# Comparison of Feedforward Neural Network Training Algorithms

Louis Anthony Wilkinson 25948873
Machine Learning 441
*Stellenbosch University*
Stellenbosch, South Africa
25948873@sun.ac.za

***Abstract*—This study compares the performance of three feedforward neural network training algorithms: Stochastic Gradient Descent (SGD), Scaled Conjugate Gradient (SCG), and LeapFrog optimisation. The algorithms are evaluated on six benchmark datasets comprising three classification and three function approximation problems of varying complexity. Single hidden layer architectures are employed across all experiments. The empirical process involves systematic determination of optimal hidden layer sizes and algorithm-specific control parameters through two-phase joint optimisation with asymmetric grid search: 16 parameter combinations for SGD and LeapFrog, 4 combinations for SCG to test theoretical parameter-free advantage. Performance evaluation utilises 10-fold cross-validation with parameter sensitivity analysis via coefficient of variation. Results reveal distinct algorithmic characteristics. SGD achieves superior performance on four of six tasks through task-dependent momentum tuning ranging from 0.0 to 0.99, winning all classification tasks and two regression problems. LeapFrog demonstrates advantage on periodic regression and rapid convergence on complex multi-class classification. SCG validates theoretical parameter-free advantage with coefficient of variation below 2.20% on four datasets, achieving competitive performance with minimal tuning but encountering severe convergence failures on three tasks including numerical instability on 17-class Beer classification. Statistical significance testing confirms algorithmic performance differences on five of six datasets at $p \leq 0.05$, with all three regression tasks achieving $p \leq 0.02$. The findings demonstrate that proper momentum tuning based on problem complexity enables SGD superiority on most tasks, while physics-inspired LeapFrog dynamics excel on periodic functions and SCG provides robust performance on simple problems despite convergence limitations on complex tasks.**

***Index Terms*—neural networks, training algorithms, stochastic gradient descent, scaled conjugate gradient, LeapFrog optimisation, feedforward networks**

## I. Introduction

Neural network training algorithms represent a fundamental component in machine learning applications. The effectiveness of neural network training algorithms directly impacts network convergence speed, solution quality, and generalisation performance. This investigation compares three distinct training approaches: Stochastic Gradient Descent (SGD), Scaled Conjugate Gradient (SCG), and the LeapFrog algorithm.

This investigation addresses the comparative performance of the three training algorithms across diverse problem domains. Classification and function approximation tasks enable thorough evaluation. Each algorithm has theoretical advantages under specific conditions, requiring empirical testing on problems of different complexity.

The research methodology employs single hidden layer feedforward networks to maintain experimental consistency. Systematic hyperparameter optimisation ensures fair algorithm comparisons. Statistical significance testing validates performance differences between training approaches.

Section II reviews the theoretical foundations of each training algorithm. Section III details the implementation specifics and experimental setup. Section IV describes the empirical process for hyperparameter optimisation. Section V presents experimental results and statistical analysis. Section VI summarises findings and implications.

Code available at: https://github.com/WilkieZA/CS441Assignment3

## II. Background

Error-based learning refers to supervised training methods where the parameters of a model are adjusted iteratively to minimise a defined error (loss) function on a training dataset. Typically, the learning objective is to find parameters $w$ that minimise the empirical error

$$E(w) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(x_i; w), y_i\big), \tag{1}$$

where $N$ is the number of training examples, $f(x_i; w)$ is the prediction of the model for input $x_i$, $y_i$ is the true target, and $\ell(\cdot)$ is a chosen loss function. A fundamental approach to minimising $E(w)$ is gradient descent. In gradient-based learning, the parameter vector is updated in the direction of the negative gradient of the error:

$$w_{t+1} = w_t - \eta \nabla_w E(w_t), \tag{2}$$

where $\eta$ is a learning rate. Gradient-based learning underlies the standard backpropagation algorithm for neural networks. Gradient descent typically converges to local minimisers rather than saddles on smooth non-convex objectives [4]. In practice, performance is also sensitive to hyperparameters such as learning rate and momentum. These hyperparameters often require empirical tuning [1].

### A. Stochastic Gradient Descent

A simple enhancement to basic gradient descent is the use of a momentum term. Momentum adds a fraction of the previous weight update to the current update. The momentum term helps smooth out oscillations and accelerates convergence by dampening the zig-zagging trajectory on the error surface. However, momentum alone does not fully resolve the slow convergence or local optima problems. The SGD implementation in this study employs classical momentum-based updates with optional Nesterov acceleration, implemented as a custom PyTorch optimiser following the standard torch.optim interface.

### B. LeapFrog Algorithm

One notable approach is the dynamic "leap-frog" method introduced by Snyman [5]. The LeapFrog algorithm interprets the error function as a potential energy surface. The training process is modeled as the motion of a particle in a conservative force field defined by the potential energy surface. The position of the particle represents the current weight vector $w$. The force acting on the particle is the negative gradient $-\nabla_w E$. The LeapFrog algorithm of Snyman simulates the trajectory of the particle and employs an interference strategy. The strategy monitors the kinetic energy of the particle and dissipates the kinetic energy when necessary. The dissipation ensures that the potential energy is systematically reduced [5].

The leap-frog procedure effectively incorporates a form of momentum while guaranteeing eventual convergence to a local minimum. Empirical tests indicate that the LeapFrog method is competitive with classical algorithms [5]. The computational cost of the dynamic approach scales approximately linearly with the problem dimension, whereas classical quasi-Newton methods exhibit quadratic scaling [5]. Snyman later introduced an automatic step-size adjustment mechanism to further improve the efficiency and stability of the algorithm [6]. The implementation uses velocity-Verlet integration with adaptive time stepping. The implementation in this study incorporates the angle-based acceptance criterion and restart mechanism from the LFOP1(b) variant as a custom PyTorch optimiser.

