Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

Optimisation for Machine Learning - 821

Comparative Study of Optimisation Algorithms for Feed-Forward Neural Networks

DJ Swanevelder - 25205269
Ben Cleveland - 25504843

June 1, 2025

Department of Applied Mathematics
Stellenbosch University

1 Introduction & Problem Statement

1.1 Context

Optimising neural networks involves minimising a loss function that is often non-convex and characterised by many local minima and saddle points. Traditional gradient-based methods such as stochastic gradient descent (SGD) and limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) rely on local gradient information, which can limit their ability to escape poor local minima. In contrast, population-based methods such as Genetic Algorithms (GAs) offer a gradient-free alternative with greater potential for global exploration. This creates a need to explore alternative optimisation strategies that can better cope with complex, high-dimensional loss surfaces.

1.2 Motivation

While gradient-based optimisers are widely used due to their efficiency and simplicity, they may struggle in complex loss landscapes where multiple local minima and plateaus are present. This can lead to suboptimal solutions, particularly in problems where global exploration is critical. This limitation highlights the importance of evaluating optimisers not only by their speed but also by their ability to reliably escape local optima. Genetic Algorithms, although more computationally intensive, provide a fundamentally different approach that may overcome some of these limitations. By comparing these methods, this study aims to highlight their respective strengths and weaknesses, and provide more evidence and clarity on the choice of optimiser for tasks involving difficult optimisation landscapes.

1.3 Problem Statement

This project aims to compare the performance of three optimisation algorithms: SGD, L-BFGS, and a custom Genetic Algorithm (GA). Each algorithm is used to train a feed-forward neural network (FFN) on a dataset with a multimodal, non-convex loss landscape. Given the limitations of traditional approaches in complex settings, this comparison aims to determine which method is most effective under such conditions. The objective is to evaluate how effectively each optimiser finds high-quality solutions, considering convergence speed, final accuracy, and computational cost. Their respective behaviours in navigating complex optimisation surfaces will also be analysed.

The study includes tuning of the FFN architecture, involving adjustments to the number of layers and neurons per layer. Additionally, hyperparameter tuning is conducted for each optimiser to ensure a fair comparison. This includes parameters such as learning rate, batch size, and optimiser-specific settings. The goal is to identify configurations that maximise performance while maintaining reproducibility and efficiency.

2 Methodology

2.1 Data Preprocessing

The input and output data were standardised to have zero mean and unit variance. This was done to improve training stability and convergence. As a result, a Mean Squared Error (MSE) of 1 corresponds to the error of simply predicting the mean of the output. Therefore, achieving a loss below 1 indicates that the model has learned meaningful patterns.

2.2 GA Design

2.2.1 Chromosome

A continuous Genetic Algorithm was adopted to directly optimise the real-valued weights and biases of the neural network. Each individual in the population encodes a full set of network parameters as a single flattened vector of floating-point values. This vector is decoded by reshaping the values back into their respective weight and bias tensors, according to the architecture of the feed-forward neural network. This approach avoids the inefficiencies of binary encoding and is better suited for problems with continuous domains, such as neural network training. Since the dataset involves continuous input and output values, and the parameters themselves are inherently real-valued, this representation is both natural and efficient.

Chromosome: Continuous GA.

2.2.2 Fitness Measure

The fitness of each individual was evaluated using the negative Mean Squared Error (MSE) computed over the validation set. MSE is commonly used in regression tasks due to its sensitivity to large errors, which cause a quadratic increase in the loss value. This property is beneficial in evolutionary settings because it amplifies the penalty for individuals whose parameters are far from optimal, making poorly performing candidates easier to distinguish and remove during selection. Additionally, MSE provides a more smooth and consistent fitness landscape, which supports more stable convergence across generations.

Fitness: Mean Squared Error.

2.2.3 Population

A fixed population size was used to simplify experimental control and ensure consistent comparisons across different runs. Introducing dynamic population sizes adds complexity and can make it difficult to isolate the causes of unexpected or undesired behaviour. Fixing the size made the system easier to debug and analyse.

Population Size: Fixed 200.

For initialisation, a grid search was conducted over several standard neural network weight initialisation methods. Xavier normal initialisation consistently produced the best results in terms of early validation performance. A grid search plot is included to illustrate the relative performance of the methods evaluated.

