TinyFoA: Memory Efficient Forward-Only Algorithm for On-Device Learning

Baichuan Huang, Amir Aminifar

Department of Electrical and Information Technology, Lund University, Sweden
{baichuan.huang, amir.aminifar}@eit.lth.se

Abstract

Forward-only algorithms offer a promising memory-efficient alternative to Backpropagation (BP) for on-device training. However, state-of-the-art forward-only algorithms, e.g., Forward-Forward (FF), still require a substantial amount of memory during the training process, often exceeding the limits of mobile edge and Internet of Things (IoT) devices. At the same time, existing memory-optimization techniques, e.g., binarizing parameters and activations, are mainly designed for BP, hence significantly degrading the classification performance when applied to state-of-the-art forward-only algorithms. In this paper, we propose a memory-efficient forward-only algorithm called TinyFoA, to reduce dynamic memory overhead in the training process. Our TinyFoA optimizes the memory efficiency not only by layer-wise training but also by partially updating each layer, as well as by binarizing the weights and the activations. We extensively evaluate our proposed TinyFoA against BP and other forward-only algorithms and demonstrate its effectiveness and superiority compared to state-of-the-art forward-only algorithms in terms of classification performance and training memory overhead, reducing the memory overheads by an order of magnitude.

Code — https://github.com/whubaichuan/TinyFoA

1 Introduction

Today, the majority of the state-of-the-art Artificial Neural Networks s (ANNs)/Deep Neural Networks (DNNs) are trained based on the Backpropagation (BP) algorithm. However, BP is known to be computationally, memory, and energy-hungry (Patterson et al. 2021, 2022; Niu et al. 2024; Pau and Aymone 2023), which is often attributed to its biologically implausible nature (Lillicrap et al. 2016). This is particularly relevant in the context of Internet of Things (IoT) and mobile devices, which are limited in terms of resources, namely computing power, memory storage, and battery/energy budget (Shi et al. 2016; Sopic, Aminifar, and Atienza 2018; Sabry et al. 2022). In this work, we focus mainly on memory efficiency, which is also known to be connected to energy/power consumption (Bi, Duan, and Gniady 2010; Bi, Chandrasekharan, and Gniady 2013; Seo 2022).

As alternatives to BP, forward-only algorithms, such as Present the Error to Perturb the Input To modulate Activity

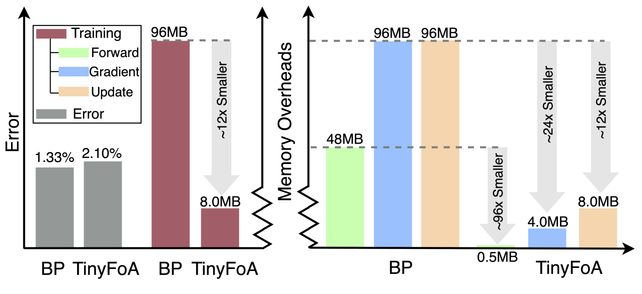


Figure 1: An overview of memory overheads is presented for (a) training between BP and TinyFoA with joint horizontal and vertical layer-wise training and binary weights; (b) sequential training steps, including forward pass, gradient calculation, and parameter update.

(PEPITA) (Dellaferrera et al. 2022) and Forward-Forward (FF) (Hinton 2022), are proposed (Srinivasan et al. 2023) to replace the backward pass of BP with only forward passes. These forward-only algorithms represent the next generation of training schemes inspired by the learning process in the cortex, often adopting the local learning principle without the need to retain the entire neural network in the memory, hence potentially more memory efficient than BP. However, these state-of-the-art forward-only algorithms still impose substantial training memory overhead, generally beyond the limits of mobile edge and IoT devices.

In this paper, we propose a memory-efficient forward-only algorithm, called TinyFoA, which reduces the dynamic training memory overheads compared to BP, as shown in Fig. 1 (a). Fig. 1 (b) shows the breakdown of the memory overheads into the forward pass, gradient calculation, and parameter update. Here, BP consumes ∼48 Megabyte (MB) in forward pass while requiring ∼96 MB in gradient calculation and parameter update. Due to end-to-end training, the error in BP propagates across the entire network (Srinivasan et al. 2023) and *all* layers update their parameters. Consequently, BP needs to store *all* the layers in the aforementioned three sequential training steps, as shown in Fig. 2 (a).

