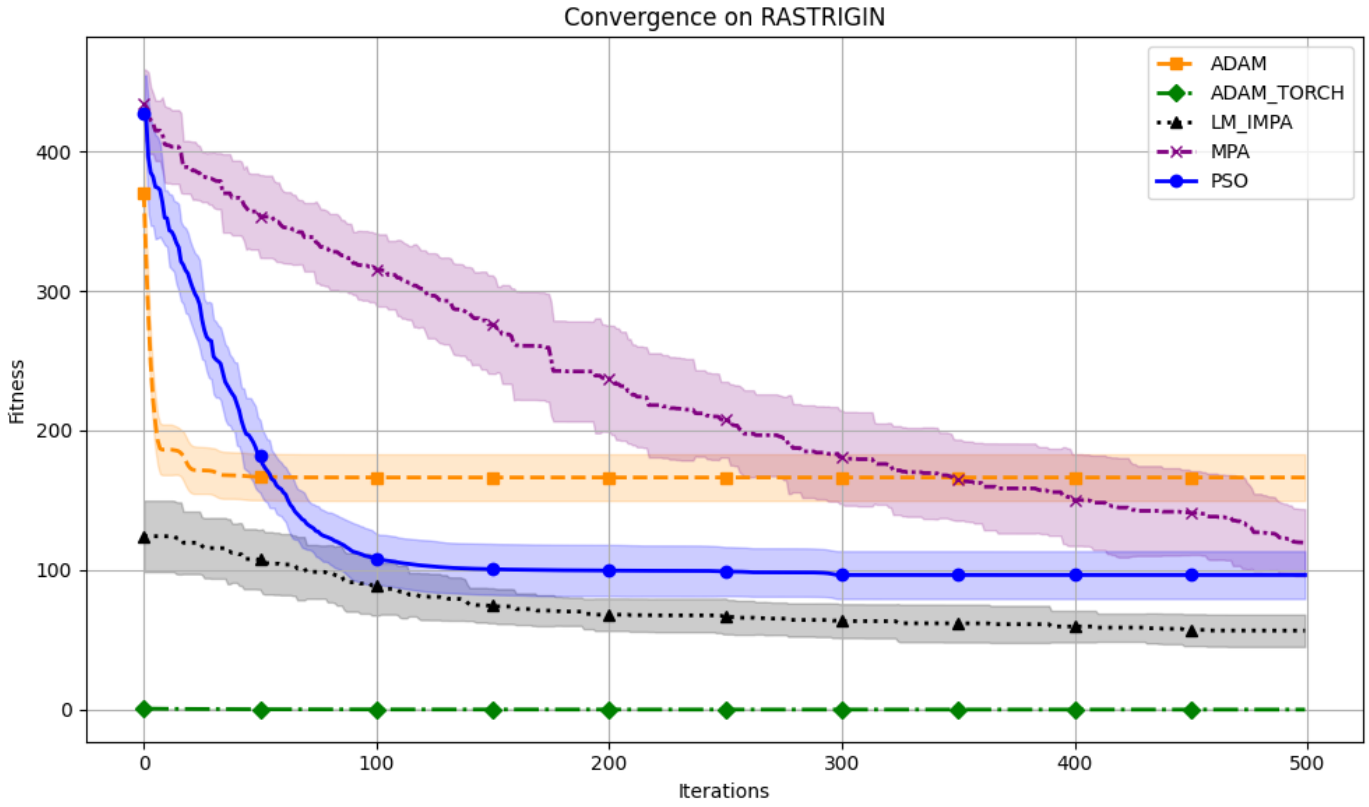

Figure 2 - Rastrigin function

Rosenbrock *(Curved Valley, Fig.3)*: LM_IMPA is optimal from the beginning; PSO showed stable convergence; MPA has more variance, but again all three achieved optimal fitness.