### C. Scaled Conjugate Gradient

Well-established numerical optimisation methods have been adapted to neural network training. Conjugate gradient (CG) methods accelerate convergence beyond plain gradient descent. CG chooses search directions that are mutually conjugate with respect to the Hessian matrix, ensuring that progress along one direction does not interfere with progress along previous directions. The conjugate directions avoid the frequent oscillations of basic gradient descent. Empirical studies have shown that a standard CG algorithm with line search converges roughly an order of magnitude faster than backpropagation [3]. Both CG and quasi-Newton techniques typically rely on a line search at each iteration to determine an appropriate step size. Line searches require multiple evaluations of the error function per iteration [3]. The extra computational cost diminishes the practical efficiency gains of CG, especially for large networks where each function and gradient evaluation is expensive.

A significant advancement to address the computational overhead of line searches was the SCG algorithm developed by Møller [3]. SCG is a variant of conjugate gradient that eliminates the need for explicit per-iteration line searches. SCG dynamically estimates an optimal step length using second-order curvature information. The step length adaptation reduces the need for manual tuning compared to standard CG methods. The experiments of Møller showed that SCG substantially outperforms standard backpropagation and conventional CG methods in training speed [3]. SCG effectively combines the efficiency of advanced optimisation techniques with the automation and simplicity of first-order gradient descent. The implementation in this study follows the algorithm of Møller directly as a custom PyTorch optimiser.

This section established the theoretical foundations of the three training algorithms under investigation. SGD provides a momentum-enhanced baseline with established convergence properties. LeapFrog introduces physics-inspired dynamics with adaptive time stepping for improved exploration of complex error surfaces. SCG eliminates line search overhead through second-order curvature estimation while maintaining computational efficiency.

## III. Implementation

The implementation addresses the practical realisation of the three training algorithms across diverse problem domains. The section describes the neural network architecture employed in all experiments. Dataset selection and preprocessing procedures ensure fair algorithm comparison. The remainder of the section is organised as follows: network architecture design, dataset characteristics, and data preprocessing methodology.

### A. Network Architecture

All experiments employ single hidden layer feedforward networks implemented using PyTorch. The architecture consists of input nodes corresponding to problem dimensionality. Hidden nodes use hyperbolic tangent activation. Output nodes use sigmoid activation for binary classification, softmax activation for multi-class classification, and linear activation for function approximation. The PyTorch framework provides automatic differentiation for gradient computation.

**Computational Environment:** All experiments were conducted on an Apple M4 processor running macOS 15.6.1. The implementation uses Python 3.13.2 with PyTorch 2.8.0 and NumPy 2.1.3. All computations employ 32-bit floating-point precision. Random seeds are fixed at 42 for reproducibility across cross-validation folds and weight initialisation.

### B. Datasets

The evaluation utilises six benchmark datasets providing varying complexity levels across classification and function approximation tasks:

**Classification Problems:**

- **Beer Style Classification**: A multiclass classification dataset with 518 samples, 20 features, and 17 beer style classes. The dataset contains quantitative measures including alcohol content measured as 'ABV' and bitterness values measured as 'IBU'. Taste descriptors comprise 11 features: 'Astringency', 'Body', 'Alcohol', 'Bitter', 'Sweet', 'Sour', 'Salty', 'Fruits', 'Hoppy', 'Spices', and 'Malty'. Review ratings include five features: 'review_aroma', 'review_appearance', 'review_palate', 'review_taste', and 'review_overall'. This represents a high-complexity classification problem with moderate class imbalance.
- **Titanic Survival Prediction**: A binary classification dataset with 891 samples and 11 features. Features include passenger class, sex, age, siblings/spouses aboard, parents/children aboard, fare, cabin, and embarkation port. The target variable indicates passenger survival where zero represents 'No' and one represents 'Yes'. This represents a medium-complexity classification problem with inherent feature interactions.
- **Iris Species Classification**: A classic three-class classification dataset with 150 samples and four features. The target variable identifies iris species. The dataset represents a low-complexity classification problem with well-separated classes.

**Function Approximation Problems:**

- **Gaussian Mixture Surface**: Three-input function combining multiple Gaussian components $f(x, y, z) = \sum_{i=1}^{3} a_i \exp(-\|\mathbf{x}-\boldsymbol{\mu}_i\|^2/\sigma_i^2)+\epsilon$. The parameters include amplitude coefficients $a_i$, center locations $\boldsymbol{\mu}_i$, spread parameters $\sigma_i$, and Gaussian noise $\epsilon$. With 1 000 observations sampled from this function, it represents high-complexity regression with multiple local optima.
- **Multi-dimensional Sinusoidal**: Two-input function $f(x, y) = \sin(2\pi x)\cos(2\pi y) + \epsilon$ where $\epsilon$ represents Gaussian noise. With 800 observations sampled from this function, it represents medium-complexity regression.
- **Synthetic Polynomial**: Single-input polynomial function $f(x) = x^3 - 2x^2 + x + \epsilon$ where $\epsilon$ represents Gaussian noise. With 500 observations sampled from this function, it represents a low-complexity regression task with nonlinear characteristics.

Figure 1 visualises the polynomial and sinusoidal functions. The polynomial exhibits nonlinear characteristics with a cubic relationship, while the sinusoidal function demonstrates periodic patterns with interaction terms between the two input dimensions. Red scatter points indicate noisy training samples drawn from each function.

### C. Data Preprocessing

Data preprocessing follows dataset-specific strategies designed to address unique characteristics and challenges identified through exploratory data analysis.

