Towards DNN Training at the Edge with Direct Feedback Alignment

Matteo Lai[1,†], Davide Nadalini[2,†] and Francesco Ratto[1,*,†]

[1]Università degli Studi di Cagliari, 09123 Cagliari, Italy
[2]Università di Bologna, 40134 Bologna, Italy

Abstract

Training neural networks at the edge enables self-adaptive and evolving systems through on-device and federated learning. However, memory and computational constraints make such training approaches, often based on backpropagation, challenging, particularly on MicroController Units (MCUs). In this work, we investigate the deployment of on-device training, based on Direct-Feedback Alignment (DFA), a biologically plausible method that replaces weight-symmetric error propagation with fixed random feedback connections. We present a baseline implementation of DFA on Parallel Ultra-Low Power (PULP) MicroController Units (MCUs), based on the PULP-TrainLib framework, analyzing its computational and memory costs for resource-efficient deployment. Experiments on the MNIST dataset demonstrate the feasibility of DFA-based training on resource-constrained devices, achieving competitive accuracy, latency and memory usage compared to standard approaches. We further discuss strategies for latency and memory optimization, including sparse update and buffer reuse, and outline a path toward resource-efficient deployment of DFA-based training on edge ultra-low-power MCUs.

Keywords

Direct Feedback Alignment, On-device learning, MCUs, PULP

1. Introduction

Modern embedded and cyber-physical systems often run Deep Neural Networks (DNNs) to enhance their in-field operability [1]. DNNs are trained on powerful, Graphics Processing Unit (GPU)-equipped servers and subsequently deployed at the edge. While this approach enables accurate models on resource-constrained devices such as MicroController Units (MCUs), the deployed DNNs remain static and can suffer accuracy degradation under domain shifts. Hence, in modern ubiquitous applications, adapting to new local data is crucial, motivating paradigms such as Continual [2] and Federated [3] Learning, which often require resource-demanding backpropagation on-device.

The rising field of On-Device Learning (ODL) has recently advanced with firmwares enabling backpropagation on MCUs [4, 5]. Yet, the high memory and computation costs of backpropagation hinder its widespread adoption on such devices. To mitigate this, "forward-only" training techniques have been proposed, which bypass the backward pass and reduce memory by avoiding activation storage during the forward pass [6, 7, 8]. Among these, Direct-Feedback Alignment (DFA) stands out as a promising and biologically plausible method: instead of requiring symmetric forward and backward weights as in Backpropagation (BP), it propagates error signals through fixed random feedback connections.

In this work, we propose a novel implementation of DFA to enable lightweight training on state-of-the-art Parallel Ultra-Low Power (PULP) MCUs. Our code leverages the optimized linear algebra operators of PULP-TrainLib [4], providing a starting point for floating-point-based DFA on PULP. Although DFA has been successfully applied in federated learning [9, 10], to the best of our knowledge it has not yet been explored for direct, fully on-device training on edge MCUs. To support deployment on such constrained devices, we also provide a computational and memory analysis of DFA for small fully-connected DNNs, demonstrating competitive accuracy with standard BP on MNIST. Finally, we

CPSWS'25: CPS Summer School PhD Workshop, September 22, 2025, Alghero, Sardinia, Italy
*Corresponding author.
†These authors contributed equally.
matteo.lai3@unica.it (M. Lai); d.nadalini@unibo.it (D. Nadalini); francesco.ratto@unica.it (F. Ratto)
0000-0002-8003-7633 (D. Nadalini); 0000-0001-5756-5879 (F. Ratto)

discuss, with reference to prior work, the envisioned latency and memory optimizations toward practical deployment of DFA-based training at the edge.