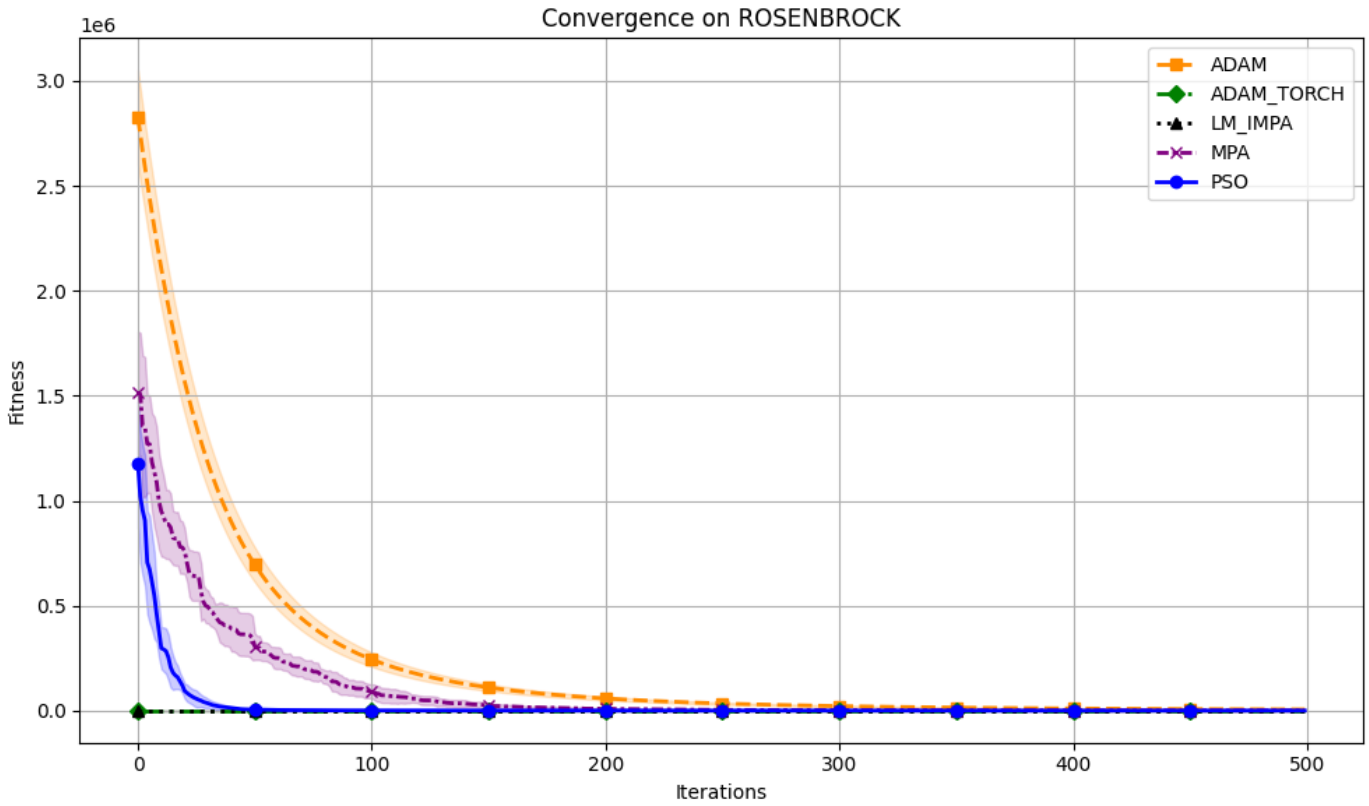

Figure 3 - Rosenbrock function

Ackley *(Multimodal w/ Decay, Fig. 4)*: PSO converges faster compared to the others; LM_IMPA progressed steadily, but both had high variance; while MPA achieved the best fitness and consistent results.