*1) Beer Style Classification:* The beer dataset presented several preprocessing challenges requiring systematic handling:

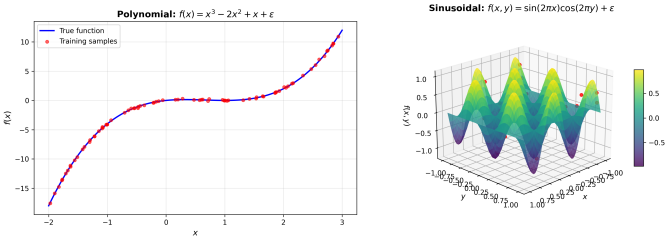


Fig. 1. Polynomial function (left) and sinusoidal function (right)

**Missing Value Treatment**: The dataset contained both explicit missing values coded as 'NaN' and placeholder values coded as 9876 representing approximately 5% missing data per feature. Placeholder values were converted to 'NaN' and imputed using median values for numerical features, preserving the distribution while maintaining dataset integrity.

**Feature Engineering**: Non-predictive columns including 'ID', 'Name', 'Brewery', and 'Description' were removed to focus on discriminative features. The final feature set includes alcohol content measured as 'ABV', bitterness measures measured as 'IBU', taste descriptors comprising 11 features, and review ratings comprising five features, totaling 20 numerical features.

**Target Encoding**: The 17 beer styles were label-encoded to integer values ranging from 0 to 16 for neural network compatibility. Class imbalance ranges from 14 to 40 samples per class, representing a moderate imbalance typical of multi-class problems.

*2) Titanic Survival Prediction:* The Titanic dataset required extensive feature engineering to capture survival patterns:

**Feature Engineering**: New features were derived from existing data. 'FamilySize' was calculated as the sum of siblings, spouses, parents, and children on board, plus one for the passenger. 'IsAlone' was created as a binary indicator. 'Title' was extracted from passenger names and grouped into five categories: Mr, Miss, Mrs, Master, and Rare. 'Deck' was extracted from cabin information with 'Unknown' for missing values.

**Missing Value Treatment**: 'Age' missing values representing 19.9% of data were imputed using median values. The two missing 'Embarked' values were imputed using mode. 'Cabin' information, missing for 77.1% of passengers, was transformed into deck information with an 'Unknown' category.

**Categorical Encoding**: All categorical variables including 'Sex', 'Embarked', 'Title', and 'Deck' were label-encoded to numerical values. The preprocessing pipeline produced 11 features capturing passenger demographics, family structure, social status, and travel details.

**Final Feature Set**: The 11 features fed to the neural network comprise: 'Pclass' (passenger class), 'Age' (median imputed), 'SibSp' (siblings/spouses aboard), 'Parch' (parents/children aboard), 'Fare' (ticket price), 'FamilySize' (engineered feature), 'IsAlone' (binary indicator), 'Sex_encoded' (label encoded), 'Embarked_encoded' (label encoded with three port

values), 'Title_encoded' (five categories including Mr, Miss, Mrs, Master, and Rare), and 'Deck_encoded' (cabin deck letter with 'Unknown' category for missing 'Cabin' values representing 77.1% of passengers).

*3) Iris Species Classification:* The Iris dataset required minimal preprocessing due to the high data quality of Iris:

**Data Quality**: No missing values were detected across all 150 samples and four features. The Iris dataset exhibits perfect class balance with 50 samples per species.

**Feature Selection**: Only the 'ID' column was removed, retaining all four morphological measurements including sepal length, sepal width, petal length, and petal width as these measurements provide strong discriminative power for species classification.

**Target Encoding**: Species names were label-encoded to integers where zero represents Iris-setosa, one represents Iris-versicolor, and two represents Iris-virginica, maintaining alphabetical ordering.

*4) Standardisation:* All datasets underwent feature standardisation using z-score normalisation with mean $\mu = 0$ and standard deviation $\sigma = 1$ to ensure optimal neural network training. Standardisation prevents features with larger scales from dominating the learning process and promotes stable gradient-based optimisation. The transformation applies independently to each feature according to

$$x_{\texttt{std}} = \frac{x - \mu}{\sigma} \tag{3}$$

where $\mu$ and $\sigma$ represent the training set mean and standard deviation respectively.

Cross-entropy loss functions are employed for all classification tasks due to superior gradient properties compared to mean squared error (MSE). When combined with sigmoid or softmax activation functions, cross-entropy produces consistent gradient magnitudes throughout training, enabling more reliable convergence in classification problems [2].

*5) Synthetic Function Approximation Data:* The three function approximation datasets were generated using deterministic mathematical functions with added Gaussian noise. Sample sizes scale with complexity: 500 samples for polynomial, 800 samples for sinusoidal, and 1000 samples for Gaussian mixture. Gaussian noise with standard deviations of 0.1-0.2 provides realistic regression challenges while maintaining clear underlying signal structure. Input features undergo standardisation following the same protocol as classification datasets.

This section detailed the implementation framework supporting fair algorithm comparison. Single hidden layer architectures provide controlled experimental conditions across six diverse datasets spanning classification and function approximation tasks. Comprehensive preprocessing ensures data quality while preserving discriminative characteristics essential for meaningful performance evaluation.

## IV. Empirical Process

The empirical process provides a systematic methodology for fair algorithm comparison. The experimental design uses rigorous hyperparameter optimisation and thorough performance evaluation. The section describes the two-phase tuning methodology, algorithm-specific parameter ranges, and performance evaluation metrics.

### A. Hyperparameter Tuning Methodology

A two-phase grid search over both algorithm parameters and network architecture ensures thorough evaluation and fair comparison:

**Phase 1: Coarse Grid Search** All algorithm-specific parameter combinations are evaluated across multiple architecture sizes. For classification datasets, five architectures are tested: 10, 15, 20, 25, and 30 hidden units for Iris and Titanic, and 20, 25, 30, 35, and 40 hidden units for Beer. Function approximation datasets use 10, 15, 20, 25, and 30 hidden units. Each algorithm's parameter grid (detailed below) is combined with these architectures, yielding 180 total configurations per dataset across the three algorithms.