2. Background

2.1. Direct Feedback Alignment

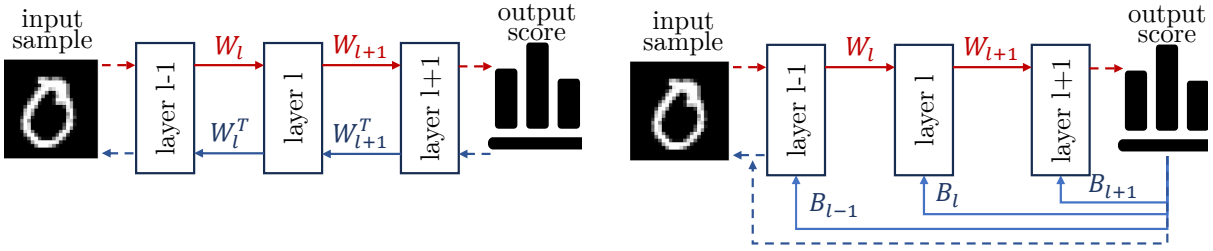
The majority of modern DNNs is trained using the backpropagation algorithm, which iteratively computes the optimization parameters of every DNN layer by propagating the prediction error through every layer. After performing a *forward* step, which computes the DNN prediction with respect to a batch of input data, a loss function L is called to compute the prediction error with respect to the labels associated with the input data. During the forward step, the input activations of every layer are stored for the following steps. Given the gradient of the loss function with respect to the DNN prediction (*error*), the *backward* step computes the optimization parameters (*weight gradients*) of every layer, whose weights are updated by an optimizer (e.g., Stochastic Gradient Descent). We depict BP flow in Fig. 2a. While showing state-of-the-art accuracy, the backpropagation algorithm requires high memory storage (i.e., for DNN activations and gradients) and computation, since the compute cost of the backward step is attested as $\approx 2\times$ the one of the forward.

To reduce such memory and computation demands, Lillicrap et al. [11] demonstrated that DNNs can be effectively trained using fixed random feedback connections. Building on such an approach, Nøkland introduced DFA [12], in which the DNN prediction error is directly broadcast to every layer through fixed random connections, rather than being backpropagated sequentially through each layer. More specifically, given a DNN composed of fully-connected layers, the error δ^l at layer l is directly derived from the error of the entire network δ^{output} through a fixed, random noise matrix B^l of appropriate size:

$$\delta^l = \frac{\partial L}{\partial a^l} \circ \sigma'(z^l) = B^l \cdot \delta^{\text{output}} \circ \sigma'(z^l)$$

where a^l is the input activation at layer l, σ' the derivative of layer's l activation function (e.g., ReLU), and $\circ$ indicates the Hadamard product. Note that, in the case of BP, $\frac{\partial L}{\partial a^l} = W^{l+1} \cdot \delta^{l+1}$, where W^{l+1} are the weights of the following layer, which impose sequential error computation.

Thanks to DFA, all hidden layers receive the error information in parallel, effectively decoupling their weight updates, which are sequentially computed in case of BP, as depicted in Fig. 2b. Moreover, DFA increases the biological plausibility of deep learning models and offers greater architectural flexibility, since learning does not rely on perfectly mirrored feedforward and feedback pathways. DFA has been shown to achieve competitive performance (e.g. on MNIST and CIFAR-10) that approaches the accuracy of conventional BP, validating it as an effective alternative for training neural networks [13].



(a) Training steps with BP.

(b) Training steps with DFA.

Figure 1: Comparison of the BP and DFA training algorithms.

2.2. PULP and PULP-TrainLib

We benchmark our ODL implementation on a System on a Chip (SoC) based on a Parallel Ultra-Low Power architecture. PULP is a scalable edge computing platform built for energy-efficient computation,

based on RISC-V cores. We target a SoC inspired by Vega [14], a 10-core MCU featuring a single core for system control tasks, and a 9-core cluster, designed to accelerate compute-intensive tasks. At the system level, the SoC features 2 MB of L2 memory, accessible by the single core, while the cluster is equipped with 128 kB of L1 memory, accessible in a single cycle by the parallel cores. Data stored in L2 can be accessed by the cluster via a reserved DMA. Every core implements a standard RV32IMFC instruction set, extended with a custom DSP ISA (Xpulp) to reduce overheads in highly uniform workloads, such as matrix multiplications, including post-increment load/store instructions and 2-level hardware loops. Finally, each cluster core is equipped with a mixed-precision Floating-Point Unit, supporting single-cycle DSP instructions, like Fused-Multiply-Accumulate.

We develop our DFA training kernels on top of PULP-TrainLib [4], an ODL software library for PULP SoCs featuring latency- and hardware-optimized training kernels to support BP on ultra-low-power edge devices. We profile our code using GVSoC [15], a behavioural event-based simulator capable of simulating our SoC's latency within 10% error compared to real hardware.

3. Baseline Implementation

We implement the baseline kernels of DFA on a PULP architecture. Furthermore, we analyze the computational and memory costs of a single fully-connected layer (Tab. 1) , comparing them with BP.