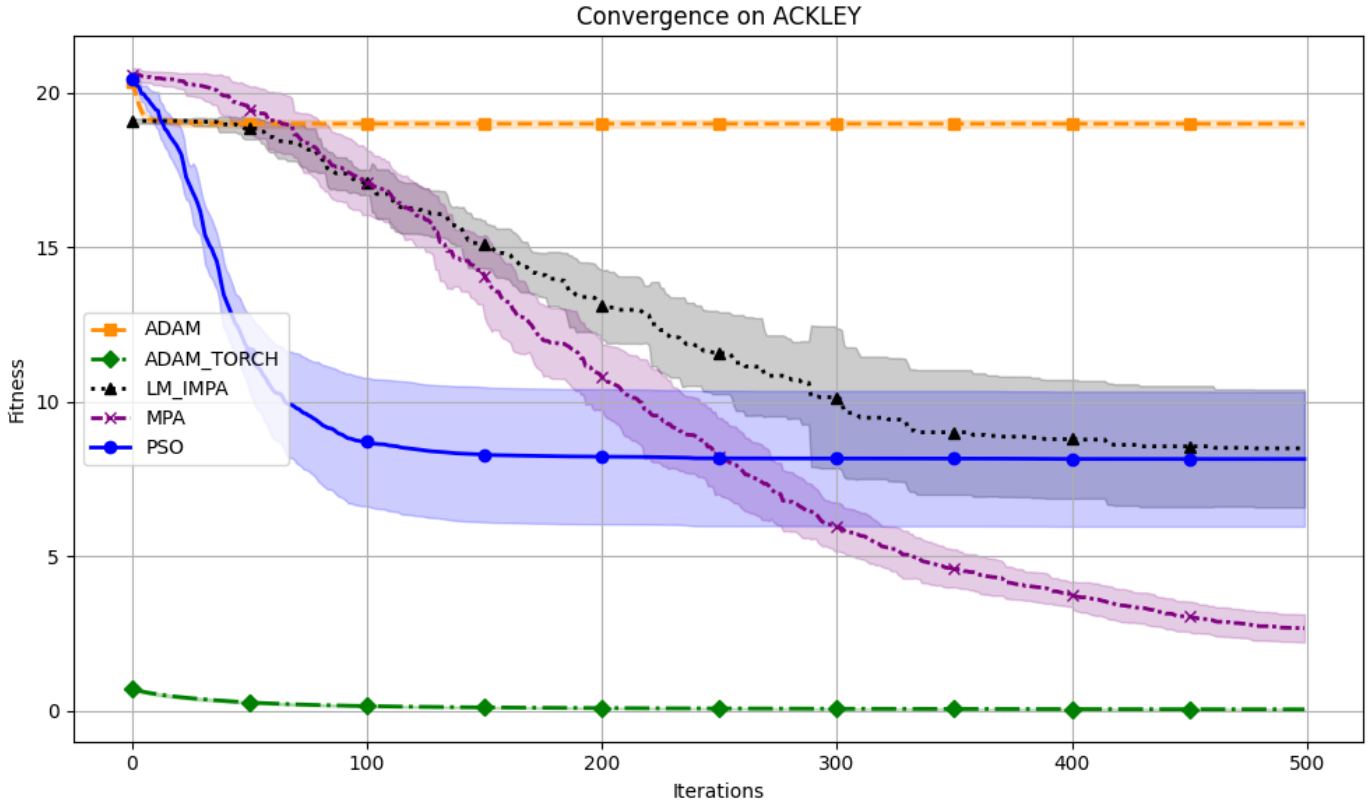

Figure 4 - Ackley function

Griewank *(Oscillatory Fig.5)*: LM_IMPA achieved optimal results; PSO and MPA both converged to optimal fitness, with PSO being faster.

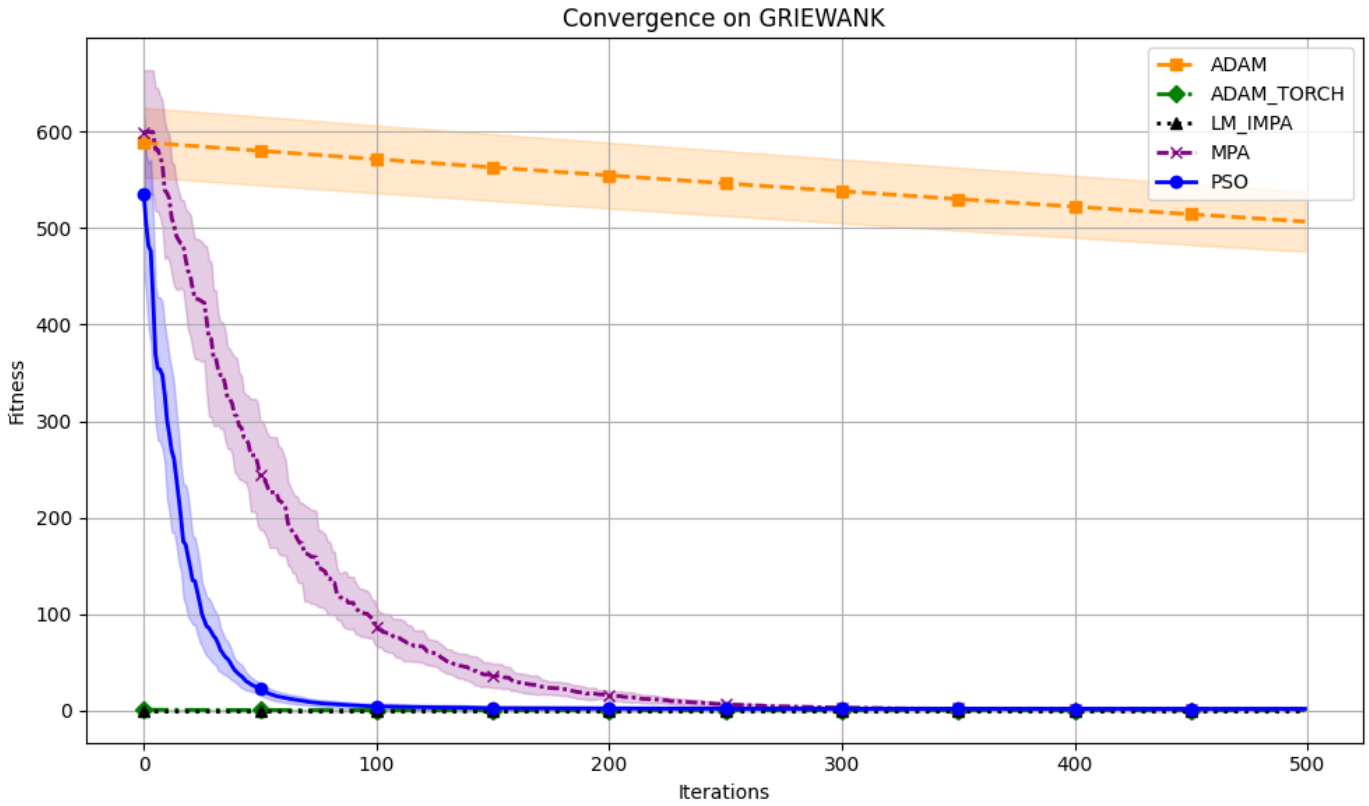

Figure 5 - Griewank function

Schwefel *(Deceptive Global Optimum, Fig.6)*: LM_IMPA found the smallest fitness value, while MPA and PSO, although MPA converging slower, achieved similar fitness values.

