ML Project Report -Custom Neural Network Simulator (Type A, Python)

Authors: C. Botticelli, G. M. Tumminello, A. Stocco.

Team's name: PlusUltra.

Course / Year: Machine Learning 2025-2026

Project type: A (implemented in Python)

INDEX

1.0 Abstract

This project presents the design and implementation of a neural network framework developed from scratch in Python. The framework supports multiple training algorithms, regularization strategies, and validation schemes. Experiments were conducted on the MONK's classification benchmarks and on the CUP regression task. For all the problems different training approaches (standard backpropagation and a cascade-correlation–inspired method) were compared. Results highlight the impact of architecture, regularization, and model selection strategies on generalization performance.

2.0 Introduction and Objectives

The project's obective is to study the behavior and performance of feed-forward neural networks trained with different learning strategies. This work covers classification (MONK dataset) and multi-output regression (CUP dataset).

Main objectives:

• To analyze learning dynamics and generalization behavior.

• To evaluate alternative training approaches beyond standard backpropagation.

3.0 Method

3.1 Software and implementation

A fully custom neural network simulator was implemented, allowing full control over:

• Network architecture.

• Weight initializer algorithms.

• Training algorithms.

• Loss functions.

• Momentum.

Main characteristics:

• Fully-connected feed-forward neural networks.

• Modular architecture.

• Support for classification and regression tasks.

• Ensemble methods were adopted for the MONK classification tasks.

• Multiple independently trained networks were combined.

• Final predictions were obtained by averaging model outputs.

• Ensemble models were used to reduce variance and improve robustness.

No ensemble was used for the CUP regression task.

3.2 Network architecture and training

Architectures:

• Single and multi-hidden-layer MLPs.

• Number of hidden units selected via model selection.

• Activation functions: Sigmoid, Tanh.

• Loss functions: Half MSE, MSE, MEE, MAE, Logcosh, Binary Crossentropy, Huber.

Training algorithms:

• Stochastic Gradient Descent (SGD).

• SGD with momentum.

• RPROP and QuickProp.

Optimization details:

• Mini-batch training.

• Learning rate decay on validation plateau.

• Early stopping based on validation error.

Regularization: L2 (Tikhonov) regularization.

3.3 Tools/Libraries

• Numpy for vector operations.

• Matplotlib for learning curves.

• Pandas for dataset loading.

• Graphviz to draw network graphs.

3.4 Alternative Training Strategy

We further implemented a Cascade Correlation model to explore the limits of fixed-topology networks. The primary goal of this work was to conduct a comparative analysis against our team's neural network, which served as the experimental baseline. By implementing a constructive algorithm that adds hidden units automatically, we investigated if a dynamic architecture could outperform the manual trial-and-error design used in standard backpropagation networks. The architecture was then applied on Monk 1,2,3 classification benchmarks and run multi-output regression on ML-CUP25 dataset.

• Number of hidden units: fixed at 20 and controlled by an hyperparameter;

• Activation function: tanh to map inputs/outputs in the range [-1, 1] both in the hidden and output units;

• Training algorithms: Quickprop and Rprop to speed up convergence and Gradient Ascent to maximize the correlation score S;

• Learning mode: Full-Batch learning to ensure stable convergence statistics.

The preliminary screening phase revealed that the constructive nature of the Cascade algorithm drastically alters the loss landscape compared to static topologies. Consequently, standard patience settings proved inadequate, requiring a calibrated two-stage search protocol (Coarse and Fine) to identify the optimal balance between learning rates and tolerance thresholds. We explored a comprehensive hyperparameter space including learning rate, patience, tolerance, and the regularization factor. The architecture dynamically toggled between QuickProp and RProp update rules to maximize convergence speed. The search results highlighted that while deeper networks reduced training error, they required significant L2 regularization to maintain generalization capability on the validation set.

3.5 Preprocessing

Input normalization:

• MONK: one-hot encoding.

• CUP: min–max normalization.

Output normalization (CUP):

• Targets scaled to symmetric ranges compatible with Tanh.

• Denormalization applied before computing final performance metrics.

Dataset shuffle both for MONK's problems and CUP.

4.0 Model selection strategies

Grid search

• Grid search with exhaustive exploration of: learning rate, L2 regularization parameter, momentum, network architecture.

• Multiple random initializations per configuration.

• Average validation performance used for comparison.

4.1 Final Model Selection

Selection criterion: lowest validation loss, stable learning curve and limited gap between training and validation error.

Final model characteristics:

• Multi-layer architecture.

• Tanh activation.

• SGD with momentum.

• L2 regularization.

• Early stopping.

MONK 1:

• Architecture: [17, 4, 1]

• Training algorithm: SGD.