Fully connected layer: $a^l = \sigma(z^l) = \sigma(W^l \cdot a^{l-1})$	Forward	Backward/Feedback	Memory
Backpropagation	$W^l \cdot a^{l-1}$	error: $(W^{l+1})^\top \cdot \delta^{l+1}$ W grad: $\delta^{(l)} \cdot (a^{(l-1)})^\top$	$z^l \in \mathbb{R}^B$, $\frac{\partial L}{\partial a^l} \in \mathbb{R}^B$, $W \in \mathbb{R}^{A \times B}$, $\frac{\partial L}{\partial W} \in \mathbb{R}^{A \times B}$
Direct Feedback Alignment	$W^l \cdot a^{l-1}$	error: $B^l \cdot \delta^{out}$ W grad: $\delta^{(l)} \cdot (a^{(l-1)})^\top$	$z^l \in \mathbb{R}^B$, $\frac{\partial L}{\partial a^l} \in \mathbb{R}^B$, $W \in \mathbb{R}^{A \times B}$, $\frac{\partial L}{\partial W} \in \mathbb{R}^{A \times B}$ $B \in \mathbb{R}^{A \times O}$

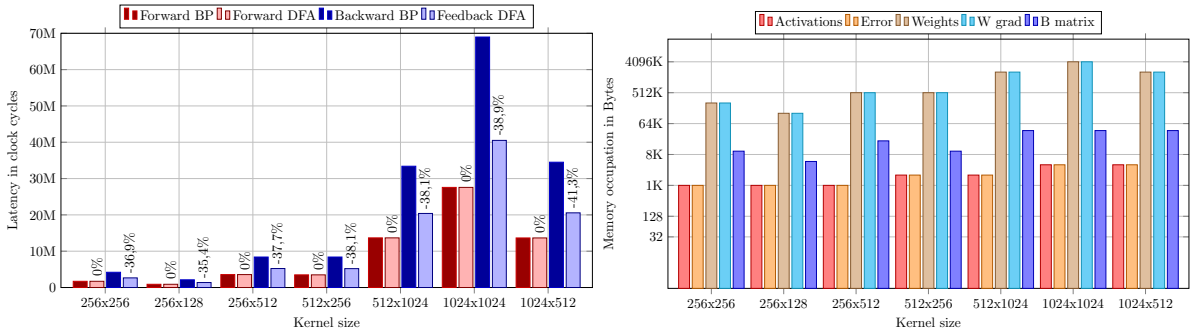
Table 1
Comparison of the main calculations involved in the *Forward* and *Backward/Feedback* passes with BP and DFA on fully connected layer. We assume the kernel W has A rows and B cols, and an output of O classes. The *Memory* columns highlights the tensors that would need to be stored in a straightforward implementation.

The forward pass consists of a matrix-vector multiplication for both methods, while the potential computational advantage of DFA over BP is assessed in the backward step. In such phase, BP computes a dot product between the transposed weight kernel and the propagated error vector, whereas DFA replaces this step with a multiplication of the output error by the noise matrix B. Notably, the computational cost of the feedback pass in DFA depends on the dimensionality of the output layer, i.e., the number of output classes. Given a DNN model with L layers, indeed, the matrix-vector multiplication of layer l, during the feedback pass, features a size of $dim(a^L) \times dim(a^l)$, where $dim(a^l)$ indicates the dimensionality of the activation at layer a^l and a^L the DNN's prediction. In contrast, the BP feedback pass of every layer scales with the size of the succeeding layer's kernel. After the layer error is computed, both algorithms derive the weight gradient with the same vector multiplication. Regarding the memory needed to execute the two training algorithms, DFA introduces an additional memory requirement to store the feedback matrix B, which needs to be persistently stored throughout training.

We implement the training kernels of DFA for fully-connected layers on top of the FP32 primitives of PULP-TrainLib. In particular, we implement the feedback step functions of DFA, while the forward step functions are shared with BP. Our feedback step functions are implemented using PULP-TrainLib's latency-optimized *matrix multiplication* fuction, to exploit loop unrolling and multi-core parallelization over the PULP cluster. This requires creating the data structures and buffers to exploit these functions. For every layer l, the respective B^l matrix is allocated in the *L2* memory of the PULP MCU and initialized with random coefficients, which exported from the trained model the same way as the layers weights. Finally, we reuse PULP-TrainLib's stochastic Gradient Descent (SGD) optimizer.