To encourage better exploration of the loss landscape, several alternative initialisation methods were tested to increase the diversity and spread of the initial population. These included:

- *Scaled Xavier*: Applied a multiplicative scalar to Xavier-initialised weights to increase variance.
- *Uniform*: Sampled all weights uniformly from a wide range of values.
- *Opposition Based*: For each random initial weight, an “opposite” weight was also sampled and used, aiming to cover both ends of the search space.

Despite their theoretical promise, all three of these alternatives led to poor early convergence and unstable behaviour.

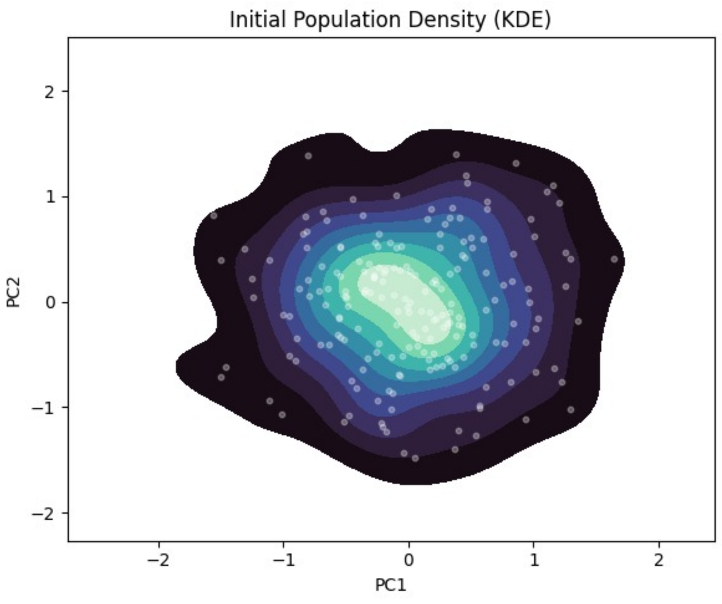

Figure 1: PCA visualisation of initial population distribution using Xaiver Moral.

The standard Xavier normal method provided well-scaled and diverse initial weights as seen in Figure 1, that supported more stable early training.

Population Initialisation: Xavier normal.

2.2.4 Selection Operator

The goal of the selection strategy was to balance exploitation, favouring individuals with lower loss, and exploration, giving weaker individuals a chance to be selected. This helps avoid premature convergence and supports broader search of the loss landscape.

Initial experiments with roulette-wheel selection revealed excessive selection pressure, which favoured a small subset of individuals and caused early stagnation. Random tournament selection with a tournament size of 3 offered a better balance by preserving strong individuals while still allowing weaker candidates to propagate. Increasing the tournament size beyond 3 resulted in behaviour similar to roulette selection, reducing population diversity too rapidly.

Selection: Random tournament (size 3)

2.2.5 Crossover

Simulated Binary Crossover (SBX) was initially used due to its ability to simulate single-point crossover behaviour in continuous domains. However, it proved computationally expensive and often led to early convergence. A hybrid approach applying BLX early in training and switching to SBX later was tested, but no meaningful performance improvement was observed.

Blend Crossover (BLX) with a tunable α parameter was then evaluated. This allowed explicit control over the exploration range around parent genes. Higher α values increased exploration by producing offspring outside the immediate bounds of the parents, while lower values focused more on local refinement. Extremely high α values degraded performance by disrupting exploitation.

BLX with an appropriately chosen α provided the most flexibility and stable performance.

Crossover: BLX-$\alpha = 0.6$

2.2.6 Mutation

Normal perturbation was applied as the mutation operator, with both the mutation probability and scale tuned to balance exploration and preservation of fit individuals. A mutation probability of 0.01 and mutation scale of 0.01 were found to maintain good individuals while allowing exploration of potentially better solutions.

Mutation: $\mathcal{N}(0,1) \times 0.01$ **with probability** $P = 0.01$**.**

2.2.7 Pairing Strategy

Several pairing strategies were evaluated to balance exploitation and exploration. The Elite-Bad strategy paired elite individuals with 'bad' individuals with probability $1 - x$, while pure elite pairing occurred with probability x. The Elite-Random strategy combined elite individuals with random parents similarly, but with random instead of Bad. These more complex strategies did not converge effectively. Consequently, pure random pairing was selected for its simplicity and reliable convergence.