• Momentum: 0.9.

• Batch size: 1 (SGD online).

• Max epochs: 1500.

• Early stopping patience: 50.

• Actual epochs trained: 242.

• L2 lambda: 0.0.

• Decay: 0.9.

• Activation function: sigmoid.

• Loss function: half MSE.

• Weight initializer: Xavier initialization.

• Random seed: 792.

Grid search MONK 1

• Learning rates tested: [0.3, 0.4, 0.5, 0.6, 0.7, 0.8].

• Seeds per learning rate: 5.

• Best learning rate found: 0.3.

MONK 2:

• Architecture: [17, 8, 1]

• Training algorithm: SGD.

• Momentum: 0.9.

• Batch size: 1 (SGD online).

• Max epochs: 2000.

• Early stopping patience: 50.

• Actual epochs trained: 235.

• L2 lambda: 0.0.

• Decay: 0.9.

• Activation function: sigmoid.

• Loss function: half MSE.

• Weight initializer: Xavier initialization.

• Random seed: 792.

Grid search MONK 2

• Learning rates tested: [0.05, 0.1, 0.15, 0.2, 0.25, 0.3]

• Seeds per learning rate: 5.

• Best learning rate found: 0.1.

Grid search MONK 3

• Architecture: [17, 6, 1]

• Training algorithm: SGD.

• Momentum: 0.85.

Batch size: 1 (SGD online).

• Max epochs: 2000.

• Early stopping patience: 50.

• Actual epochs trained: 235.

• L2 lambda: 0.02.

• Decay: 0.9.

• Activation function: sigmoid.

• Loss function: half MSE.

• Weight initializer: Xavier initialization.

• Random seed: 496.

MONK 3:

Grid search MONK 3

• Learning rates tested: [0.05, 0.1, 0.15, 0.2, 0.25, 0.3]

• Seeds per learning rate: 5.

• Best learning rate found: 0.05.

CUP:

• Training algorithm: SGD.

• Batch size: 16.

• Max epochs: 2000.

• Early stopping patience: 300.

• Decay: 0.98.

• Mu: 1.75.

• Eta_plus: 1.2

• Eta_minus: 0.5

• Activation function: tanh.

• Training Loss function: half MSE.

• Validation Loss function: MEE.

• Weight initializer: Xavier initialization.

Grid search CUP

• Learning rates tested: [0.005, 0.001, 0.002]

• Best learning rate found: 0.3.

• L2 lambda tested: [0.00005, 0.0001, 0.0005]

• Best L2 lambda found: 0.00005.

• Momentums tested: [0.9, 0.95]

• Best momentum found: 0.9.

• Architectures tested: [12, 60, 4], [12, 80, 4], [12, 50, 30, 4], [12, 60, 40, 4], [12, 80, 60, 4], [12, 60, 40, 20, 4]

• Best architecture found: [12, 60, 40, 20, 4]

• Best seed: 281.

6.0 Validation schema

MONK datasets:

• Predefined training and test sets.

• Test set used only once for final evaluation.

CUP dataset:

• Explicit split into Training / Validation / Internal Test sets.

• Validation set used for model selection.

• Internal test set used for final assessment before blind test.

7.0 Results

Monk's results

Task	Architecture	Loss type	Training algorithm	Momentum	Max epochs	Batch size	patience	Actual epochs trained	L2 lambda	Decay	Activation function	Weight initializer	Half MSE (TR/ VS/TS)	Accuracy (TR /VS/TS) (%)
MONK 1	[17, 5, 1]	Half MSE	SGD	0.9	1500	1 (SGD online)	50	242	0.0	0.9	Sigmoid	Xavier	0.000364/ 0.000364/ 0.000358	100% /100% / 100%
MONK 2	[17, 8, 1]	Half MSE	SGD	0.9	2000	1 (SGD online)	50	235	0.0	0.9	Sigmoid	Xavier	0.000812/ 0.000812/ 0.000788	100% /100% / 100%
MONK 3	[17, 6, 1]	Half MSE	SGD	0.85	2000	1 (SGD online)	50	235	0.0	0.9	Sigmoid	Xavier		97.26% / 97.96% / 98.84 %
MONK 3 (reg)	[17, 6, 1]	Half MSE	SGD	0.85	2000	1 (SGD online)	50		/					

Table 1: Average predictions results for the Monk's task.