Fig. 4 reports the results of an experimental assessment of our implementation over single layers

with growing input and output sizes, measuring actual execution times and memory footprint. For DFA, we assume an output error referred to a 10-class problem, like MNIST and CIFAR-10. As both BP and DFA use the same matrix multiplication kernel, we measure the same forward latency. In the feedback/backward phase, however, DFA consistently outperforms BP: the feedback pass is faster since the B matrix is smaller than the transposed weight matrix W^T. This advantage grows with the kernel size, reducing latency from −36% to −41% compared to BP. Nevertheless, the overall speed-up in training is constrained by the weight gradient computation, which is identical for both algorithms, and remains the dominant cost in the backward/feedback stage. The B matrix also slightly increases the total memory footprint, being an order of magnitude smaller than W and its gradient, which has the same size.



(a) Computation latency.

(b) Memory utilization.

Figure 2: Comparison of BP and DFA over single layers in terms of latency and memory needed.

We benchmark the performances of a full DNN by deploying a model featuring two fully connected layers (first layer with 784x36 kernel and ReLU activation, second layer 36x10 with softmax activation for classification) to our simulated PULP platform and evaluated its training performance on the MNIST dataset. We trained this network using Biotorch[1], a PyTorch-based library specializing in biologically plausible learning algorithms that supports DFA among other training algorithms. We trained the model using *Crossentropy* loss function, SGD with optimizer with 10^{-4} *learning rate* for both the DFA and BP. For the DFA we used a *xavier* distribution for noise matrix initialization. Results are shown in Table 2, where the proposed DFA approach is compared with the BP obtained with a PyTorch training and deployment with PULP-TrainLib. The results in Table 2 from this baseline show that: i) classification accuracy at convergence is comparable between BP and DFA, confirming that DFA can be used effectively even in networks trained directly on microcontrollers without degradation in final performance. ii) training latency and memory occupation is also comparable between the two approaches. This is largely due to both methods relying on the same highly optimized matrix multiplication routines, and the relatively small size of the feedback matrices in this shallow network.

Training Method	Accuracy with 15 epochs	Forward Latency 1 Core	Forward Latency 8 Cores	Backward/Feedback Latency 1 Core	Backward/Feedback Latency 8 Cores	Memory occupation
BP	98.2	24.8K cc	3.9K cc	34.5K cc	7.1K cc	242 KB
DFA	97.9	24.6K cc	4.0K cc	34.5K cc	6.9K cc	246 KB
Variation	+0.3%	-0,5%	+1,0%	-0,1%	-2,7%	+2,1%

Table 2
Results over a tiny DNN.

[1]https://github.com/jsalbert/biotorch

4. Envisioned DFA Optimizations

In Section 3, we demonstrated the feasibility of DFA deployment on PULP MCUs. Results on a tiny DNN show only limited advantages of DFA over BP, although these benefits increase with larger layers, as confirmed by our tests. Nonetheless, the envisioned optimizations described in this section, which will be implemented over the proposed baseline DFA, can deliver significant latency and memory reduction compared to BP.

Update Parallelization One of the often-cited advantages of DFA over BP is its inherent parallelism. In BP, the computation of gradients is inherently sequential, as each layer depends on the error signal propagated from the subsequent layer. In contrast, DFA allows each layer to calculate its weight gradient independently, using the same global output error and its own feedback matrix **B**. This characteristic makes DFA an attractive candidate for parallel execution in multiple computingnits. However, PULP-TrainLib backpropagation kernels are able to exploit fine-grained intra-layer parallelism, achieving up to 7.5× speedup when running a single-layer BP over 8 cores. This leaves limited headroom for additional speedup through inter-layer parallelism, as would be enabled by DFA. Since the theoretical maximum speedup for parallelizing DFA across 8 layers (or cores) is 8×, the benefit over the existing implementation is marginal at best. Therefore, we did not pursue a parallel DFA implementation on this platform, as it would offer negligible gains over the already optimized BP kernels. That said, the situation may differ significantly on other microcontroller-class platforms where optimized training libraries are not available, or where compilers and runtime support do not automatically extract parallelism from BP. In such cases, DFA could provide a clean and hardware-friendly strategy for exploiting parallelism in on-device training. By enabling independent, concurrent weight updates across layers, DFA could better utilize idle cores or distributed computing elements, especially in platforms with limited support for shared-memory parallelism.