**Phase 2: Fine Architecture Search** The best configuration for each algorithm from Phase 1 undergoes architecture refinement. Five additional architectures are tested around the optimal hidden layer size from Phase 1: the best size minus four units, minus two units, the best size, plus two units, and plus four units. This per-algorithm refinement identifies the optimal architecture for each training method independently.

When multiple configurations achieve identical performance, ties are broken by preferring lower standard deviation, then smaller architectures.

The two-phase methodology allows each training algorithm to achieve optimal performance under the preferred configuration of each algorithm, providing statistically robust comparisons while revealing algorithm-specific architectural preferences.

### B. Parameter Tuning

Algorithm-specific parameters are optimised using grid search following the original paper specifications:

**SGD Parameters:**

- Learning rate: $\eta \in \{0.001, 0.01, 0.05, 0.1\}$
- Momentum coefficient: $\mu \in \{0.0, 0.5, 0.9, 0.99\}$

**SCG Parameters:**

- Hessian approximation parameter: $\sigma \in \{1 \times 10^{-6}, 1 \times 10^{-5}\}$
- Levenberg-Marquardt parameter: $\lambda \in \{1 \times 10^{-3}, 1 \times 10^{-2}\}$

**LeapFrog Parameters:**

- Initial time step: $\Delta t \in \{1 \times 10^{-4}, 1 \times 10^{-3}\}$
- Maximum step constraint: $\delta \in \{0.1, 1.0\}$
- Growth factor: $\xi \in \{0.1, 0.2\}$
- Restart threshold: $m \in \{3, 5\}$

The parameter grid sizes differ across algorithms: SGD explores 16 combinations, SCG explores 4 combinations, and LeapFrog explores 16 combinations. This asymmetry reflects algorithmic characteristics rather than experimental bias. SCG's theoretical advantage lies in parameter robustness

through automatic step size adaptation via second-order curvature estimation. The algorithm requires minimal manual tuning compared to first-order methods. The smaller grid for SCG tests whether this parameter-free advantage holds empirically. Equal search budgets for SGD and LeapFrog enable fair comparison of first-order momentum methods against physics-inspired dynamics. Parameter sensitivity analysis quantifies robustness differences through coefficient of variation across tested configurations.

### C. Performance Evaluation

Classification performance is measured using classification accuracy, which provides straightforward comparison across algorithms and remains appropriate given the absence of severe class imbalance in the selected datasets. Function approximation performance is measured using root mean squared error (RMSE), the standard metric for regression tasks.

The empirical methodology establishes rigorous experimental protocols for algorithm comparison. Two-phase hyperparameter optimisation ensures each algorithm achieves optimal performance under preferred configurations. Statistical evaluation through 10-fold stratified cross-validation provides robust performance estimates while convergence analysis quantifies computational efficiency beyond final accuracy metrics.

Cross-validation employs stratified folds for classification tasks to preserve class distributions across training and validation splits. All algorithms are evaluated on identical fold partitions using a fixed random seed of 42, ensuring fair comparison without fold-selection bias. Data leakage is prevented through proper standardisation protocol: feature statistics including mean and standard deviation are computed exclusively on training fold data, then applied to validation fold data. This ensures validation performance accurately reflects generalisation capability on unseen data.

## V. Results and Discussion

The experimental results demonstrate distinct performance characteristics across the three training algorithms. Classification and function approximation tasks reveal algorithm-specific strengths and limitations. Convergence analysis quantifies computational efficiency beyond final accuracy metrics. Statistical significance testing validates meaningful performance differences. The section is organised as follows: classification results, function approximation results, computational efficiency analysis, parameter sensitivity analysis, statistical significance testing, and comparative algorithm analysis.

### A. Classification Results

The hyperparameter tuning process revealed significant differences in algorithm performance across the three classification datasets. Table I shows the optimal configurations and performance metrics achieved by each training algorithm. Complete hyperparameter configurations for all algorithms and datasets are provided in the Appendix.

TABLE I
HYPERPARAMETER TUNING RESULTS FOR CLASSIFICATION DATASETS WITH OPTIMAL CONFIGURATIONS

| Dataset | SGD | | SCG | | LeapFrog | |
|---|---|---|---|---|---|---|
| | Acc. | $h$ | Acc. | $h$ | Acc. | $h$ |
| Iris | **0.987** | **19** | 0.967 | 6 | 0.980 | 15 |
| Titanic | **0.837** | **22** | 0.831 | 16 | 0.797 | 27 |
| Beer | **0.902** | **18** | 0.697 | 10 | 0.884 | 34 |

*1) Dataset-Specific Performance:* **Iris Dataset Performance:** SGD achieved superior performance with 98.67% ± 2.67% using 19 hidden units with learning rate $\eta = 0.05$ and aggressive momentum $\mu = 0.99$. LeapFrog reached 98.00% ± 4.27% with 15 hidden units using parameters $\Delta t = 10^{-4}$, $\delta = 1.0$, $\xi = 0.1$, and $m = 5$. SCG achieved 96.67% ± 5.37% with minimal six hidden units using $\sigma = 10^{-6}$ and $\lambda = 10^{-3}$. All three algorithms perform well on simple three-class Iris classification, with SGD achieving marginal advantage through aggressive momentum.

