Systems & Control Transactions

Research Article - Peer Reviewed Conference Proceeding
ESCAPE 35 - European Symposium on Computer Aided Process Engineering
Ghent, Belgium. 6-9 July 2025
Jan F.M. Van Impe, Grégoire Léonard, Satyajeet S. Bhonsale,
Monika E. Polańska, Filip Logist (Eds.)

PSE PRESS

A Novel Approach to Gradient Evaluation and Efficient Deep Learning: A Hybrid Method

Bogdan Dorneanu[a], Vasileios K. Mappas[a], and Harvey Arellano-Garcia[a]*

[a] Brandenburgische Technische Universitat Cottbus-Senftenberg, FG Prozess- und Anlagentechnik, Cottbus, Germany
* Corresponding Author: arellano@b-tu.de.

ABSTRACT

Deep learning faces significant challenges in efficiently training large-scale models. These issues are closely linked, as efficient training often depends on precise and computationally feasible gradient calculations. This work introduces innovative methodologies to improve deep learning network (DLN) training in complex systems. A novel approach to DLN training is proposed by adapting the block coordinate descent (BCD) method, which optimizes individual layers sequentially. This is combined with traditional batch-based training to create a hybrid method that harnesses the strengths of both techniques. Additionally, the study explores Iterated Control Random Search (ICRS) for initializing parameters and applies quasi-Newton methods like L-BFGS with restricted iterations to enhance optimization. By tackling DLN training efficiency, this contribution offers a comprehensive framework to address key challenges in modern machine learning. The proposed methods improve scalability and effectiveness, especially for handling complex real-world problems. Examples from Process Systems Engineering illustrate how these advancements can directly enhance the training of large-scale systems.

Keywords: Artificial Intelligence, Machine Learning, Numerical Methods.

INTRODUCTION

Deep learning, a specialized subset of machine learning, utilizes artificial neural networks with multiple layers to model and learn intricate patterns from large datasets. Unlike traditional machine learning, which often relies on manually engineered features, deep learning autonomously extracts hierarchical representations of data through its layered architecture. This capability enables it to address highly complex problems in domains such as image and speech recognition, natural language processing, drug discovery, and genomics [1]. By learning directly from raw data, deep learning reduces the need for domain-specific feature engineering, offering remarkable flexibility and scalability.

Despite this transformative potential, deep learning faces several challenges. One of the primary obstacles is computational complexity. Training deep neural networks (DNNs) requires substantial resources, including high-performance GPUs and significant memory capacity for data storage. These requirements make deep learning both resource- and cost-intensive. Additionally, deep models demand large amounts of data to achieve high accuracy, posing a limitation in fields where data collection is sparse or expensive.

Optimization of deep learning models presents another significant challenge. Training involves tuning numerous hyperparameters (i.e., learning rates, layer configurations), which is both time-consuming and computationally demanding. Various optimization techniques, including stochastic gradient descent (SGD) or ADAM have been developed, to improve training efficiency and convergence. However, the process remains difficult due to the complexity of deep architectures.

Despite these challenges, significant advancements continue to improve the robustness and efficiency of deep learning systems. The training of deep learning networks (DLNs) involves optimizing a loss function to minimize the difference between model prediction and actual targets. Backpropagation, introduced by Rumelhart et al. [2], remains the foundational algorithm for training DLNs, efficiently calculating gradients using the chain rule. However, backpropagation faces challenges such as vanishing gradients in deep networks, which slow or halt

learning [3]. Techniques like batch training and optimization methods such as stochastic gradient descent (SGD) or ADAM have been developed to address these issues, improving convergence and computational efficiency. Recent advances include block coordinate descent (BCD) [4], which updates parameters layer by layer to reduce computational overhead, and its application in binary neural networks for resource-constrained devices.