In contrast, our TinyFoA is built on joint horizontal and vertical layer-wise training, where it focuses on training only parts of a single layer. As such, TinyFoA only requires storing

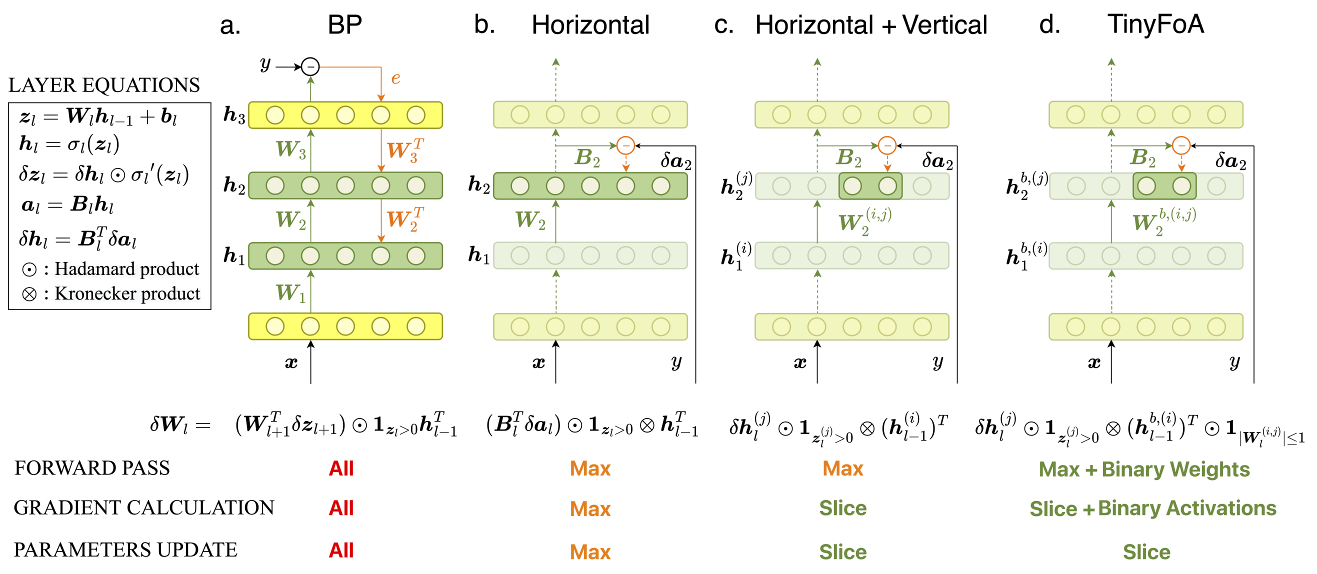

a. BP
b. Horizontal
c. Horizontal + Vertical
d. TinyFoA
LAYER EQUATIONS
$z_l = W_l h_{l-1} + b_l$
$h_l = \sigma_l(z_l)$
$\delta z_l = \delta h_l \odot \sigma_l'(z_l)$
$a_l = B_l h_l$
$\delta h_l = B_l^T \delta a_l$
$\odot$: Hadamard product
$\otimes$: Kronecker product
y
e
h_3
W_3
W_3^T
h_2
W_2
W_2^T
h_1
W_1
x
B_2
δa_2
$h_2^{(j)}$
$W_2^{(i,j)}$
$h_1^{(i)}$
$h_2^{b,(j)}$
$W_2^{b,(i,j)}$
$h_1^{b,(i)}$
$\delta W_l =$
$(W_{l+1}^T \delta z_{l+1}) \odot \mathbf{1}_{z_l>0} h_{l-1}^T$
$(B_l^T \delta a_l) \odot \mathbf{1}_{z_l>0} \otimes h_{l-1}^T$
$\delta h_l^{(j)} \odot \mathbf{1}_{z_l^{(j)}>0} \otimes (h_{l-1}^{(i)})^T$
$\delta h_l^{(j)} \odot \mathbf{1}_{z_l^{(j)}>0} \otimes (h_{l-1}^{b,(i)})^T \odot \mathbf{1}_{|W_l^{(i,j)}|\leq 1}$
FORWARD PASS
GRADIENT CALCULATION
PARAMETERS UPDATE
All
Max
Max
Max + Binary Weights
All
Max
Slice
Slice + Binary Activations
All
Max
Slice
Slice

Figure 2: TinyFoA contains horizontal and vertical layer-wise training with binary weights and binary activations.

a single layer at any given time since only one layer updates its parameters per iteration. Therefore, in the worst case, the memory required depends on the layer with the *maximum* size. By updating only a portion of the full-precision weights, i.e., *slice*, with their corresponding gradients, the training memory overhead is even further reduced.[1]

To further reduce dynamic training memory overheads, we propose to binarize the weights and activations. Binarizing the weights reduces training memory overheads by $\sim$32 times and the computation by $\sim$2 times in forward pass; Furthermore, binarizing activations can accelerate the forward pass by up to $\sim$64 times, leveraging 1-bit XNOR operations combined with pop-count operations (Rastegari et al. 2016).