**Titanic Dataset Performance:** On the medium-complexity Titanic survival prediction task, SGD achieved superior performance at 83.72% ± 3.08% with 22 hidden units and zero momentum $\mu = 0.0$. SCG achieved competitive performance at 83.05% ± 2.74% with 16 hidden units. LeapFrog reached 79.68% ± 2.26% with 27 hidden units. The close performance between SGD and SCG indicates that binary classification represents a well-conditioned optimisation problem.

**Beer Dataset Performance:** The high-complexity Beer style classification task revealed significant algorithmic differences. SGD achieved the highest performance at 90.15% ± 3.04% with 18 hidden units using learning rate $\eta = 0.05$ and moderate momentum $\mu = 0.5$. LeapFrog reached competitive performance at 88.41% ± 3.86% with 34 hidden units. However, SCG struggled on the Beer dataset, achieving only 69.67% ± 21.58% with 10 hidden units and exhibiting high variance across cross-validation folds. The large performance gap and variance suggest that SCG encounters numerical instability on high-dimensional 17-class problems, while SGD and LeapFrog maintain stable optimisation.

### B. Function Approximation Results

The hyperparameter tuning process revealed distinct algorithmic characteristics across the three regression datasets. Table II presents the optimal configurations and performance metrics achieved by each training algorithm.

TABLE II
FUNCTION APPROXIMATION RESULTS FROM HYPERPARAMETER TUNING WITH OPTIMAL CONFIGURATIONS

| Dataset | SGD | | SCG | | LeapFrog | |
|---|---|---|---|---|---|---|
| | RMSE | $h$ | RMSE | $h$ | RMSE | $h$ |
| Polynomial | **0.955** | **28** | 1.413 | 32 | 1.035 | 34 |
| Sinusoidal | 0.389 | 17 | 0.490 | 22 | **0.274** | **30** |
| Gaussian Mix | **0.224** | **30** | 0.237 | 23 | 0.226 | 16 |

*1) Dataset-Specific Performance:* **Polynomial Regression Results:** SGD achieved superior performance with RMSE of 0.955 ± 0.210 using 28 hidden units with learning rate $\eta = 0.05$ and aggressive momentum $\mu = 0.99$. LeapFrog reached RMSE 1.035 ± 0.170 with 34 hidden units. SCG exhibited weaker performance with RMSE 1.413 ± 0.256 using 32 hidden units, suggesting second-order curvature estimation struggles with polynomial optimisation landscapes despite using larger architectures.

**Sinusoidal Function Performance:** LeapFrog achieved the best performance on the 2D sinusoidal task with RMSE 0.274 ± 0.007 using 30 hidden units. SGD reached RMSE 0.389 ± 0.116 with 17 hidden units using learning rate $\eta = 0.1$ and aggressive momentum $\mu = 0.99$. SCG achieved RMSE 0.490 ± 0.053 with 22 hidden units. The results reveal LeapFrog has an advantage on periodic regression tasks.

**Gaussian Mixture Results:** The highest complexity function approximation task revealed close performance across algorithms. SGD achieved the best performance with RMSE 0.224 ± 0.007 using 30 hidden units and momentum $\mu = 0.9$, marginally outperforming LeapFrog with RMSE 0.226 ± 0.009 using 16 units and SCG with RMSE 0.237 ± 0.012 using 23 units. The tight clustering of results demonstrates that all three algorithms handle multi-modal regression effectively.

### C. Computational Efficiency and Convergence Analysis

Beyond final accuracy, computational efficiency determines practical algorithm applicability. This investigation employs two convergence metrics: epoch count until convergence and wall-clock time to convergence.

*1) Convergence Measurement Methodology:* Training convergence is determined using algorithm-specific stopping criteria with a maximum of 2 000 epochs. SGD employs validation-based early stopping with patience of 100 epochs without improvement, marking convergence at the epoch achieving best validation loss. SCG and LeapFrog employ gradient-based convergence detection through stability tracking: convergence is marked when the gradient norm remains stable for consecutive epochs, indicating the algorithm has reached a stationary point. All algorithms track the epoch achieving best validation loss and record wall-clock time at that epoch.

Each algorithm configuration undergoes 10-fold stratified cross-validation, where each fold trains independently and records its own convergence epoch and wall-clock time. The convergence metrics reported in Tables III and IV represent mean values averaged across all 10 folds.

*2) Convergence Results:* Table III presents convergence metrics across classification datasets.

SGD achieves fastest convergence on Iris with 41.1 epochs and 0.016s, reaching cross-entropy loss 0.054. LeapFrog demonstrates rapid convergence on Beer with 48.0 epochs and 0.021s, achieving loss 0.340, while SGD requires 1400.1 epochs and 0.363s to reach loss 0.327. SCG converges on Iris with 193.5 epochs and 0.191s, with a loss of 0.081, but fails to converge within 2 000 epochs on Titanic and Beer. LeapFrog similarly fails to converge on Titanic, where only SGD reaches convergence at 832.7 epochs (loss 0.405). The failure to converge indicates that gradient-based stopping criteria prove too strict for SCG and LeapFrog on complex binary and multi-class problems, with multiple folds reaching the 2000-epoch limit.