While the algorithmic logic of such methods is implementation-agnostic, the practical deployment of deep learning frameworks can benefit significantly from modern software engineering practices. Object-oriented programming (OOP) facilitates modular and reusable code structures, which are particularly valuable for building and scaling complex deep learning systems. Frameworks like TensorFlow or PyTorch adopt OOP principles to encapsulate components such as layers, optimizers, and loss functions, facilitating scalability and distributed training [5]. These innovations collectively address the growing complexity of DLNs while improving training efficiency and adaptability across diverse hardware environments. The proposed framework, while using algorithmic innovations of BCD and ICRS, leverages the advantages of OOP to create a flexible and extensible platform. The following sections will detail the algorithmic aspects of the approach, and where relevant, highlight how OOP principles contribute to its practical implementation.

METHODOLOGY

This contribution introduces a novel approach to deep learning optimization by addressing key limitations in traditional methods, particularly in computational efficiency, scalability, and flexibility. Unlike conventional techniques such as SGD and ADAM, which optimize all parameters simultaneously and often struggle with the complexity of deep networks, the proposed framework illustrated in Figure 1 leverages a Block Coordinate Descent (BCD)-based strategy.

This method dynamically identifies and optimizes the most influential layers using sensitivity measures [3], ensuring computational resources are allocated effectively. The modularity enabled by the proposed OOP design allows to easily swap and configure different BCD variants, such as those using different sensitivity measures or layer selection strategies.

To further enhance global exploration and avoid local minima, the framework integrates Iterated Control Random Search (ICRS) [6], a global optimization technique that efficiently explores non-convex parameter spaces. By combining ICRS for initial exploration with BCD for layer-specific refinement, the framework achieves more robust convergence and improved training stability for complex neural architectures. Additionally, the framework incorporates object-oriented design to overcome the modularity and adaptability of existing deep learning tools. The ICRS optimizer is implemented as a separate class that can be easily integrated with other components of the framework due to the well-defined interfaces of the OOP design. Similarly, different optimization algorithms (Table 1) are encapsulated within dedicated classes, adhering to a common interface, facilitating easy swapping and comparison of optimization strategies. The framework is implemented in Python 3.11 on an AMD Ryzen 9 7950X 16-core processor with 56GB RAM.

It is important to clarify that the algorithm depicted in Figure 1 represents the computational flow of the proposed hybrid training approach and is, in principle, independent of any specific programming paradigm. The core logic of BCD and ICRS could be implemented in various ways. However, an OOP approach is chosen to enhance code organisation, reusability and maintainability. Specifically, the framework is structured around classes representing different layers, optimizers and training strategies. This facilitates easy swapping and modification of individual components without affecting the rest of the system.

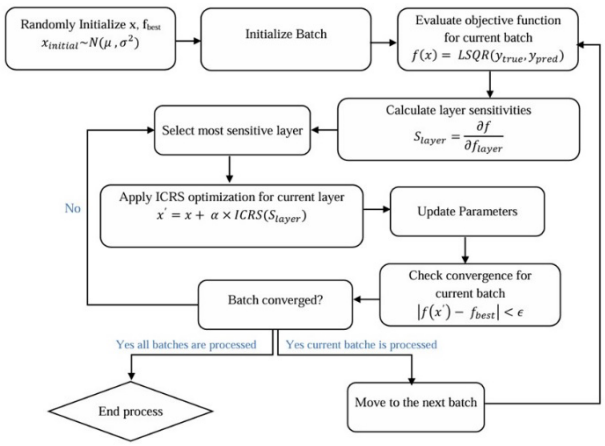

Figure 1. OOP deep learning network training algorithm

Table 1: Optimization methods

Optimizer	Type
ADAM	Stochastic gradient-based
L-BFGS	Quasi-Newton gradient-based
Barzilai-Borwein	Gradient-based
ICRS	Derivative free

CASE STUDIES

To evaluate the performance of the proposed deep learning framework, two datasets from the UCI Machine Learning Repository [7] are utilized: the combined cycle power plant (CCPP) dataset, and the steel industry energy consumption (SIEC). The CCPP dataset consists of

9,568 samples with four features and one target variable. This dataset captures data from a CCPP where electricity is generated through gas and steam turbines working in tandem, making it a well-suited benchmark for regression tasks in industrial energy systems.