Pairing: Random

2.2.8 Replacement Strategy

To ensure convergence while maintaining exploration, the replacement strategy preserved 20% of the elite individuals automatically in each generation. The remaining 80% consisted of offspring generated through crossover, allowing exploration of new solutions.

Replacement: 20% elite, 80% offspring

2.2.9 Pseudo Code of Final Algorithm

Algorithm 1 High-Level Overview of GA + SGD Workflow

```
1: 1) Initialization
2: Set random seeds (NumPy, PyTorch)
3: Define GA hyperparameters

4: 2) Data Preparation & Model Utilities
5: Load $X_{train}, y_{train}, X_{val}, y_{val}$ as torch tensors
6: function BUILD_MODEL
7:     Return new FFN with Xavier-initialised weights
8: end function

9: 3) GA Operators
10: function TOURNAMENT_SELECT(pop, fitness)
11:     Return fittest of $k$ randomly selected genomes
12: end function
13: function CROSSOVER_AND_MUTATE(p1, p2)
14:     Perform BLX-$\alpha$ crossover and apply Gaussian mutation
15:     return new child genome
16: end function

17: 4) Initialize Population
18: for $i = 1$ to $POP\_SIZE$ do All individuals initialised using Xavier-Normal
19: end for

20: 5) Main Evolutionary Loop
21: for $gen = 1$ to $GENERATIONS$ do
22:     (a) Evaluate: compute train MSE for each genome $\rightarrow$ fitness
23:     Record best train MSE and validation MSE of best genome
24:     (b) Elitism: retain top $ELITE\_FRAC$ as elites
25:     if $gen$ mod $REFINE\_EVERY == 0$ then
26:         (c) Local Refinement: run SGD on elites for $REFINE\_EPOCHS$
27:         Update elite genomes with refined weights
28:     end if
29:     (d) Reproduction: fill new population via Blx crossover + mutation
30: end for
```

2.3 GA Hyper Parameter Tuning:

All hyper parameter tuning for the GA , as well as meta-strategy and design tuning, was covered in section 2.2. It included things like the tuning of the following: Blx-alpha parameter, tournament size, elitism percentage, mutation parameters, mutation probability, refinement frequency, refinement duration, number of generations, population size, replacement strategy, pairing strategy, breeding strategy, initialisation strategy, among others.

2.4 Hyper Parameter Tuning: LBFGS and SGD

The SGD optimiser was tuned using the Optuna hyperparameter optimisation framework with a fixed batch size of 1 to implement pure stochastic gradient descent. The search space was comprised of a log-uniform distribution for the learning rate over $[1 \times 10^{-5},\ 1 \times 10^{-1}]$, and a uniform distribution for the momentum parameter over $[0.0,\ 0.99]$. For each trial, a new pair ($\texttt{lr}, \texttt{momentum}$) was generated, a new 2×24 ReLU network with Xavier-normal initialisation was constructed, and then the model was trained for 10 epochs using SGD so as to get an idea of how appropriate it was for the problem. The final validation MSE was returned to Optuna for minimisation across 20 trials.

For the L-BFGS optimiser, Optuna was used to search over three hyperparameters: learning rate `lr`, maximum number of iterations per optimiser step `max_iter`, and the number of full-batch epochs `epochs`. The learning rate was generated from a log-uniform distribution in the range $[1 \times 10^{-2},\ 10]$, while `max_iter` was chosen from the categorical set $\{10,\ 20,\ 50\}$, and `epochs` from $\{20,\ 40,\ 60,\ 80,\ 100\}$. Each trial built a new 2×24 ReLU network with Xavier-normal initialisation, applied the chosen hyperparameters, and trained the model using full-batch L-BFGS with the closure function.

2.5 FNN Tuning

To determine a good FFN architecture, a manual grid search was performed over a predefined set of layer–unit configurations using vanilla SGD. All candidate architectures had ReLU activations and Xavier-normal weight initialisation, and were trained for 10 epochs with a learning rate of 0.01 and batch size of 32. The grid included the following configurations: $(1, 24)$, $(2, 24)$, $(1, 8)$, $(2, 16)$, $(2, 20)$, $(2, 8)$, and $(1, 16)$, covering both shallow and deep networks of varying width. Larger networks with more neurons and more layers were considered, but it turned out that given the computational cost of the number of GA experiments we intended to run, these were not very feasible.