Unique Noise matrix According to Launay et al. [16], a single global feedback matrix **B** can be shared across all layers in Direct Feedback Alignment (DFA), provided that appropriate slicing is applied to match each layer's shape. Specifically, rather than assigning a separate noise matrix $\mathbf{B}_i$ to each layer, a unique **B** of sufficient size can be statically allocated, and each layer can use a dedicated slice of it for its gradient computation. This approach, depicted in Fig. 3 reduces the total memory required for storing feedback matrices from $\sum_i \text{dimout} \times \text{dimin}, i$ to just the maximum slice needed. On MCUs with tight memory budgets, this optimization is especially beneficial when scaling to deeper networks. This approach is showing to achieve optimal accuracy with fully connected layers, but it is limited with deep convolutional ones.

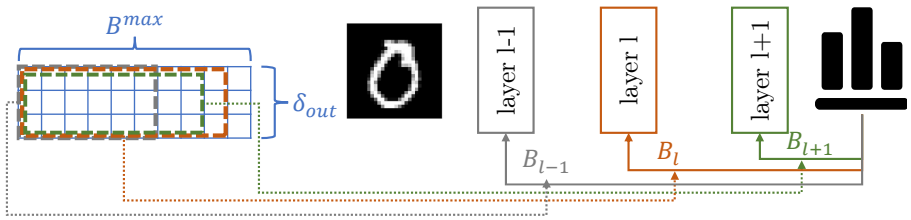
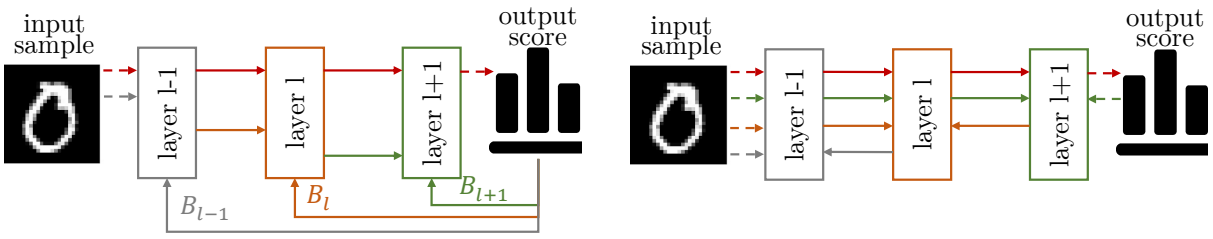


Figure 3: DFA with a shared noise matrix *B*.

Memory buffer sharing for activations In standard backpropagation (BP), each layer's input activations must be stored during the forward pass, since they are later required to compute gradients in the backward pass. This causes the memory footprint to grow with network depth. DFA) relaxes this requirement. Since each weight update depends only on the current layer's input activations and the global output error, layers weights can be updated in any order. As illustrated in Fig. 4a, we

propose reusing a single memory buffer, sized for the largest activation vector in the network. During the forward pass, depicted in red, activations are not stored but simply overwritten layer by layer, with the sole purpose of computing the network output and the global error δ_{out}. Once the error is available, training can proceed layer by layer in forward order. For layer $l-1$, the input activations are recomputed on-the-fly, immediately used to compute the weights gradient. Now activation for layer l can be computed and then the layer $l-1$ updated and its activation discarded. The procedure continues for layer l, then $l+1$, and so forth. In this way, activations are never accumulated in memory, and the buffer can be reused across all layers. Such scheduling would be prohibitive for BP. Because weight updates must start from the last layer and proceed backwards, omitting stored activations would require repeatedly recomputing large portions of the forward pass, as shown in Fig. 4b. Each earlier layer would therefore add considerable overhead. The advantage of DFA becomes more pronounced as the network depth increases. While this approach introduces additional computation due to repeated forward steps, execution time is often a soft constraint in ODL, whereas memory is a hard one to accomodate large networks.



(a) DFA training stepswithout storing activations.

(b) BP training steps without storing activations.

Figure 4: Comparison of the BP and DFA training algorithms implemented without storing activation vectors.

Sparse Weight Update and Layers Freezing In [17], sparse weight update and layer freezing are identified as common ODL techniques to reduce training computation. Sparse weight update uses a sparse feedback matrix to modify only weights that are expected to be more sensitive. This approach has be demonstrated when combined with DFA [18], suggesting a potential path to further reduce computational cost without significantly impacting accuracy. Layer freezing focuses on updating only a subset of layers during training. Besides for ODL, it has been shown to work effectively with BP also in a federated learning context [19]. A layer freezing strategy could be particularly effective with DFA, as weight updates are parallel and independent; thus, one could adjust the number of updated layers according to the available time or energy budget. Further investigation is required to assess the efficacy of this approach in combination with DFA.