The SIEC dataset contains 35,040 samples with fifteen features and one target variable. The data is collected from a smart small-scale steel industry in South Korea at 15-minute intervals over a year, providing a high-resolution temporal dataset suitable for time-series forecasting and energy efficiency analysis.

For the CCPP dataset, a fully connected deep learning network was implemented, with an input layer of four neurons (one per feature), followed by hidden layers with varying configurations (i.e., 5,5,5; 10,10,10, etc.) and an output layer consisting of a single neuron for predicting power output. Each layer type (input, hidden, output) is implemented as a separate class, allowing for easy modification of layer parameters and the addition of new layer types without affecting other parts of the network.

To further enhance performance, four distinct block-selection strategies were explored: most sensitive layer selection, random layer selection, forward sequential selection, and ping-pong directional updates. The model's accuracy was assessed using the coefficient of determination (R^2), mean squared error (MSE), and mean absolute error (MAE).

For the SIEC dataset, a similar architecture is adopted with fifteen input neurons in the input layer and sensitivity-based block selection is employed to adaptively refine layer-wise updates. Various activation functions, including sigmoid, Tanh, ReLU and Leaky ReLU activation functions, are tested in the hidden layers for both datasets, to evaluate their impact on learning dynamics. Each activation function is implemented as a distinct class. This allows for easy experimentation with new activation functions by simply creating a new class that implements the required interface, without modifying the core network architecture.

Hyperparameters for both datasets were tuned to ensure effective and stable learning and included parameters such as batch subset size set to 1.1 times the number of network parameters and sensitivity tolerance adjusted to 10 times the projected gradient tolerance. The tuning process involved systematically exploring different combinations of learning rates, batch sizes, etc. to optimize model performance.

RESULTS AND DISCUSSION

The CCPP dataset

The first stage of evaluating the performance of the proposed DLN training framework looked into the different types of activation functions, while keeping the network architecture and hyperparameters constant.

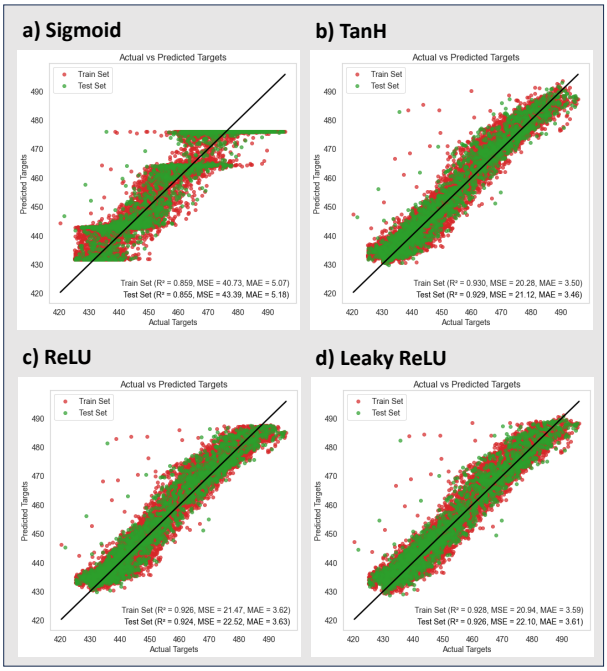

Figure 2. Performance comparison for various activation functions

As illustrated in Figure 2, Tanh emerges as the most reliable activation function, achieving high R^2 values (0.929 for the test set) and low MSE/MAE across multiple runs. LeakyReLU performs comparably but exhibits slightly higher variability in predictions. ReLU shows good performance but has more scattered test points compared to Tanh and LeakyReLU. Sigmoid performs the worst due to higher residual variance and non-uniform distributions. Visualizations of actual versus predicted values confirm that Tanh and LeakyReLU produce predictions closer to the diagonal line, indicating more accurate fits. Residual plots further demonstrate that Tanh has more uniform residuals with fewer outliers compared to other activation functions.