Task	Learning rates tested	L2 lambda tested	Best L2 lambda found	Best learning rate found	Seeds per learning rate	Best seed found
MONK 1	[0.3, 0.4, 0.5, 0.6, 0.7, 0.8]	/	/	0.1	5	223
MONK 2	[0.001, 0.002, 0.003, 0.005, 0.01]	/	/	0.005	5	
MONK 3	[0.05, 0.1, 0.15, 0.2, 0.25, 0.3]	/	/	0.05	5	
MONK 3 (reg)		[0.003, 0.0005, 0.005, 0.1, 0.15, 0.2, 0.25, 0.3]				

Table 2: Grid search for the Monk's task.

Task	Half MSE (TR/VS/TS)	Accuracy (TR/VS/TS) (%)
MONK 1	0.016839 / 0.019390 / 0.029984	98.8372% / 94.7368% / 93.5185%
MONK 2	[0.001, 0.002, 0.003, 0.005, 0.01]	5
MONK 3	[0.003, 0.0005, 0.005, 0.1, 0.15, 0.2, 0.25, 0.3]	5

Table 3: Ensamble results for the Monk's task.

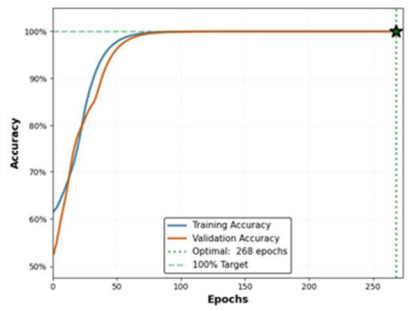
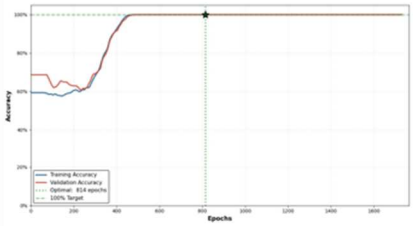
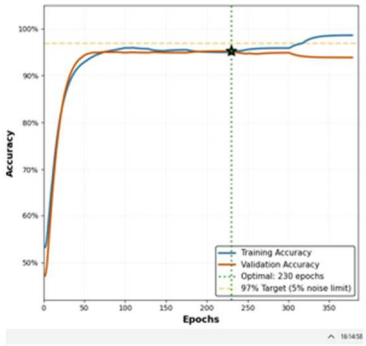
MONK 1	MONK 2	MONK 3
		
100% Accuracy	97% Accuracy	97% Accuracy

Table 4. Plot of the accuracy obtained on Monk's problems.

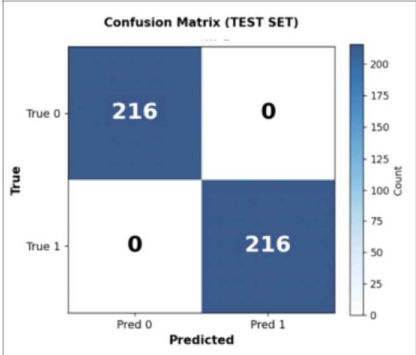
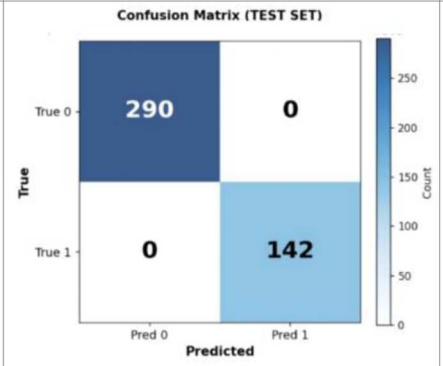
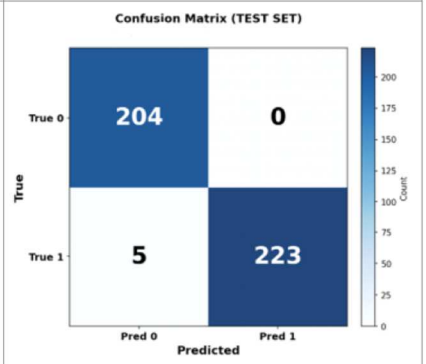
MONK 1	MONK 2	MONK 3
		

Figure 2: Confusion matrixes for the three MONK's benchmarks.

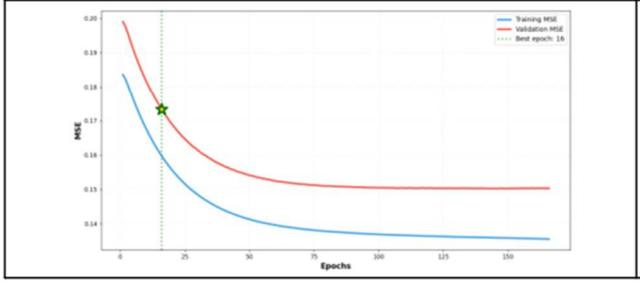
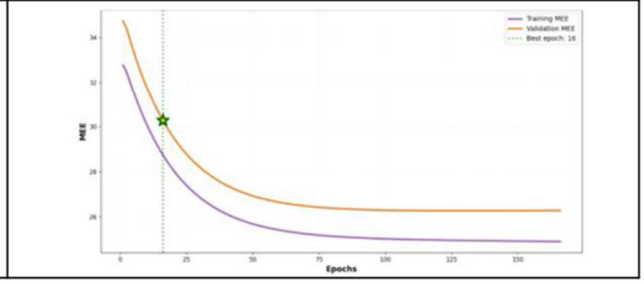
CUP's results