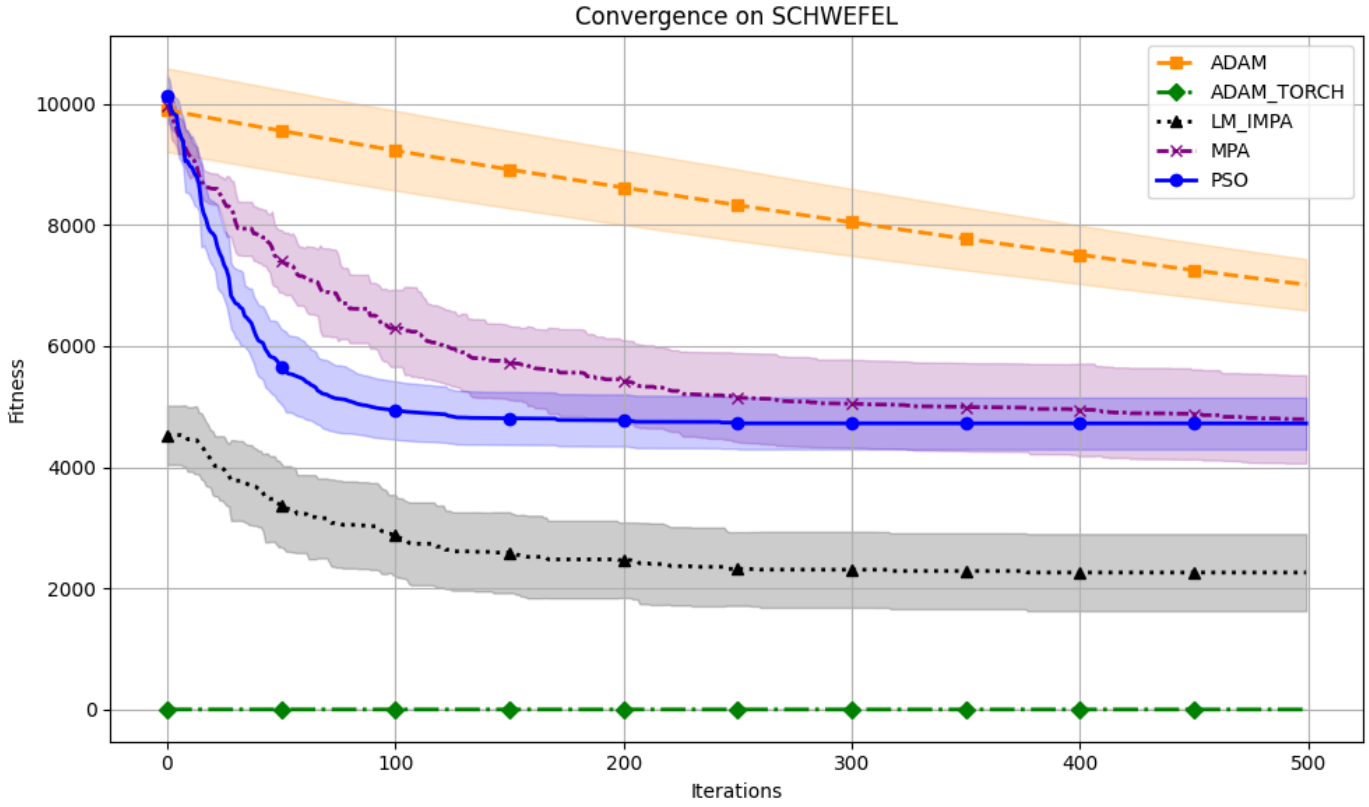

Figure 6 - Schwefel function

Key Insights:

- Adam_torch consistently achieved near zero or zero fitness early in the optimisation process. In contrast, our implementation (gradient-free optimisation using numerical estimation) struggles in many functions. Our implementation optimised the weights directly whereas the Torch version was implemented as a neural network and found the weights using backpropagation. This highlights the importance of gradient information when using a gradient descent-based algorithm such as Adam.

- LM_IMPA combines the strengths of local search and population-based strategies, making it versatile across different benchmarks (like Sphere, Rosenbrock, Griewank) as it starts at a lower fitness, indicating that it has a competitive edge over pure optimisers like MPA.
- MPA performs better on complex multimodal functions like Ackley, implying its ability to escape local minima. It has a higher variance, especially on deceptive or rugged functions (like Rastrigin and Schwefel) indicating sensitivity to initialisation and a need for more guided search.
- PSO demonstrates fast convergences however often does not achieve or plateaus before reaching optimal results, highlighting a potential lack of exploitation in later stages.

This showcases the effectiveness of hybridising MPA with the Levenberg-Marquardt algorithm, improving the initial starting points for the genetic algorithm. It demonstrates that bio-inspired algorithms can achieve results similar to gradient-based optimisers in specific optimisation landscapes. Finally, the contrast in performance of Adam and Adam Torch showcases the importance of algorithmic design and utilising the algorithm in the intended way.