To ensure robustness, multiple runs are conducted with randomized test sets for Tanh and LeakyReLU. This approach assesses whether these activation functions maintain consistent performance across different data splits. Results confirm that Tanh consistently outperforms LeakyReLU in terms of stability and accuracy, with slightly higher R^2 values and lower MSE/MAE on average. For example, in one run, Tanh achieves an R^2 of 0.930 on the test set compared to 0.926 for LeakyReLU. Additionally, Tanh exhibits tighter clustering of residuals around zero across runs, suggesting better generalizability. Further, the impact of different block selection strategies on deep learning model performance strategies have been examined.

Figure 3 illustrates the different layers selected by each of the four approaches considered. The findings revealed that the most sensitive layer method consistently

outperformed the other strategies across the performance metrics. This method demonstrated the highest and most consistent R^2 values for both training and test sets, while also exhibiting lower and more uniform MSE and MAE values across multiple runs. Notably, the Most Sensitive Layer approach showed the lowest standard deviation in R^2, MSE, and MAE, indicating superior robustness and stability compared to the other methods.

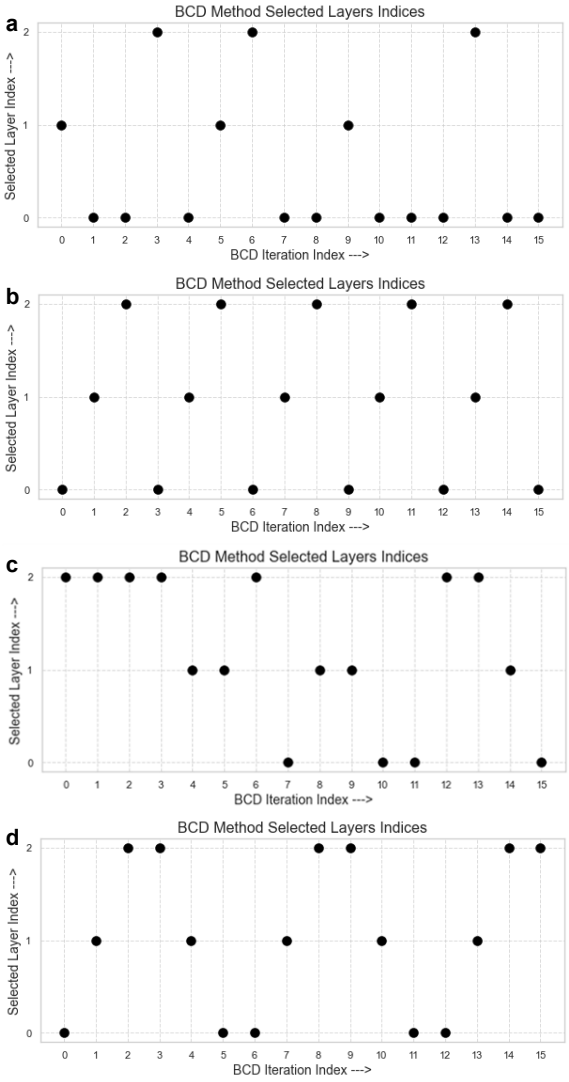

Figure 3. Selection of layers for the BCD using: a) most sensitive layers; b) random layer; c) forward sequentially; and d) ping-pong directionality approach.

The results suggest that the most sensitive layer method is better suited for handling variations in handling input data compared to the other methods, making it the most robust and reliable strategy for optimizing the performance of the deep learning model. To investigate the influence of the four optimization algorithms, and their integration with the BCD, a batch shell method is employed to improve model performance. BCD divides the dataset into mini-batches, optimizing each batch separately, while the Batch Shell Method focuses on layer-by-layer optimization, prioritizing the Most Sensitive Layer in the network.