3 Results

3.1 Hyper Parameter Tuning Results:

The results of the 20 trials were evaluated using the validation mean squared error (MSE). With the architecture fixed to a 2-layer, 24-unit ReLU network and Xavier-normal initialisation, the best-performing configuration used a learning rate of 4.87×10^{-4} and a momentum of 0.8766, with training performed using pure stochastic updates (batch size of 1). This configuration achieved a validation MSE of 0.7969 (during tuning: Not final run), representing a substantial improvement over default SGD parameters.

For L-BFGS, the results for the 20 trials were again compared using validation mean squared error (MSE). The best configuration—learning rate of 0.9669, $maxiter = 20$, and `epochs` = 100—achieved a significantly lower validation MSE of 0.1841 (Again: During tuning).

3.2 Final FFN architecture:

Architecture tuning was performed via grid search using vanilla SGD on a fixed training protocol. Among several layer–unit combinations, the 2-layer, 24-unit architecture attained the lowest validation MSE of 0.9876, marginally outperforming its 1-layer counterpart. This was then the chosen architecture for our experiments.

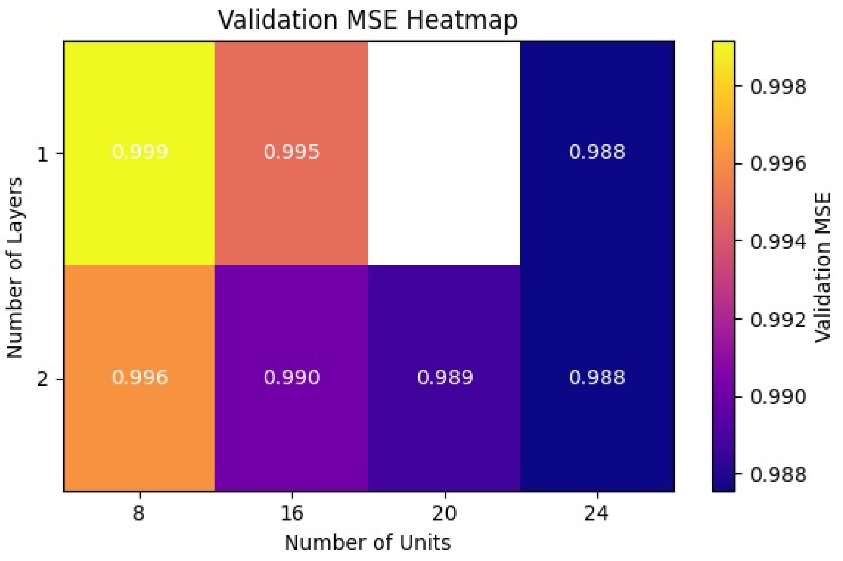

Figure 2: Grid Search results for FFN Architecture.

3.3 Learning Curves and Convergence Plots For Each Optimiser:

4 Learning Curves and Convergence Analysis

To assess the training dynamics and convergence behaviour of each optimiser, we present the learning curves showing the training and validation mean squared error (MSE) over epochs (or generations). These plots help visualise the stability, convergence speed, and generalisation performance of the models trained using different optimisation strategies.

