Comparison of Feedforward Neural Network Training Algorithms

BS Steyn
(Department of Computer Science)
Stellenbosch University
Stellenbosch, South Africa
21740178@sun.ac.za

***Abstract*—This study compares stochastic gradient descent (SGD), scaled conjugate gradient (SCG), and LeapFrog(LFO) on single-hidden-layer multilayer perceptrons for standard classification tasks. Models use standardised features, hidden width selected by cross-validation, and matched full-batch training budgets. On the tabular datasets, both LFO and SGD reached near-perfect accuracy, while SCG consistently underperformed. Statistical tests across the three datasets did not indicate significant differences after adjustment LFO and SGD emerge as practical defaults, with LFO offering a small stability edge.**

I. Introduction

The effectiveness of training neural networks [8] is determined not only by the chosen architecture but also by the optimisation algorithm employed. First-order methods such as Stochastic Gradient Descent (SGD) [7] remain widely used due to their simplicity and scalability. In contrast, curvature-aware methods such as Scaled Conjugate Gradient (SCG) [5] aim to accelerate convergence by exploiting approximate second-order information. More recently, dynamic schemes such as the Leap-Frog Optimiser (LFO) [3] have been proposed, introducing structured updates inspired by symplectic integration with step-size control. The objective of this study is to compare SGD, SCG, and LFO when applied to single-hidden-layer multilayer perceptrons trained on standard classification datasets. The comparison focuses on stability, sample efficiency under a fixed iteration budget, and test performance. The scope of the evaluation is deliberately constrained to isolate the effect of the optimiser. All models employ a single hidden layer, features are standardised using training statistics, and hidden layer width is determined through cross-validation. Training uses matched full-batch updates with identical initialisations and iteration budgets across all optimisers. The empirical design includes three benchmark classification datasets with varying dimensionality and decision-boundary complexity. Primary metrics are classification accuracy, with macro-F1 and mean absolute error reported as supporting evidence. Statistical comparisons across datasets are conducted using paired non-parametric tests with multiplicity adjustment.

The remainder of this paper is separated into the following sections: Section II provides a background for this paper. Section III outlines the datasets used in this paper. Section IV provides an overview of the implementation. Section V provides an overview of the evaluation metrics used in this paper. Section VI provides the results of which are summarised and concluded in Section VII.

II. Background

This section provides a background on the individual training algorithms.

A. Neural Networks

Artificial neural networks [8] are computational models inspired by biological neurons, designed to approximate complex functions through interconnected processing units. A multilayer perceptron consists of an input layer, one or more hidden layers, and an output layer. Each neuron computes a weighted sum of its inputs, applies a non-linear activation function, and propagates the result forward. The training process involves adjusting the weights to minimise a chosen loss function, typically by exploiting gradient-based optimisation. Even a single-hidden-layer network is theoretically capable of approximating any continuous function, provided a sufficient number of hidden units are used. The training algorithm therefore, plays a critical role in ensuring stable convergence and generalisation.

B. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [7] is a first-order optimisation method widely applied in neural network training. The algorithm updates model parameters iteratively by moving them in the direction opposite to the gradient of the loss function. For a weight vector w, the update rule is given by

$$w_{t+1} = w_t - \eta \nabla L(w_t), \tag{1}$$

where η denotes the learning rate and $\nabla L(w_t)$ the gradient of the loss at time step t. In practice, momentum is often incorporated to accelerate convergence and reduce oscillations in directions of high curvature. Despite its simplicity, SGD remains a robust baseline due to its scalability and adaptability across diverse datasets and architectures. Its sensitivity to the learning rate, however, requires careful tuning to avoid divergence or stagnation.

C. Scaled Conjugate Gradient

Scaled Conjugate Gradient (SCG) [5] is a second-order optimisation method that approximates curvature information without explicitly computing the Hessian matrix. The algorithm combines features of conjugate gradient techniques with a scaling mechanism that stabilises the line search. Each update step is computed along conjugate directions, which ideally accelerates convergence compared to standard gradient descent. SCG introduces a damping parameter to regulate step sizes when curvature estimates are unreliable. Although SCG eliminates the need for costly line searches, it remains sensitive to the choice of initial parameters and to the curvature-probing step size. In practice, SCG can achieve fast convergence in well-conditioned problems, but its robustness across noisy or complex loss landscapes is limited.