Hillis-Style ListSort

The results show stark contrasts between optimisers in terms of convergence and final performance (*Fig.7*):
Both MPA and LM_IMPA achieved an optimal fitness value, both having similar convergence patterns and variances. Adam_Torch was able to start at a lower fitness, however, achieved optimal results in a similar number of iterations to MPA and LM_IMPA.
PSO was the worst at this problem and plateaued early at local minima.
The ListSort task effectively distinguishes optimisers not just by their ability to explore, but also by their ability to generalise symbolic logic across examples without explicit gradient information.

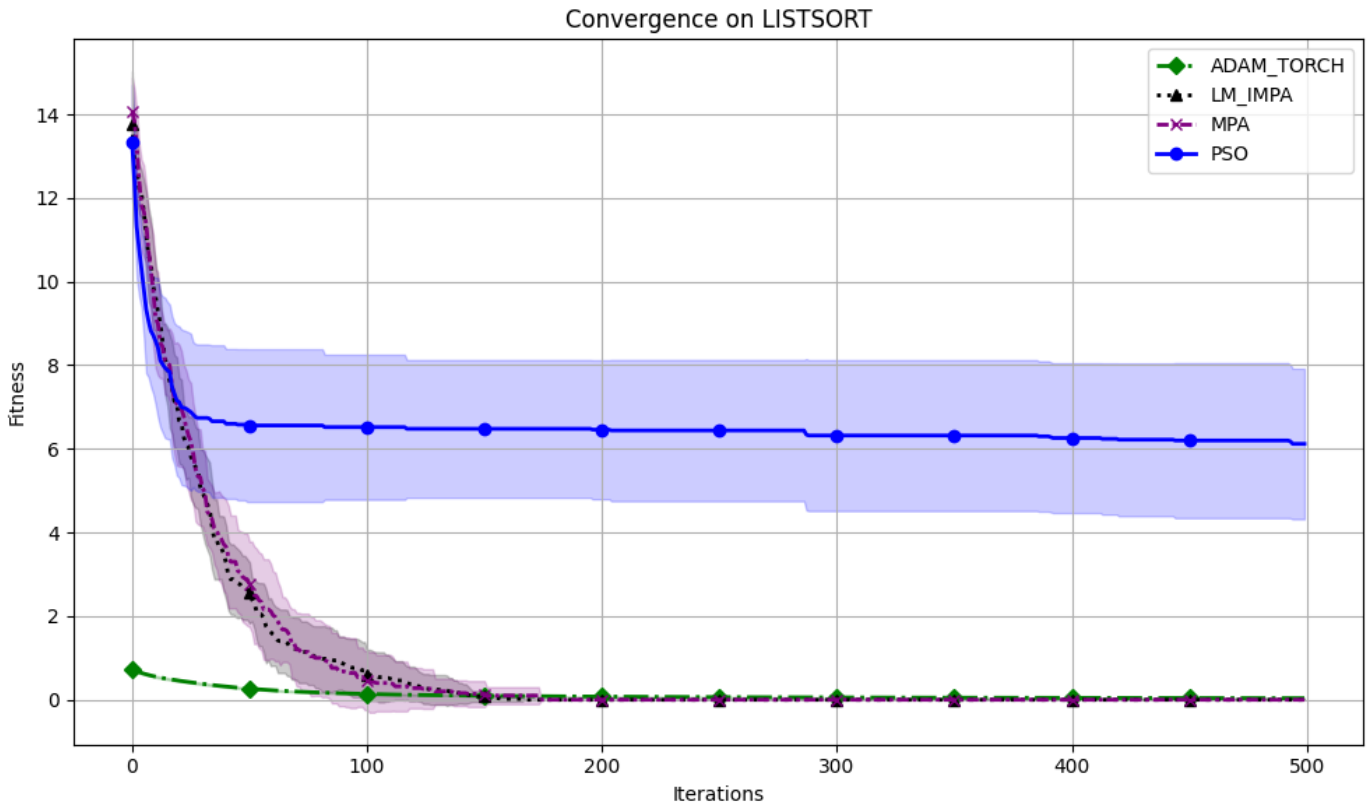

Figure 7 - Hillis style ListSort

Real-World Neural Network Task

The convergence behaviour for the fixed neural network across 500 iterations is shown here (*Fig.7)*
Key insights

- **PSO** was slightly faster at converging compared to MPA and LM-IMPA but had similar final scores.
- **MPA** and **LM-MPA** had very similar performances, likely indicating that the underlying fitness landscape of this classification problem was not rugged or complex, without these the LM-MPA optimiser cannot showcase the benefits of the hybridisation.
- Bio-inspired optimisers outperformed ADAM_torch, in terms of convergence speed, final fitness and variance across runs.