4.1 SGD Optimiser

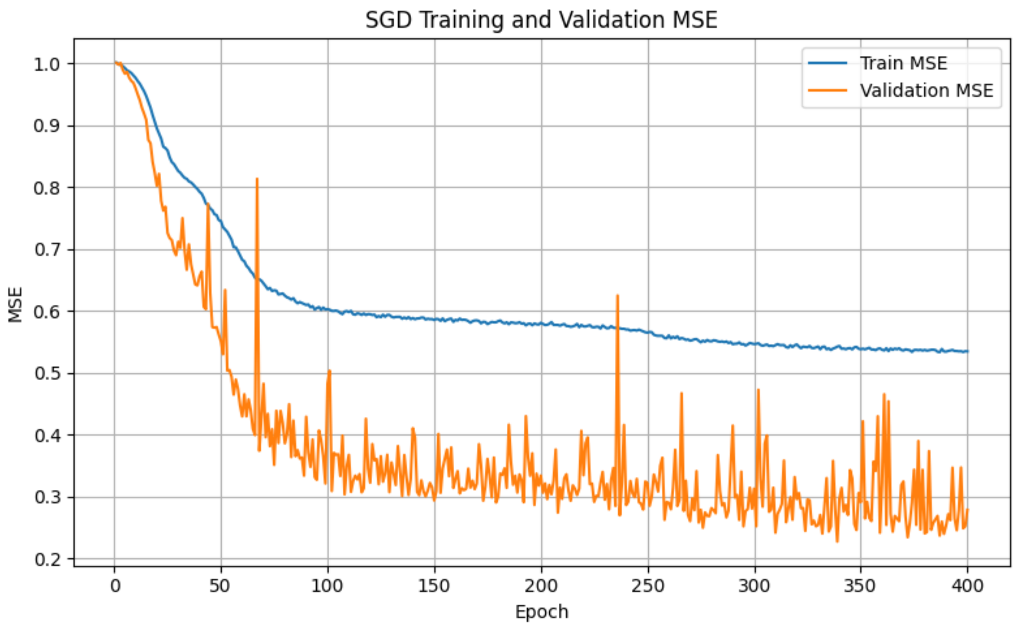

Figure 3: Training and validation MSE over epochs for the final SGD-trained FFN.

4.2 L-BFGS Optimiser

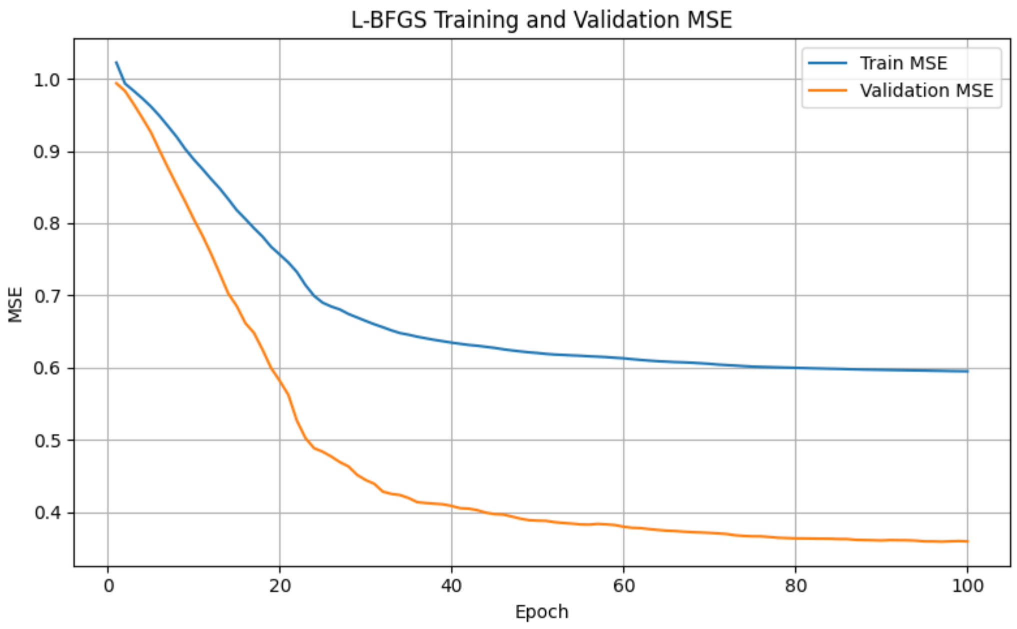

Figure 4: Training and validation MSE over epochs for the L-BFGS-trained FFN.