D. Leap-Frog Optimiser

The Leap-Frog Optimiser (LFO) [3] is a dynamic integration-based approach inspired by symplectic methods in numerical analysis. It maintains both parameter values and auxiliary momentum-like terms, which are updated in alternating half-steps. This scheme allows the optimiser to simulate a discretised dynamical system with controlled step sizes. The leap-frog updates preserve stability under larger integration intervals compared to explicit gradient descent, while still avoiding the overhead of full curvature computations. The method can be expressed as

$$v_{t+\frac{1}{2}} = v_t - \frac{\Delta t}{2}\nabla L(w_t), \tag{2}$$

$$w_{t+1} = w_t + \Delta t\, v_{t+\frac{1}{2}}, \tag{3}$$

$$v_{t+1} = v_{t+\frac{1}{2}} - \frac{\Delta t}{2}\nabla L(w_{t+1}), \tag{4}$$

III. Datasets

This section provides an overview and justification for the chosen datasets used in this paper.

A. General Justification

The chosen benchmark classification datasets are **Iris**, **Wine** and **Digits**. These sets cover low, medium and high-dimensional feature spaces with increasing decision-boundary complexity. Individual justifications are provided in the relevant subsections.

B. Iris

The **Iris** dataset contains 150 observations of iris flowers, equally distributed across three species: *setosa*, *versicolor* and *virginica*. Each entry contains four continuous measurements. These measurements are sepal length, sepal width, petal length, and petal width. All are recorded in centimetres. The classes are balanced as shown in figure 1:

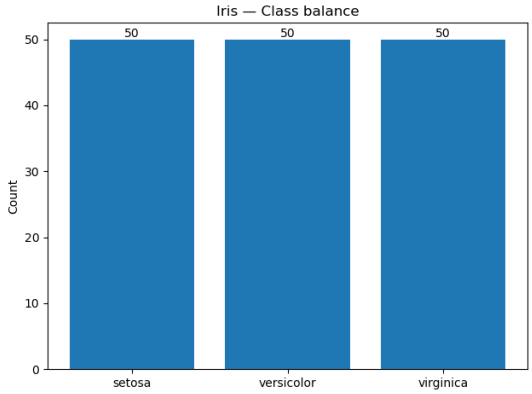

Fig. 1: Graph showing the distribution of the Iris dataset

Features are standardised to have a mean of zero and a variance of one to neutralise scale effects. The **Iris** dataset provides a compact, noise-tolerant benchmark that is sensitive to both under- and over-parameterisation in a single-hidden-layer network, rendering it appropriate for stress-testing convergence behaviour and regularisation choices at a small scale.

C. Wine

The **Wine** dataset contains 178 observations from a chemical analysis of wines originating from three cultivars. Each entry contains 13 continuous measurements. The class balance is shown in Figure 2:

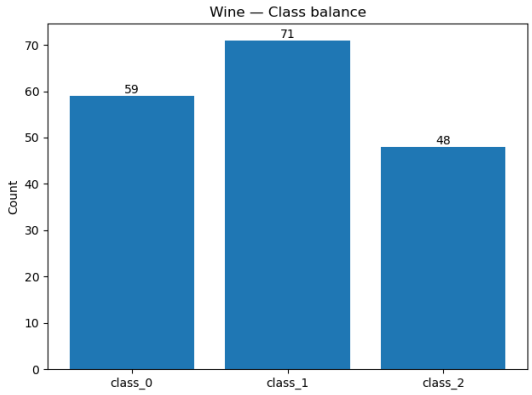

Fig. 2: Graph showing the distribution of the Wine dataset

Features are standardised with no imputation required. The **Wine** dataset introduces a higher-dimensional continuous feature space with real-world correlations and varying predictor scales. The dataset is appropriate for evaluating sensitivity to feature scaling, weight decay, and line-search or step-size policies.