TABLE III
CONVERGENCE ANALYSIS FOR CLASSIFICATION DATASETS

| | SGD | | SCG | | LeapFrog | |
|---|---|---|---|---|---|---|
| Dataset | Ep. | T (s) | Ep. | T (s) | Ep. | T (s) |
| Iris | **41.1** | **0.016** | 193.5 | 0.191 | 59.9 | 0.039 |
| Titanic | **832.7** | **0.237** | – | – | – | – |
| Beer | 1400.1 | 0.363 | – | – | **48.0** | **0.021** |

TABLE IV
CONVERGENCE ANALYSIS FOR FUNCTION APPROXIMATION DATASETS

| | SGD | | SCG | | LeapFrog | |
|---|---|---|---|---|---|---|
| Dataset | Ep. | T (s) | Ep. | T (s) | Ep. | T (s) |
| Polynomial | 160.8 | **0.042** | 128.7 | 0.090 | **102.7** | 0.050 |
| Sinusoidal | **288.1** | **0.080** | – | – | 998.5 | 0.618 |
| Gaussian Mix | **675.0** | **0.197** | 814.5 | 0.626 | 718.3 | 0.300 |

All three algorithms converge successfully on polynomial regression. SGD achieves fastest wall-clock time at 0.042s despite requiring 160.8 epochs (MSE 0.012), demonstrating low per-epoch computational cost. LeapFrog converges in fewest epochs at 102.7 but requires 0.050s (MSE 0.026), while SCG requires 128.7 epochs and 0.090s (MSE 0.057). On sinusoidal regression, SGD converges at 288.1 epochs and 0.080s (MSE 0.585), while SCG fails to converge within 2000 epochs and LeapFrog requires 998.5 epochs and 0.618s (MSE 0.282). The Gaussian mixture surface proves challenging for all algorithms, requiring 675–814 epochs, with SGD achieving fastest time at 0.197s and lowest MSE 0.156, though SCG experiences convergence difficulties with eight folds reaching the epoch limit (MSE 0.175).

*3) Training Loss Curves:* Figures 2 through 5 present training dynamics from independent visualisation runs using optimal hyperparameters identified in Tables I and II. Loss curves show average validation loss across 10 cross-validation folds. Convergence epochs and loss values differ from Tables III and IV due to stochastic variation across independent runs, though both reflect identical optimal configurations.

The loss curves reveal distinct algorithmic convergence characteristics. SGD demonstrates steady monotonic descent with smooth loss reduction, exhibiting reliable but often slow convergence requiring careful momentum tuning. LeapFrog displays physics-inspired dynamics with rapid initial descent followed by oscillatory behaviour and occasional temporary loss increases before final convergence, reflecting the algorithm's kinetic energy dissipation mechanism. SCG achieves smooth exponential-like decay through automatic step size adaptation, occasionally exhibiting sudden loss drops when

conjugate directions align favourably with the error surface geometry.

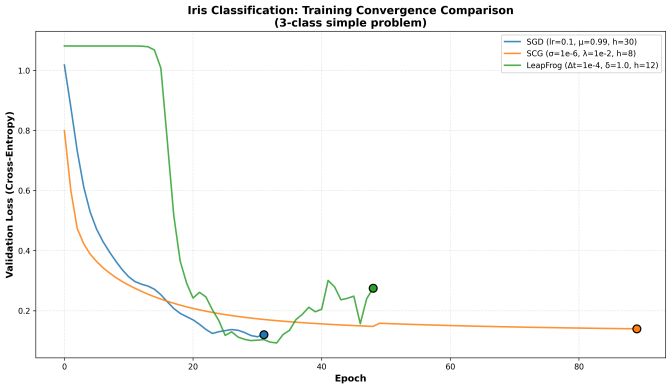


Fig. 2. Training convergence on Iris classification

Figure 2 shows SGD converging at epoch 32 with loss 0.120, SCG at epoch 90 with loss 0.139, and LeapFrog at epoch 49 with loss 0.275.

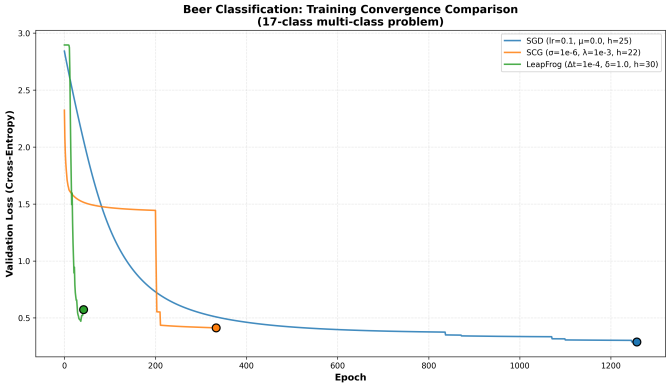


Fig. 3. Training convergence on Beer classification

Figure 3 shows LeapFrog achieving rapid convergence at epoch 43 with loss 0.573, SGD requiring 1258 epochs with loss 0.289, while SCG converges prematurely at epoch 334 with high loss 0.413.

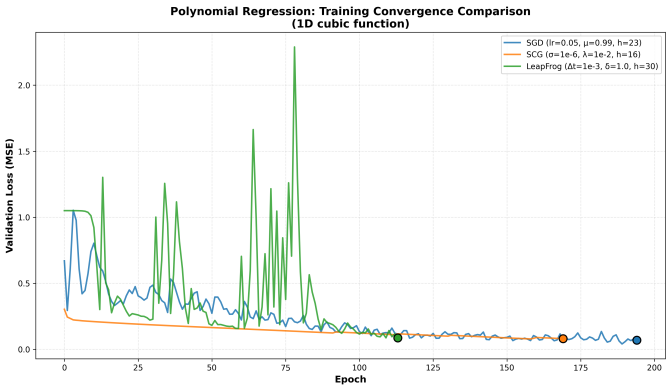


Fig. 4. Training convergence on Polynomial regression

Figure 4 shows LeapFrog converging at epoch 114 with MSE 0.086, SGD at epoch 195 with MSE 0.069, and SCG at epoch 170 with MSE 0.081.