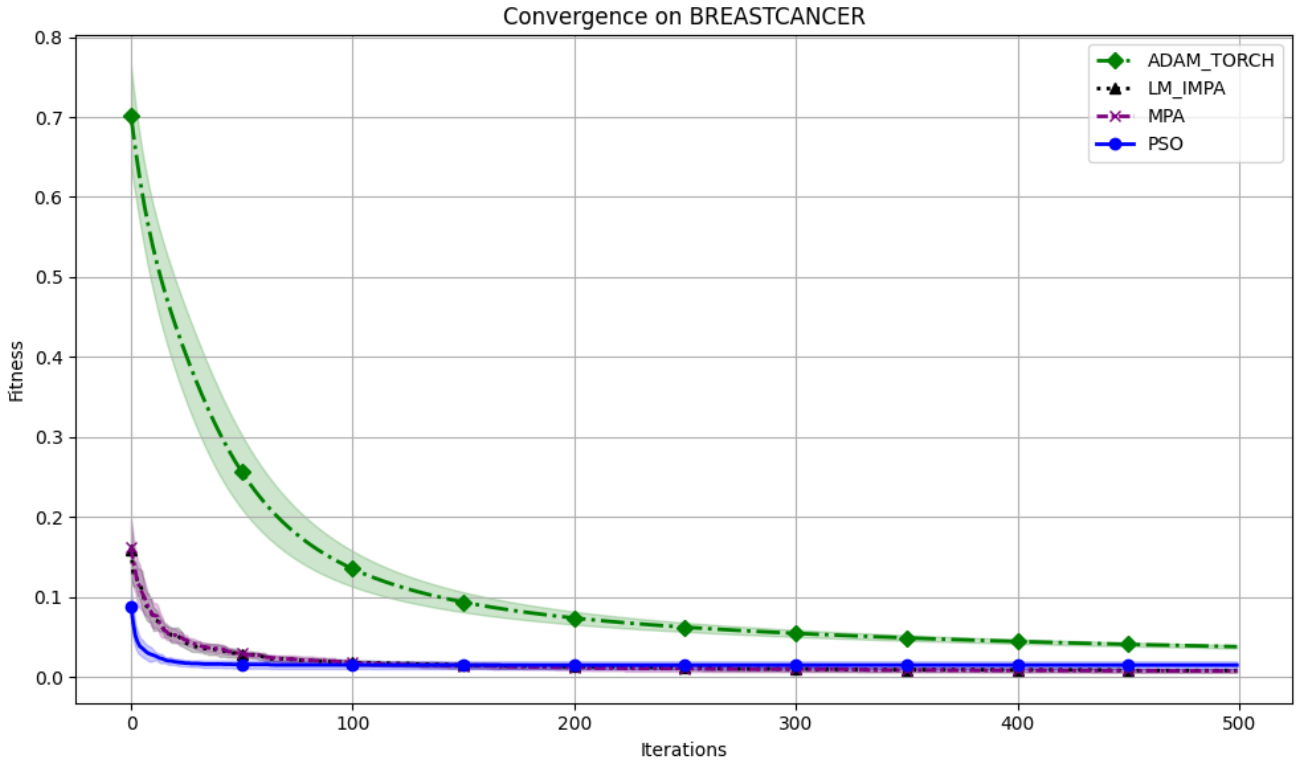

Figure 8 - Fixed architecture for Breast Cancer classification

As mentioned, the implementation was then extended to include automatic neural network architecture derivation (Fig 9). The table below showcases the architecture chosen by the algorithms after training. (Adam_Torch, was not extended in this problem having a fixed architecture)