D. Digits

The **Digits** dataset contains 1797 handwritten digits 0-9. Each is stored in an *8x8* greyscale bitmap flattened to *64* features. The classes are balanced as shown in figure 3:

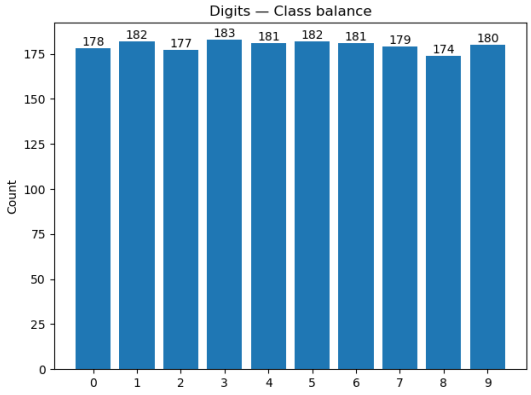

Fig. 3: Graph showing the distribution of the Digits dataset

Pixel intensities are scaled to the unit interval and then standardised per feature. Stratified splits are used for evaluation. The **Digits** dataset exercises the capacity of a single-hidden-layer network to model non-linear image patterns without a convolutional structure, providing a stringent test of training dynamics and optimiser robustness.

IV. Implementation

This paper implements the discussed neural-network models in PyTorch [6]; SGD uses the built-in optimiser, Scaled Conjugate Gradient (SCG) and LeapFrog P1(b) (LFO) are implemented as custom PyTorch optimisers with closure-based gradient evaluation. Data handling uses NumPy [1] and pandas [4]; visualisation uses Matplotlib [2].

V. Evaluation Metric

This section outlines the evaluation procedure, including optimiser tuning, training protocol, and statistical testing

A. Optimiser tuning

Each optimiser is tuned on the training data via a compact grid:

- **SGD**: For SGD the compact grid is constructed using $\{0.10, 0.05, 0.01\}$ x momentum $\{0, 0.9\}$.
- **SCG**: For SCG the compact gird is constructed using curvature-probe step $\theta_0 \in \{1x10^{-4}, 5x10^{-5}, 2x10^{-4}\}$ x the dampening parameter. The dampening parameter adapts during optimisation.
- **LFO**: For LFO, the compact grid is constructed using $\Delta \in \{10^{-3}, 5x10^{-3}, 10^{-2}\}$ x maximum step $\delta \in \{0.05,0.10,0.20\}$.

For each grid point, only the configuration with the best validation score is kept. The same selected configuration is then evaluated once on the test split.

B. Training

Training employs full-batch updates to equalise step accounting across algorithms. A fixed iteration budget is used per run. Network weights are initialised with a fixed random seed per dataset; the identical initialisation is reused for each optimiser to control for start-point variance. No early stopping is applied during selection or testing.

C. Performance Metrics

Primary metrics are accuracy for classification. Macro-F1 (classification) is reported as secondary evidence. For each dataset–algorithm pair, the final test score is reported together with the selected hyperparameters.

D. Statistical Evaluation

Cross-dataset comparisons use paired scores aggregated per dataset. Pairwise differences between algorithms are assessed with the Wilcoxon signed-rank test with an α = 0.05.

VI. Results

The results are presented in the following three subsections. Subsection A presents the performance of the models. Subsection B presents the statistical significance of the performance. Subsection C presents the best found configuration for each model.

A. Classification performance

Tables I and II report test accuracy and macro-F1 for the three optimisers on Iris, Wine, and Digits. Mean scores (averaged over the three datasets) appear in the final columns. On the Iris and Wine datasets, both LFO and SGD achieved near-perfect performance, while SCG failed to produce a competitive classifier. On Digits, LFO and SGD maintained strong performance, highlighting their robustness compared to SCG.

TABLE I: Test performance on classification datasets for accuracy. Values rounded to three decimals.

Optimiser	Iris Acc	Wine Acc	Digits Acc	Mean Acc
SGD	1.000	1.000	0.973	0.991
SCG	0.000	0.311	0.129	0.147
LFO	1.000	1.000	0.976	0.992

TABLE II: Test performance on classification datasets for marco-F1 score. Values rounded to three decimals.

Optimiser	Iris F1	Wine F1	Digits F1	Mean F1
SGD	1.000	1.000	0.973	0.991
SCG	0.000	0.251	0.106	0.119
LFO	1.000	1.000	0.976	0.992

On Iris and Wine, both LFO and SGD reach perfect accuracy and macro-F1, while SCG fails to learn a useful classifier. On Digits, LFO (0.976 / 0.976) and SGD (0.973 / 0.973) remain strong, with SCG markedly weaker (0.129 / 0.106).