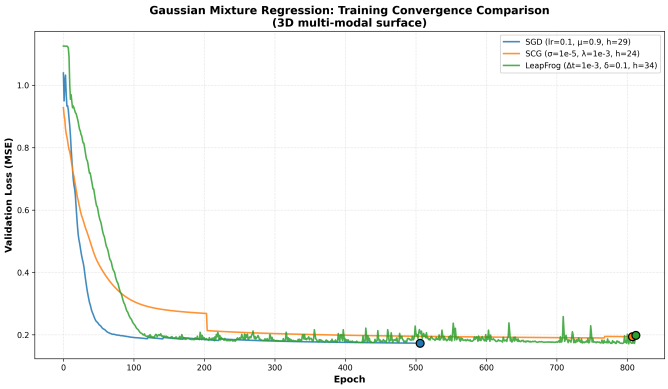


Fig. 5. Training convergence on Gaussian mixture regression

Figure 5 shows SGD converging at epoch 507 with MSE 0.173, LeapFrog at epoch 813 with MSE 0.198, while SCG converges at epoch 808 with MSE 0.194.

### *D. Parameter Sensitivity Analysis*

Beyond identifying optimal configurations, quantifying parameter sensitivity reveals algorithmic robustness characteristics. The coefficient of variation provides a standardised measure of parameter sensitivity, computed as the ratio of standard deviation to mean performance across all Phase 1 parameter combinations tested per algorithm. Lower coefficients indicate robust performance across parameter configurations, while higher coefficients reveal strong sensitivity requiring careful tuning.

Table V presents sensitivity analysis results across all datasets. The analysis reveals striking differences in algorithmic robustness.

TABLE V
PARAMETER SENSITIVITY ANALYSIS VIA COEFFICIENT OF VARIATION ACROSS PHASE 1 PARAMETER COMBINATIONS

| Dataset | SGD CV (%) | SCG CV (%) | LeapFrog CV (%) |
|---|---|---|---|
| Iris | 3.78 | **0.53** | 1.67 |
| Titanic | 3.47 | **0.51** | 0.90 |
| Beer | 23.01 | 15.21 | **1.58** |
| Polynomial | 268.39 | 6.93 | **6.72** |
| Sinusoidal | 3.70 | **2.20** | 24.45 |
| Gaussian Mix | 32.38 | **7.44** | 17.96 |

SCG demonstrates superior parameter robustness across four of six datasets with 0.53% on Iris, 0.51% on Titanic, 2.20% on Sinusoidal, and 7.44% on Gaussian mixture, validating theoretical parameter-free advantage through automatic step size adaptation. LeapFrog exhibits lowest sensitivity on Beer and Polynomial with 1.58% and 6.72% respectively, but requires precise configuration for Sinusoidal functions with 24.45% variation. SGD shows extreme sensitivity on

Polynomial regression at 268.39% and Gaussian mixture at 32.38%, underscoring critical importance of momentum tuning from 0.0 to 0.99 across tasks.

### *E. Statistical Significance*

The Friedman test, a non-parametric equivalent to repeated measures ANOVA, was employed to detect performance differences across the three training algorithms. The Friedman test operates on cross-validation fold scores, ranking algorithms per fold and testing whether rank distributions differ significantly. When the Friedman test indicated significance at $\alpha = 0.05$, Nemenyi post-hoc pairwise comparisons identified which specific algorithm pairs differed significantly while controlling family-wise error rate.

Table VI presents the statistical test results across all datasets.

TABLE VI
STATISTICAL SIGNIFICANCE ANALYSIS USING FRIEDMAN TEST WITH NEMENYI POST-HOC COMPARISONS

| Dataset | $p$-value | Significant |
|---|---|---|
| Iris | 0.097 | No |
| Titanic | 0.034 | Yes |
| Beer | 0.007 | Yes |
| Polynomial | 0.0005 | Yes |
| Sinusoidal | 0.0018 | Yes |
| Gaussian Mix | 0.020 | Yes |

Five of six datasets demonstrate statistically significant algorithmic differences at $\alpha = 0.05$. Only Iris classification shows no significant difference with $p = 0.097$, suggesting observed performance variations fall within cross-validation variance on simple three-class problems. Titanic and Beer classification achieve significance with $p = 0.034$ and $p = 0.007$ respectively, validating SGD superiority on binary and multi-class problems. All three regression tasks demonstrate strong significance: Polynomial with $p = 0.0005$, Sinusoidal with $p = 0.0018$, and Gaussian mixture with $p = 0.020$. The universal significance across complex problems confirms that algorithm selection meaningfully impacts performance beyond random variation.

### *F. Comparative Algorithm Analysis*

SGD achieved superior performance on four of six tasks through systematic momentum tuning from 0.0 to 0.99. The algorithm won all three classification tasks with 98.67% on Iris, 83.72% on Titanic, and 90.15% on Beer, plus Polynomial and Gaussian mixture regression with RMSE 0.955 and 0.224 respectively. Momentum requirements vary dramatically: aggressive momentum of $\mu = 0.99$ optimises Iris classification and Polynomial regression, zero momentum suits Titanic binary classification, while moderate momentum $\mu = 0.5$ handles Beer multi-class problems. SGD favored architectures from 18 to 30 hidden units, with convergence ranging from 41.1 epochs on Iris to 1400.1 epochs on Beer. Parameter sensitivity reached 268.39% on Polynomial regression, underscoring critical importance of momentum tuning.

LeapFrog demonstrated advantage on Sinusoidal regression with RMSE 0.274, achieving rapid convergence through physics-inspired dynamics. The algorithm proved effective on Beer classification, converging at 48.0 epochs compared to SGD 1400.1 epochs. LeapFrog preferred larger architectures spanning 15 to 34 hidden units. However, the algorithm failed to converge on Titanic within 2 000 epochs and required 998.5 epochs on Sinusoidal, indicating gradient-based stopping criteria prove too strict for certain problems. Parameter sensitivity ranged from 1.58% to 24.45%.