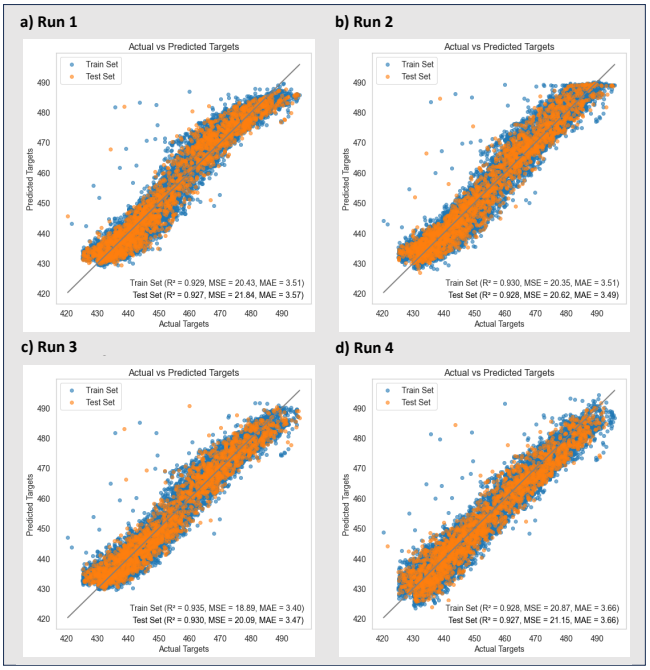

Figure 4. Performance comparison for four runs with ICRS for the CCPP dataset

A randomized test set approach is employed to compare ICRS and L-BFGS across five runs, revealing comparable performance but slight differences in consistency (Figure 4). ICRS achieves test R^2 values ranging from 0.921 to 0.931, with MSE between 19.79 and 23.27 and MAE between 3.46 and 3.60, while L-BFGS shows test R^2 values from 0.914 to 0.931, MSE between 20.01 and 25.53, and MAE between 3.44 and 3.80. Although both optimizers perform well, ICRS demonstrates slightly better stability across runs, particularly in residual distributions.

When combining these optimization methods with BCD and Batch Shell, ADAM provides faster convergence due to its adaptive learning rates but is more sensitive to noisy gradients in smaller datasets or complex models. In contrast, L-BFGS achieves slightly higher accuracy but requires longer training times due to its computational intensity in approximating curvature information. The Batch Shell Method emerges as a promising approach for improving layer-specific optimization by targeting the Most Sensitive Layer, especially when combined with robust optimizers like ICRS or L-BFGS. This integration ensures that the model benefits from both global optimization strategies (e.g., ADAM) and targeted improvements in layer-specific sensitivity (via Batch Shell).

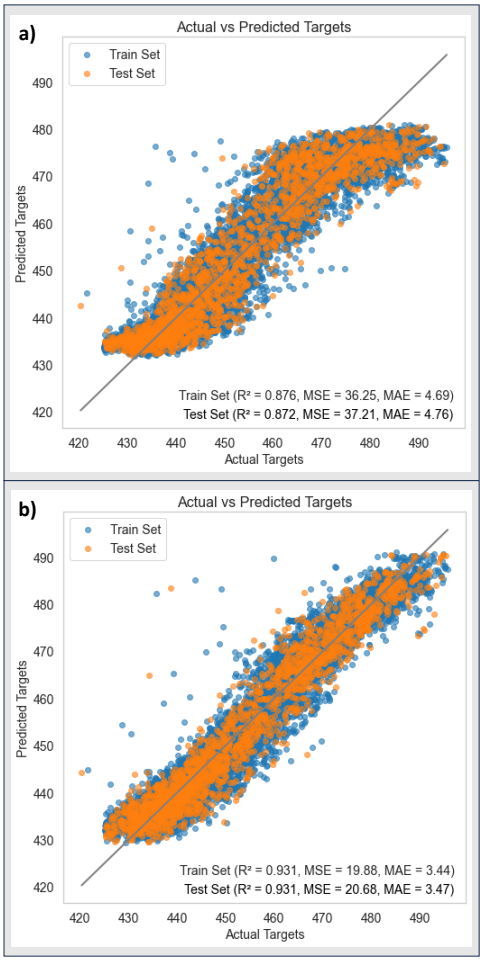

Figure 5. Performance comparison for model training using the CCPP dataset: a) without integration of ICRS with BCD; and b) with integration of ICRS with BCD.