4.3 Hybrid GA + Periodic SGD

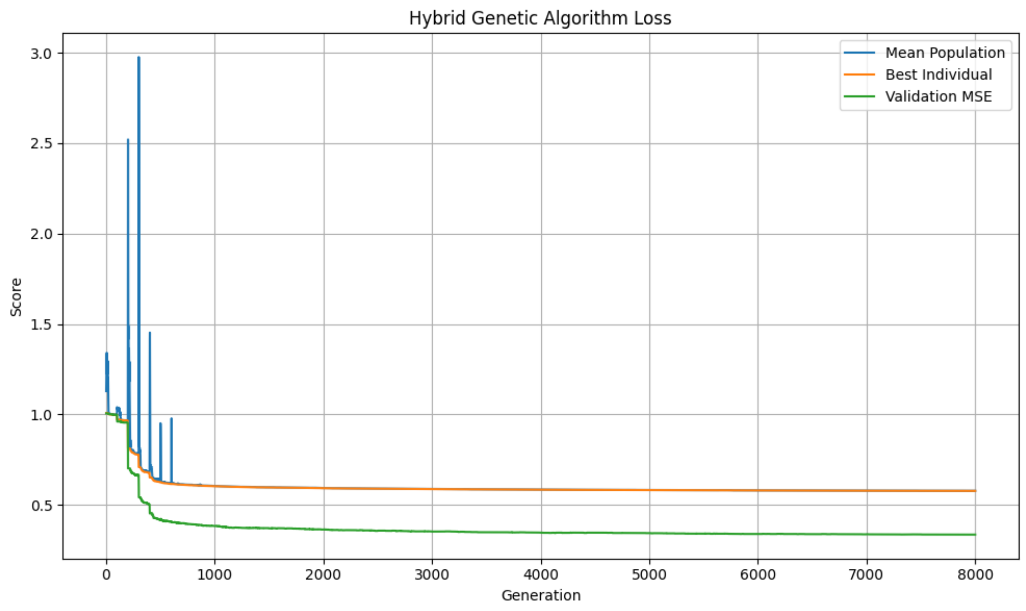

Figure 5: Training and validation MSE across generations for the hybrid GA with periodic SGD refinement.

4.4 Computational Times for Each Optimiser:

Optimiser Algorithm	Mean Time per Epoch (s)	Std Time per Epoch (s)
Hybrid GA	0.645318	13.4567
SGD	0.598595	0.116897
L-BFGS	0.440521	0.0518793

Table 1: Mean and standard deviation of time per epoch for each optimiser.

4.5 Test-set Scores for Each Optimiser:

Optimiser Algorithm	MSE (Scaled)	RMSE (Original)
Hybrid GA	0.3324	8.1872
SGD	0.5042	10.0830
L-BFGS	0.3549	8.4591

Table 2: Mean and standard deviation of time per epoch for each optimiser.

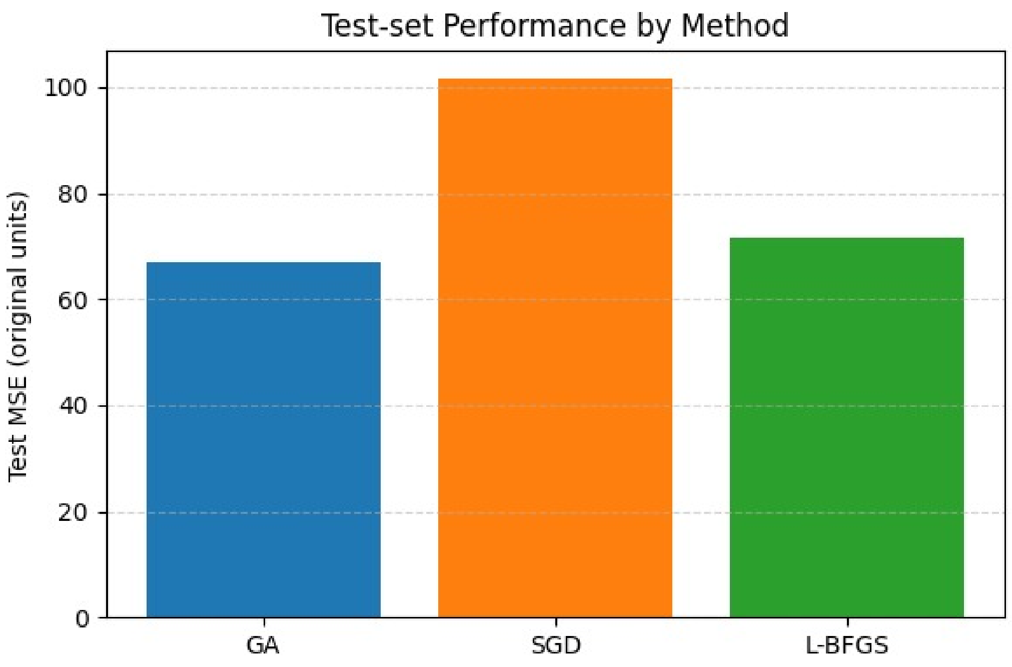

Figure 6: Final Test Set performance per Optimiser (MSE Original Units)