5. Conclusion

In this work, we presented a baseline implementation of the DFA algorithm on the PULP platform, which, to the best of our knowledge, represents the first implementation on MCUs. The implementation exploits the optimized kernels available in the PULP Train-Lib, combining it with DFA training available in Biotoch. This approach achieves comparable latency results to standard BP already available in PULP-TrainLib. Moreover, we envision some possible optimization techniques to improve latency or reduce the training memory by exploiting the independence of every layer update provided by DFA.

Acknowledgments

This work was supported by MYRTUS. “MYRTUS is funded by the European Union, by grant No. 101135183. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.”

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: improve grammar, spelling and general readability. After using these tool(s)/service(s), the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Francesca Palumbo et al. "Multi-Partner Project: Key Enabling Technologies for Cognitive Computing Continuum-MYRTUS Project Perspective". In: *2025 DATE Conference*. IEEE. 2025.

[2] Liyuan Wang et al. "A comprehensive survey of continual learning: Theory, method and application". In: *IEEE transactions on pattern analysis and machine intelligence* 46.8 (2024), pp. 5362–5383.

[3] Chen Zhang et al. "A survey on federated learning". In: *Knowledge-Based Systems* 216 (2021), p. 106775.

[4] Davide Nadalini et al. "Pulp-trainlib: Enabling on-device training for risc-v multi-core mcus through performance-driven autotuning". In: *International Conference on Embedded Computer Systems*. Springer. 2022, pp. 200–216.

[5] Lars Wulfert et al. "AIfES: A next-generation edge AI framework". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.6 (2024), pp. 4519–4533.

[6] Xiaohan Zhao et al. "Accelerated On-Device Forward Neural Network Training with Module-Wise Descending Asynchronism". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 52265–52292.

[7] Baichuan Huang et al. "TinyFoA: Memory Efficient Forward-Only Algorithm for On-Device Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 16. 2025.

[8] Danilo Pietro Pau et al. "TinyRCE: Multipurpose forward learning for resource restricted devices". In: *IEEE Sensors Letters* 7.10 (2023), pp. 1–4.

[9] Kijung Jung et al. "LAFD: Local-differentially private and asynchronous federated learning with direct feedback alignment". In: *Ieee Access* 11 (2023), pp. 86754–86769.

[10] Luca Colombo et al. "TIFeD: a Tiny Integer-based Federated learning algorithm with Direct feedback alignment". In: *Proceedings of the Third International Conference on AI-ML Systems*. 2023.

[11] Timothy P Lillicrap et al. "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nature communications* 7.1 (2016), p. 13276.

[12] Arild Nøkland. "Direct feedback alignment provides learning in deep neural networks". In: *Advances in neural information processing systems* 29 (2016).

[13] Albert Jiménez Sanfiz et al. *Benchmarking the Accuracy and Robustness of Feedback Alignment Algorithms*. 2021. arXiv: 2108.13446 [cs.LG].

[14] Davide Rossi et al. "Vega: A ten-core SoC for IoT endnodes with DNN acceleration and cognitive wake-up from MRAM-based state-retentive sleep mode". In: *IEEE Journal of Solid-State Circuits* 57.1 (2021), pp. 127–139.

[15] Nazareno Bruschi et al. "GVSoC: A Highly Configurable, Fast and Accurate Full-Platform Simulator for RISC-V based IoT Processors". In: *2021 IEEE 39th International Conference on Computer Design (ICCD)*. 2021, pp. 409–416.

[16] Julien Launay et al. "Principled training of neural networks with direct feedback alignment". In: *arXiv preprint arXiv:1906.04554* (2019).

[17] Shuai Zhu et al. "On-device training: A first overview on existing systems". In: *ACM transactions on sensor networks* 20.6 (2024), pp. 1–39.

[18] Brian Crafton et al. "Direct feedback alignment with sparse connections for local learning". In: *Frontiers in neuroscience* 13 (2019), p. 525.

[19] Erich Malan et al. "Automatic layer freezing for communication efficiency in cross-device federated learning". In: *IEEE Internet of Things Journal* 11.4 (2023), pp. 6072–6083.