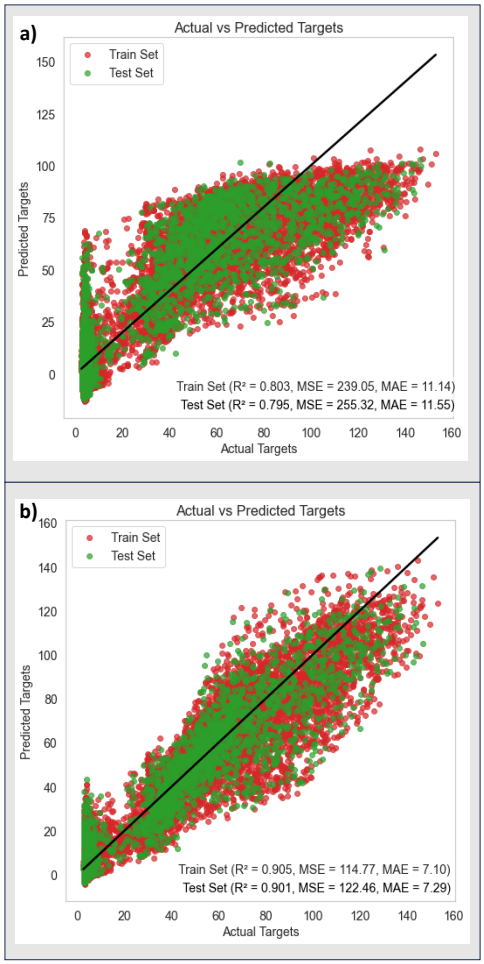

Figure 6. Performance comparison for model training using the SIEC dataset: a) without integration of ICRS with BCD; and b) with integration of ICRS with BCD.

Overall, the study highlights that combining advanced optimization techniques with structured learning strategies can significantly enhance deep learning performance for predictive tasks.

To assess the implementation of the BCD as a key optimization strategy, this sequential procedure to optimize model parameters layer by layer has been integrated with the ICRS to enhance the initialization of parameters and improve convergence in high-dimensional spaces. Using the CCPP dataset, the model trained with BCD and ICRS achieved R^2 values of 0.931 on both training and test sets, demonstrating superior accuracy compared to the model trained without BCD, which yielded R^2 values of 0.876 and 0.872 for training and test sets, respectively.

As shown in Figure 5, the BCD-ICRS combination significantly enhances model convergence and improves training efficiency.

BCD's layer-specific optimization, coupled with the robust parameter initialization provided by ICRS, allows the model to converge to a higher accuracy faster than training without BCD, reducing the computational cost of achieving optimal performance.

The SIEC dataset

The procedure was subsequently applied to the SIEC dataset. Similarly to the results for the CCPP case study, the most sensitive layer approach showed a more uniform distribution of performance metrics compared to the other methods, both in training and testing phases.

Furthermore, the integration of the ICRS optimizer with the BCD method demonstrated strong performance across various block selection strategies, as illustrated in Figure 6. These results demonstrate that the targeted layer optimization provided by the BCD improves the accuracy and robustness of the SIEC model, showcasing the adaptability of this method to the specific characteristics of this dataset. The performance improvement, while specific to the SIEC data, suggests the potential of

BCD for generalizable application in complex system modelling, as also evidenced by the similar benefits observed with the CCPP dataset (Figure 5), albeit with different data characteristics.