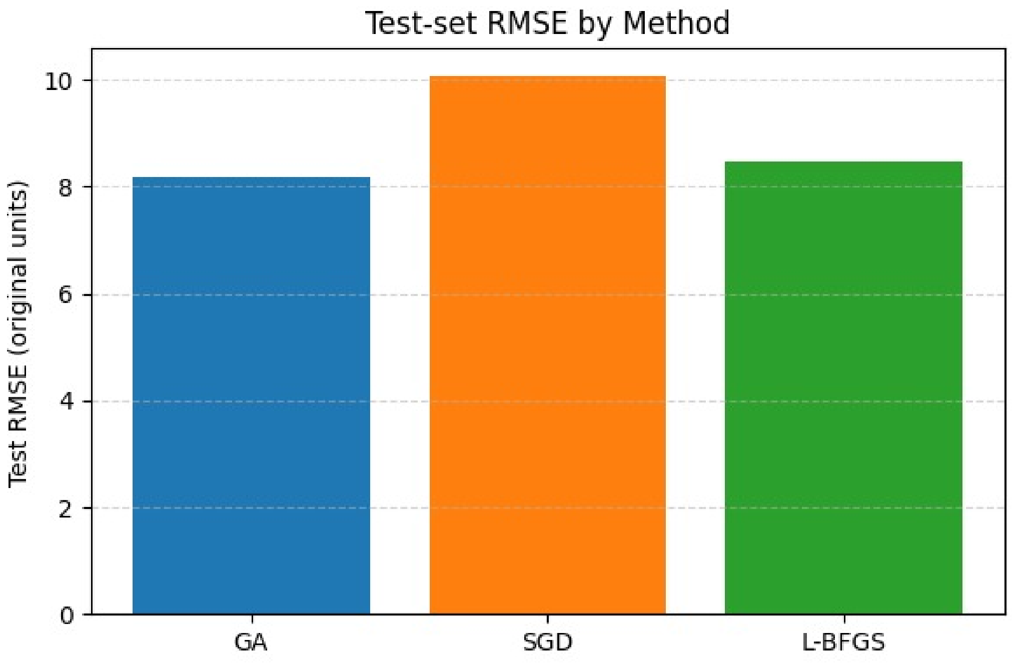

Figure 7: Final Test Set performance per Optimiser (RMSE Original Units)

5 Discussion

5.1 Short Comings, Failures and Strengths of Each Optimiser

The dataset presents a high-dimensional input-output mapping with a wide output range and strong nonlinearity. These characteristics suggest a complex, multimodal loss landscape with numerous local minima, saddle points, and flat or noisy regions. Such properties can significantly hinder the performance of traditional optimisation methods, particularly those relying solely on local gradient information.

L-BFGS, as a second-order quasi-Newton optimiser, performs well in smooth regions where the curvature is well-behaved. However, its deterministic nature and reliance on local curvature estimates make it vulnerable in multimodal landscapes. When the Hessian is ill-conditioned, L-BFGS may misrepresent the true curvature, leading to poor update directions and often divergence. Despite this, it often converges reliably to a nearby solution, though not necessarily the global optimum. Given a larger FFN, L-BFGS is expected to converge more consistently to a more optimal solution; but given the smaller nature of our FFN, the Hessian is (as stated previously) ill-conditioned, and the curvature information does not assist greatly.

SGD, implemented with a batch size of one to enforce full stochasticity, offers an advantage in such rugged landscapes. The inherent noise in its updates enables it to escape shallow minima and traverse plateaus more effectively. However, its performance is highly sensitive to hyperparameters like learning rate and is also quite sensitive to initialisation. It also tends to converge more slowly and inconsistently than more structured optimisers, often requiring careful tuning to avoid instability or premature convergence. The complicated nature of the landscape and gradient information evokes noisy and sometimes divergent updates, causing unpredictable and unstable performance.