SCG validated theoretical parameter-free advantage with low coefficient of variation below 2.20% on Iris, Titanic, and Sinusoidal, achieving competitive performance with minimal tuning. The algorithm favored compact architectures from 6 to 23 hidden units. However, SCG encountered severe numerical instability on Beer classification with 69.67% accuracy and 21.58% variance, failing to converge within 2000 epochs. SCG similarly failed to converge on Titanic and Sinusoidal regression, with multiple folds reaching the epoch limit. The findings indicate gradient-based convergence criteria and second-order curvature estimation struggle on complex multi-class problems and certain regression landscapes.

## VI. CONCLUSIONS

This empirical study compared three feedforward neural network training algorithms across classification and function approximation tasks of varying complexity, revealing distinct performance characteristics for each approach.

**Key Findings:** Stochastic Gradient Descent (SGD) achieved superior performance on four of six tasks through momentum tuning from 0.0 to 0.99: aggressive momentum optimised Iris and Polynomial tasks, zero momentum suited Titanic binary classification, while moderate momentum handled Beer multi-class problems. This momentum variation demonstrates that requirements depend on loss landscape geometry and class complexity. LeapFrog demonstrated advantage on Sinusoidal regression with root mean squared error (RMSE) 0.274 and rapid convergence on Beer classification at 48.0 epochs. Scaled Conjugate Gradient (SCG) validated theoretical parameter-free advantage with coefficient of variation below 2.20% on simple problems, but encountered severe failures to converge on Titanic, Beer, and Sinusoidal tasks, with Beer accuracy collapsing to 69.67% and 21.58% variance.

**Statistical Significance:** Five of six datasets achieved statistical significance at $\alpha = 0.05$. Titanic and Beer classification plus all three regression tasks demonstrated $p \leq 0.02$, validating meaningful algorithmic differences. Only Iris classification showed no significance with $p = 0.097$, indicating simple three-class problems fall within cross-validation variance.

**Practical Recommendations:** SGD with properly tuned momentum is recommended for most tasks, achieving statistically superior accuracy on Titanic and Beer classification plus Polynomial and Gaussian regression through task-specific momentum from 0.0 to 0.99. LeapFrog suits Sinusoidal regression and problems requiring rapid convergence, demonstrating statistical advantage on periodic functions. SCG enables rapid

prototyping on simple problems like Iris with minimal tuning but should be avoided for complex tasks due to convergence failures and numerical instability. Statistical significance on five of six datasets validates that algorithm selection meaningfully impacts performance beyond random variation.

## REFERENCES


[1] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimisation," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[2] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, JMLR Workshop and Conference Proceedings, JMLR.org, pp. 249–256, 2010.

[3] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, pp. 525–533, 1993.

[4] I. Panageas and G. Piliouras, "Gradient descent only converges to minimizers: Non-isolated critical points and invariant regions," in *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, vol. 67, LIPIcs, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 2:1–2:12, 2017.

[5] J. A. Snyman, "A new and dynamic method for unconstrained minimization," *Applied Mathematical Modelling*, vol. 6, pp. 449–462, 1982.

[6] J. A. Snyman, "An improved version of the original leap-frog dynamic method for unconstrained minimization: LFOP1(b)," *Applied Mathematical Modelling*, vol. 7, pp. 216–218, 1983.


## APPENDIX A
## LIST OF ACRONYMS

- **MSE**: mean squared error
- **RMSE**: root mean squared error
- **SCG**: Scaled Conjugate Gradient
- **SGD**: Stochastic Gradient Descent

## APPENDIX B
## OPTIMAL HYPERPARAMETERS

Tables VII, VIII, and IX present the complete set of optimal hyperparameters identified through the two-phase optimisation process for all algorithm-dataset combinations.

TABLE VII
OPTIMAL SGD HYPERPARAMETERS

| Dataset | $\eta$ | $\mu$ | $h$ |
|---|---|---|---|
| Iris | 0.05 | 0.99 | 19 |
| Titanic | 0.1 | 0.0 | 22 |
| Beer | 0.05 | 0.5 | 18 |
| Polynomial | 0.05 | 0.99 | 28 |
| Sinusoidal | 0.1 | 0.99 | 17 |
| Gaussian Mix | 0.1 | 0.9 | 30 |

TABLE VIII
OPTIMAL SCG HYPERPARAMETERS

| Dataset | $\sigma_0$ | $\lambda$ | $h$ |
|---|---|---|---|
| Iris | $10^{-6}$ | $10^{-3}$ | 6 |
| Titanic | $10^{-5}$ | $10^{-3}$ | 16 |
| Beer | $10^{-6}$ | $10^{-3}$ | 10 |
| Polynomial | $10^{-5}$ | $10^{-2}$ | 32 |
| Sinusoidal | $10^{-6}$ | $10^{-2}$ | 22 |
| Gaussian Mix | $10^{-5}$ | $10^{-3}$ | 23 |

TABLE IX
OPTIMAL LEAPFROG HYPERPARAMETERS

| Dataset | $\Delta t$ | $\delta$ | $\xi$ | $m$ | $h$ |
|---|---|---|---|---|---|
| Iris | $1 \times 10^{-4}$ | 1.0 | 0.1 | 5 | 15 |
| Titanic | $1 \times 10^{-3}$ | 0.1 | 0.2 | 5 | 27 |
| Beer | $1 \times 10^{-4}$ | 1.0 | 0.2 | 3 | 34 |
| Polynomial | $1 \times 10^{-3}$ | 1.0 | 0.1 | 5 | 34 |
| Sinusoidal | $1 \times 10^{-4}$ | 1.0 | 0.1 | 5 | 30 |
| Gaussian Mix | $1 \times 10^{-4}$ | 0.1 | 0.1 | 3 | 16 |