Overall, the object-oriented framework proved effective for both datasets by leveraging modularity and advanced optimization techniques. A core contribution of this work is demonstrating the algorithmic advantages of the proposed hybrid approach. In this prototype implementation, training the SIEC dataset with L-BFGS took ~220 minutes, while BCD-ICRS achieved comparable results in just 35 minutes (6x speedup). For the CCPP dataset, L-BFGS required ~35 minutes, whereas ICRS converged in 7 minutes (5x acceleration). These results validate the method's efficiency gains with substantial empirical improvements. While GPU-accelerated frameworks can train similar models in seconds, the focus is on advancing algorithmic efficiency, which remains critical for large, high-dimensional problems, where solver performance is a key bottleneck [8].

CONCLUSIONS

This work introduces a novel hybrid deep learning training framework that addresses key challenges in optimizing complex neural networks. While object-oriented programming is prevalent in modern deep learning solutions, our approach distinguishes itself through the novel integration of BCD and ICRS, allowing for modular experimentation with layer-specific optimization strategies. BCD enables sequential, resource-efficient layer optimization, while ICRS provides robust initial parameters, enhancing convergence in high-dimensional spaces. This combination improves scalability and computational efficiency while offering a customizable environment adaptable to diverse architectures and datasets.

The framework's ability to integrate batch-based processing with adaptive optimization methods underscores its ability to handle large-scale DLNNs. Empirical validation on CCPP and SIEC datasets from the UCI Machine Learning Repository demonstrates significant improvements in training efficiency, accuracy, and robustness over conventional methods, setting a new benchmark for industrial applications. Its modularity allows users to easily adjust network configuration and integrate additional layers without extensive programming, making it highly adaptable for various datasets or applications.

Current work focuses on extending this framework to additional datasets and domains to further validate its scalability and generalizability. Future efforts will implement the hybrid approach in CUDA to fully quantify its practical impact on deep learning workloads, particularly for complex models where second-order optimization remains computationally challenging. Additionally, integrating automated hyperparameter tuning and exploring real-time applications in industrial settings will provide valuable insights into optimizing deep learning models for practical use cases.

Moreover, automatic network architecture configuration will enable the framework to dynamically adjust its architecture, such as the number of layers, neurons per layer, and activation functions, based on the characteristics of the dataset and the optimization goals. Exploring real-time architecture reconfiguration during training could provide valuable insights into dynamic optimization strategies, allowing models to adapt to changing data distributions or computational constraints in industrial applications. While this presents significant implementation challenges, advancements in adaptive learning algorithms and hardware acceleration may make such dynamic reconfiguration feasible in future research.

REFERENCES

1. Manakitsa N et al. A review of machine learning and deep learning for object detection, semantic segmentation, and human action recognition in machine robotic vision. *Technol.* 12:15 (2024)
2. Rumelhart DE et al. Learning representations by backpropagation errors. *Nature* 323:533-536 (1986)
3. Zhang S et al. Hierarchical multi-scale parametric optimization of deep neural networks. *Appl. Intell.* 53:24963-24990 (2023)
4. Liu M et al. An Improved ADAM optimization algorithm combining adaptive coefficients and composite gradients based on randomized block coordinate descent. *Comput. Intell. Neurosci.* 2023:4765891 (2023)
5. Lodhi YR, Shukla SVS. Applying object-oriented principles to machine learning frameworks for improved scalability. *IJSCI* 1:10-17 (2024)
6. Li B et al. ICRS-Filter: A randomized direct search algorithm for constrained nonconvex optimization problems. *CHERD* 106:178-190 (2016)
7. Dua D and Graff C. UCI Machine learning repository. Irvine, CA: University of California, School of Information and Computer Science (2019) Retrieved from: https://archive.ics.uci.edu/ml
8. Byrd RH, Hansen SL, Nocedal J, Singer Y. A stochastic quasi-Newton method for large-scale optimization. *SIAM J Optim.* 2:1008-1031 (2016)