Algorithm	Hidden node 1	Hidden node 2
MPA	2	0
PSO	9	20
LM_IMPA	20	0

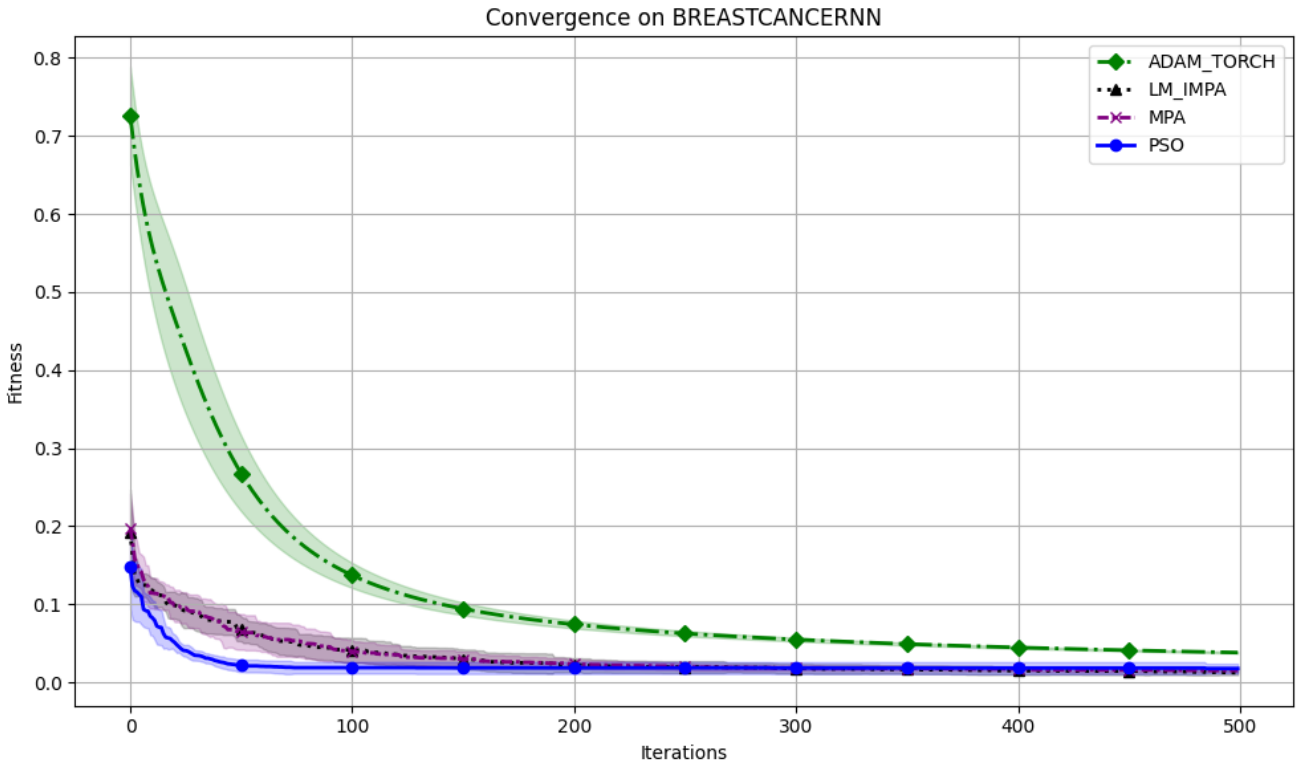

Figure 9 - Neural network architecture for breast cancer classification

The results for this implementation are similar to the fixed neural network implementation, however looking at the chosen architecture some observations can be made. MPA converged with the simplest architecture, choosing only two hidden nodes. PSO evolved to a large neural network with 9 hidden nodes in the first layer and 20 nodes in the second, this may indicate that the algorithm had found more features to optimise for. Finally, LM_IMPA evolved to have 20 nodes and had similar results to MPA.
This task demonstrates that, provided that the genome encoding supports it, a bio-inspired algorithm can simultaneously find a neural network architecture and minimise the weights, providing an alternative to manual neural network designs used in standard machine learning.

Limitations & Future Work

Limitations:

- Although benchmarking is an effective way of showcasing the advantages or disadvantages of optimisers, they might not mimic real-world problems, which have noisy, dynamic or ever-changing fitness landscapes. Consequently, results may not generalise directly to applied domains.
- Both bio-inspired algorithms and the Adam variants were evaluated using default or literature-recommended hyperparameters. This may have limited their performance potential. Optimising hyperparameters might improve performance across benchmarks and the real-life dataset. Especially with Adam, optimising the hyperparameters might give equal performance values to the bio-inspired algorithms.
- The fitness landscape for the classification task, might be too simple to give opportunities to algorithms like LM-IMPA to fully demonstrate their advantages.

Future Works:

- Testing on more complex datasets or more complex problems, such as kinematic calibration, harder classification tasks such as CIFAR-10 or MNIST. These more complex problems might showcase the advantages or disadvantages of some bio-inspired algorithms and might showcase that gradient-based methods might be better in some scenarios.
- Investigating more bio-inspired or hybrid algorithms for example: Honey Badger Algorithm, Grey Wolf Optimizer, Coyote Optimization Algorithm, and many more hybrid algorithms and gradient-based methods such as SGD or ADAGrad, could broaden the comparison and uncover niche strengths.
- In our real-world example, the optimisers were modifying the weights and model shape together. However there was no motivation to get to a smaller neural network shape, either the fitness function could be adjusted to include minimising the number of parameters or the objectives split into a multi-objective function, this requires changing the algorithms to accept multiple optimisation objectives, however, it might showcase a trade-off between accuracy and simplicity.
- Leveraging tools like Optuna or Hyperopt for systematic hyperparameter search could significantly boost optimizer performance and uncover optimal configurations for different types of problems.
- Evaluating optimizers under perturbed input data, noisy gradients, or adversarial examples could help assess their resilience. Understanding generalization capabilities under uncertainty is critical for deployment.

Conclusion

This study conducted a comparison of several optimisation algorithms - **Particles Swarm Optimisation (PSO), Marine Predator Algorithm (MPA), Levenberg Marquardt Improved MPA (LM-IMPA)** and **Adam** - across synthetic benchmark functions and a real-world neural network training task.
The results from the benchmark functions demonstrated LM-IMPA consistently outperformed other optimisers in deceptive landscapes. PSO exhibited strong early-phase convergence, while MPA showed reliable but slower performance. The LM-IMPA hybrid strategy showcases how hybridisation can leverage both algorithms to improve performance.
In the real-world classification tasks, all bio-inspired algorithms surpass Adam in convergence speed and final loss. However, the similar performance of LM-IMPA and MPA indicates that the underlying fitness function might have lacked the complexity to showcase the hybridisation advantage.
These findings reinforce previous papers that showcase the suitability of bio-inspired algorithms compared to a traditional gradient-based one; future work will focus on testing algorithms on more complex real-world tasks as well as showcasing the impact of hyper-parameter tuning and multi-objective tasks.