Task	Training algorithm	Loss type	Batch size	Max epochs	Patience	Decay	Actual epochs trained	μ	Weight initializer	Activation function	Seed
CUP	SGD	MSE	16	1500	150	0.98	242	1.75	Xavier	Tanh	345

Table 4: Average predictions results for the CUP's task.

Task	Learning rate tested	Best learning rate found	L2 Lambda tested	Best L2 Lambda found	Momentums tested	Best momentum found	Architectures tested	Best architecture found	MSE (TR) /MEE(VS) / MEE(TS)
CUP	[0.005, 0.001, 0.002]	0.005	[0.00005, 0,0001, 0.0005]	0.0005	[0.9, 0.95]	0.9	[12, 50, 30, 4], [12, 60, 40, 4]. [12. 80, 60, 4], [12, 60, 40, 20, 4], [12, 60, 4], [12, 80, 4]	[12, 60, 40, 4]	0.125161/ 26.2214/ 26.2644

Table 5: Grid search for the CUP's task.

MSE	MEE
	

8.0 Discussion

Regularization is essential for stable generalization on CUP. Larger architectures improve expressiveness but increase overfitting risk. Random grid search explores the hyperparameter space more efficiently than exhaustive grid search. The Cascade Correlation approach identified regularization as the single most critical factor for the Cascade-Correlation architecture on the CUP dataset; without it, the aggressive unit-addition strategy tends to overfit the training noise. Furthermore, while RProp demonstrated faster convergence in early stages, QuickProp provided superior stability as the network depth increased.

9.0 Conclusions

A complete neural network framework was successfully implemented from scratch. The experiments confirm theoretical expectations on bias–variance trade-off and regularization. Model selection strategy significantly impacts final performance and training cost. Alternative training strategies can achieve competitive results but require careful validation.

To reproduce the experiments, run simple_search_M1.py, simple_search_M2.py, or simple_search_M3.py to execute training and evaluation for the respective tasks. Run best_run_CUP.py to train the final model with the optimal hyperparameters found during the grid search and generate the loss/metric plots. Run simple_GS_CUP.py to reproduce the Grid Search process.

Run main_cup_cascade.py to execute the Cascade Correlation training on the CUP dataset or cascade_grid_search.py to explore the hyperparameter space for the constructive architecture.

10.0 References

Barron, J. T. (2019). A General and Adaptive Robust Loss Function. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4331-4340.

Borchani, H., Varando, G., Bielza, C., & Larrañaga, P. (2015). A survey on multi-output regression. *WIREs Data Mining and Knowledge Discovery*, 5(5), 216-233.

Elsebach, R. (1994). Evaluation of forecasts in AR models with outliers. *OR Spectrum*, 16, 41-45.

Fahlman, S. E. (1988). An Empirical Study of Learning Speed in Back-Propagation Networks. *Technical Report CMU-CS-88-1623*, Carnegie Mellon University.

Fahlman, S. E., & Lebiere, C. (1990). *The Cascade-Correlation Learning Architecture*. Advances in Neural Information Processing Systems 2 (NIPS 1989), pp. 524-532. Morgan Kaufmann.

Ghosh, A., Kumar, H., & Sastry, P. S. (2017). Robust Loss Functions under Label Noise for Deep Neural Networks.

Huber, P. J. (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1), 73-101.

Qi, J., Du, J., Siniscalchi, S. M., Ma, X., & Lee, C.-H. (2020). On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression.

Riedmiller, M., & Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *IEEE International Conference on Neural Networks (ICNN)*, pp. 586-591.

Sun, Q., Zhou, W.-X., & Fan, J. (2020). Adaptive Huber Regression. *Journal of the American Statistical Association*, 115(529), 254-265.

Volarić, T., Ljubić, H., Vasić, D., & Rozić, R. (2025). Should MSE Remain the Default Benchmarking Loss Function for Long-Term Time Series Forecasting? *SN Computer Science*, 6:804.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 785–794.

Grover, A., & Ermon, S. (2019). Uncertainty Autoencoders: Learning Compressed Representations via Variational Information Maximization. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*.