B. Statistical comparison

Pairwise comparisons across the three datasets use the Wilcoxon signed-rank test with Holm adjustment at $\alpha = 0.05$. With only three datasets, none of the pairwise differences reach significance after correction (Table III).

TABLE III: Wilcoxon signed-rank tests across datasets (higher is better). Holm-adjusted decision thresholds shown for reference.

Pair	p-value	Holm threshold
SGD vs SCG	0.25	0.01667
SCG vs LFO	0.25	0.02500
SGD vs LFO	1.00	0.05000

C. Best configurations

The best configuration found for each training algorithm for each dataset is shown below:

- **Iris:** SGD $\{$`lr = 0.10, momentum = 0.9`$\}$; SCG $\{\sigma_0$ `= 1e-4`$\}$; LFO $\{\Delta t$ `= 0.001,` δ `= 0.10,` M `= 3,` δ_i `= 1e-3`$\}$.
- **Wine:** SGD $\{$`lr = 0.10, momentum = 0.0`$\}$; SCG $\{\sigma_0$ `= 1e-4`$\}$; LFO $\{\Delta t$ `= 0.001,` δ `= 0.05,` M `= 3,` δ_i `= 1e-3`$\}$.
- **Digits:** SGD $\{$`lr = 0.05, momentum = 0.9`$\}$; SCG $\{\sigma_0$ `= 1e-4`$\}$; LFO $\{\Delta t$ `= 0.005,` δ `= 0.10,` M `= 3,` δ_i `= 1e-3`$\}$.

VII. Conclusion

This study evaluates three training algorithms—stochastic gradient descent (SGD), scaled conjugate gradient (SCG), and LeapFrog P1(b) (LFO)—on single-hidden-layer multilayer perceptrons across standard classification benchmarks. Under a matched training budget and identical initialisations, LFO and SGD achieve consistently high performance, reaching perfect accuracy on the smaller tabular tasks and strong results on the image task. Under the tested settings, SCG demonstrated weaker performance across all three datasets, reflecting its sensitivity to curvature estimation.

The empirical comparison indicates that simple, well-tuned first-order updates and the leapfrog dynamic with adaptive time stepping provide reliable optimisation behaviour for shallow networks. LFO offers a small average edge over SGD while maintaining stable convergence, at the cost of closure-based gradient evaluations. SCG's sensitivity to curvature probing and damping choices manifests as poor robustness in this setting.

Statistical testing across the available datasets does not yield significance after multiple-comparison adjustment, which reflects the limited sample of tasks rather than an absence of effect. The direction and magnitude of the observed differences, together with the per-dataset results, nevertheless support the practical preference for LFO or SGD for comparable architectures and data scales.

Future work should broaden the evidence base. Recommended extensions include larger and more diverse datasets, deeper architectures to probe optimiser–capacity interactions, and ablations on LFO's control parameters (time-step limits, restart criteria, and noise injection for minibatch training). Incorporating stronger baselines (e.g., momentum variants and adaptive methods), reporting compute-adjusted efficiency metrics, and analysing convergence traces will further clarify trade-offs. Overall, the findings endorse LFO and SGD as dependable choices for single-hidden-layer models, with LFO offering a principled alternative when stability and exploratory dynamics are desirable. https://github.com/Bernhardt-Steyn/21740178ML03

References

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. "Array Programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[2] John D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511614118.

[4] Wes McKinney. "Data Structures for Statistical Computing in Python". In: (2010). Ed. by Stéfan van der Walt and Jarrod Millman, pp. 51–56.

[5] Martin Fodslette Møller. "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning". In: *Neural Networks* 6.4 (1993), pp. 525–533. DOI: 10.1016/S0893-6080(05)80056-5.

[6] Adam Paszke, Sam Gross, Francisco Massa, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019, pp. 8024–8035.

[7] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586.

[8] Frank Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[9] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. Ed. by David E. Rumelhart and James L. McClelland. MIT Press, 1986, pp. 318–362.

[10] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83. DOI: 10.2307/3001968.