References

1. Yang, X.S., 2020. *Nature-inspired optimization algorithms*. 2nd ed. London: Academic Press. Available at: https://books.google.co.uk/books?hl=en&lr=&id=La_YDwAAQBAJ
2. Fister, I., Fister Jr, I., Yang, X.S. and Brest, J., 2013. A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*. Available at: https://i2pc.es/coss/Docencia/SignalProcessingReviews/Fister2013.pdf
3. Kennedy, J. and Eberhart, R., 1995. Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. IEEE, Vol. 4, pp.1942–1948. Available at: https://ieeexplore.ieee.org/abstract/document/488968
4. Faramarzi, A., Heidarinejad, M., Mirjalili, S. and Gandomi, A.H., 2020. Marine predators algorithm: A nature-inspired metaheuristic. *Expert Systems with Applications*, 152, p.113377. Available at: https://www.sciencedirect.com/science/article/pii/S0957417420302025
5. Forrest, S., 1996. Genetic algorithms. *ACM Computing Surveys (CSUR)*, 28(1), pp.77–80. Available at: https://dl.acm.org/doi/pdf/10.1145/234313.234350
6. Surjanovic, S. and Bingham, D., 2013. Virtual Library of Simulation Experiments: Test Functions and Datasets. [online] Simon Fraser University. Available at: https://www.sfu.ca/~ssurjano/optimization.html
7. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, pp.2825–2830. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
8. Yang, X.S., 2010. *Nature-inspired metaheuristic algorithms*. 2nd ed. Frome, UK: Luniver Press. Available at: https://books.google.co.uk/books?hl=en&lr=&id=iVB_ETlh4ogC
9. Fister Jr, I., Yang, X.S., Fister, I., Brest, J. and Fister, D., 2013. A brief review of nature-inspired algorithms for optimization. *arXiv preprint* arXiv:1307.4186. Available at: https://arxiv.org/abs/1307.4186
10. Yang, X.S., 2013. Metaheuristic optimization: Nature-inspired algorithms and applications. In: X.S. Yang, ed. *Artificial intelligence, evolutionary computing and metaheuristics: In the footsteps of Alan Turing*. Berlin: Springer Berlin Heidelberg, pp.405–420. Available at: https://link.springer.com/chapter/10.1007/978-3-642-29694-9_16
11. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint* arXiv:1412.6980. Available at: https://arxiv.org/abs/1412.6980
12. Hillis, W.D., 1990. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1–3), pp.228–234.
13. Feng, A., Zhou, Y., Zhang, R., Zhao, W., Li, Z. and Zhu, M., 2025. A novel kinematic calibration method for robot based on the Levenberg–Marquardt and improved Marine Predators algorithm. *Measurement*, 243, p.116125. Available at: https://www.sciencedirect.com/science/article/pii/S0263224124020104

Appendix 1:

Benchmark functions:

Benchmark	Function	d used	Global minimum	Bounds
Sphere	f(x) = i=1dxi2	30	f(x) = 0 at x = (0,...,0)	[-5.5]
Schwefel	f(x) = 418.9829d - i=1dxisin(xi	30	f(x) = 0 at x = (420.9687,..., 420.9687)	[-500,500]
Rosenbrock	f(x) =i=1d-1[100(xi+1-xi2)2 +(xi-1)2	30	f(x) = 0 at x = (1,...,1)	[-5,10]
Rastrigin	f(x) =10d+i=1d[xi2 -10cos(2xi)]	30	f(x) = 0 at x = (0,...,0)	[-5.12,5.12]
Griewank	f(x) =i=1dxi24000-i=1dcos(x2i)+1	30	f(x) = 0 at x = (0,...,0)	[-600,600]
Ackley	f(x) = -a e-b1bi=1dxi2 -e1di=1dcos(cxi)+a+e1 Where a = 20, b = 0.2 and c = 2	30	f(x) = 0 at x = (0,...,0)	[-32.768, 32.768]

Functions, dimensions, bounds and minima from:

Surjanovic, S. & Bingham, D. (2013). Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved April 25, 2025, from http://www.sfu.ca/~ssurjano.

Appendix 2:

Benchmark Functions	Adam_torch	MPA	PSO	LM_IMPA
Sphere	0.035120	0.011338	0.001262	0.000000
Rastrigin	0.038324	93.438670	77.620627	41.788245
Rosenbrock	0.037776	89.619949	81.623672	0.000000
Ackley	0.038149	2.610640	4.157828	4.255200
Griewank	0.035335	1.060741	0.015026	0.000000
Schwefel	0.043047	3339.089989	4050.054007	1307.373241
BreastCancer	0.035482	0.004396	0.008791	0.006593
ListSort	0.036192	0.000000	3.600000	0.000000