The hybrid GA + SGD approach addresses the shortcomings of both previous methods by combining global search with local refinement. The genetic algorithm components—specifically BLX-α crossover and mutation—enable broad, global exploration of the parameter space, ensuring diversity and wide coverage. Periodic application of SGD every 100 generations allows the most promising solutions to be locally refined. The GA itself was also able to encourage convergence on good solutions in places (elite solutions in generations > 600) where SGD caused divergence. Thus our GA design in itself was competent in both global exploration and local exploitation. This combination yielded the most consistent and superior test MSE results across experiments, demonstrating strong capability in navigating complex, multimodal loss landscapes.

In summary, L-BFGS provides stable and predictable convergence but is limited by its reliance on local curvature and its sensitivity to ill-conditioning. SGD introduces valuable stochasticity, promoting exploration but requiring careful tuning and initialisation to be effective. The GA + SGD hybrid offers a robust solution, effectively balancing exploration and exploitation to outperform both standalone methods in this setting.

5.2 Trade-offs between Optimisers

The comparative performance of the three optimisation strategies reveals clear trade-offs in convergence speed, computational cost, and solution quality. L-BFGS and SGD, even after hyperparameter tuning, showed faster initial convergence but frequently diverged or stalled, requiring early stopping to prevent overfitting or instability. In contrast, the GA-based approaches, particularly the hybrid GA+SGD, demonstrated steady improvement over time and did not exhibit divergent behaviour, only consitent (though slow) improvement.

However, this robustness comes at a cost. The hybrid GA incurred significantly higher computational expense, running more than ten times slower per epoch compared to L-BFGS or SGD. Incorporating periodic SGD updates helped reduce stagnation and slightly improved convergence speed, but the method remained the most computationally intensive. Despite the increased cost, the hybrid GA consistently achieved the highest solution quality in terms of final test MSE, illustrating a clear trade-off: greater robustness and performance at the expense of time and efficiency.

5.3 Recommendations and Conclusions

This study demonstrates that optimiser performance is highly sensitive to the characteristics of the loss landscape. For problems involving multimodal, non-convex, and high-dimensional search spaces, traditional gradient-based methods like L-BFGS and SGD are often limited—either by premature convergence or sensitivity to hyperparameters.

In such settings, a hybrid approach that combines global search via Genetic Algorithms with local refinement via gradient descent provides a more reliable and effective optimisation strategy. The GA+SGD hybrid consistently achieved superior performance in test loss, albeit at higher computational cost.

Where computational resources permit, this trade-off is justifiable. However, in time-sensitive or resource-constrained applications, SGD remains a reasonable fallback, provided hyperparameter tuning is carefully performed. L-BFGS, while efficient and stable in smoother regions, is less suitable for rugged or ill-conditioned landscapes.

In conclusion, no single optimiser is universally best. The choice depends on the optimisation landscape and resource constraints. This work highlights the value of hybrid methods in achieving high-quality solutions where global exploration and local exploitation are both necessary.

References

[1] Lecture 1: A Gentle Introduction to Evolutionary Algorithms. Stellenbosch University, Chris Laubscher-Pretorius, 2025.

[2] `torch.optim.LBFGS` — PyTorch Documentation. `https://docs.pytorch.org/docs/stable/generated/torch.optim.LBFGS.html`

[3] `torch.optim.SGD` — PyTorch Documentation. `https://docs.pytorch.org/docs/stable/generated/torch.optim.SGD.html`

[4] *Limited-memory BFGS.* Wikipedia. `https://en.wikipedia.org/wiki/Limited-memory_BFGS`

[5] *Stochastic gradient descent.* Wikipedia. `https://en.wikipedia.org/wiki/Stochastic_gradient_descent`

[6] *What Are Genetic Algorithms?* Spiceworks AI Article. `https://www.spiceworks.com/tech/artificial-intelligence/articles/what-are-genetic-algorithms/`

[7] M. Tuba, M. Subotic, et al. *Genetic Algorithm – An Overview of Applications and Variants.* In: IntechOpen (2022). `https://www.intechopen.com/chapters/81745`

[8] ChatGPT was used to summarise insights, reword original content for academic writing, generate plotting and boilerplate code, produce docstrings, and assist in implementing GA and PyTorch-based architectures. The authors ensured all outputs were reviewed and adapted to maintain academic integrity.