We compare TinyFoA against BP and three other state-of-the-art approaches, namely, Direct Random Target Projection (DRTP) (Frenkel, Lefebvre, and Bol 2021), PEPITA (Dellaferrera et al. 2022), and FF (Hinton 2022). We evaluate our proposed approach on MNIST (LeCun 1998), CIFAR-10 (Krizhevsky 2009), and CIFAR-100 (Krizhevsky 2009). In addition, we extend our evaluation to a real-world medical IoT application on cardiac arrhythmia classification based on the MIT-BIH Arrhythmia Electrocardiogram (ECG) dataset (Mark et al. 1982), where memory overheads are a major constraint for on-device learning (Rahmani et al. 2018; Sabry et al. 2022; Baghersalimi et al. 2023).

Our main contributions can be summarized as follows:

- We propose TinyFoA, a memory-efficient forward-only algorithm built on joint horizontal and vertical layer-wise training with binarization to reduce the dynamic training memory overheads compared to end-to-end training BP.
- We extensively evaluate our proposed TinyFoA against BP and demonstrate that TinyFoA with joint horizontal and vertical layer-wise training and binary weights reduces the training memory overheads in the forward pass by $\sim$96$\times$, the training memory overheads in gradient calculation by $\sim$24$\times$, and the training memory overheads in parameter update by $\sim$12$\times$, while maintaining a comparable classification performance to the standard BP training algorithm, as shown in Fig. 1. Additionally, TinyFoA outperforms state-of-the-art forward-only algorithms including DRTP, PEPITA, and FF.

[1]Updating parameters utilize full-precision parameters and gradient, as maintaining precision weights during the update is essential for Stochastic Gradient Descent (SGD) to function effectively (Courbariaux, Bengio, and David 2015).

2 TinyFoA

TinyFoA is a backpropagation-free algorithm built on joint horizontal and vertical layer-wise training with binary weights and binary activations, to reduce the dynamic training memory overheads, as shown in Fig. 2.

2.1 Joint Horizontal and Vertical Training

Let us consider a DNN with L layers. The input $\boldsymbol{x} \in \mathbb{R}^{D_x \times 1}$ and the target $\boldsymbol{y} \in \mathbb{R}^{N_c \times 1}$ are considered for training the DNN, where D_x is the size of input $\boldsymbol{x}$ and N_c is the number of classes. The activations of the hidden layer l of the DNN are denoted as $\boldsymbol{h}_l$, where $\boldsymbol{h}_0 = \boldsymbol{x}$.

We propose joint horizontal and vertical layer-wise training, where we train only parts of a single layer, i.e., *slice*. In the forward pass, for the hidden layer l, the activations $\boldsymbol{h}_l \in \mathbb{R}^{D_l \times 1}$ is based on $\boldsymbol{h}_{l-1} \in \mathbb{R}^{D_{l-1} \times 1}$, weights $\boldsymbol{W}_l \in \mathbb{R}^{D_l \times D_{l-1}}$ and biases $\boldsymbol{b}_l \in \mathbb{R}^{D_l \times 1}$, where D_l is the size of output of layer l and D_{l-1} is the size of input to layer l. For each $\boldsymbol{h}_l$, an auxiliary classifier with the fixed random matrix $\boldsymbol{B}_l \in \mathbb{R}^{N_c \times D_l}$ is employed to project the $\boldsymbol{h}_l \in \mathbb{R}^{D_l \times 1}$ to the output logits $\boldsymbol{a}_l \in \mathbb{R}^{N_c \times 1}$, i.e., $\boldsymbol{a}_l = \boldsymbol{B}_l \boldsymbol{h}_l$. Next, the logits $\boldsymbol{a}_l$ are used to calculate the local cross-entropy loss and its gradient $\delta \boldsymbol{a}_l \in \mathbb{R}^{N_c \times 1}$.

In gradient calculation, we divide the input activation $\boldsymbol{h}_{l-1} \in \mathbb{R}^{D_{l-1} \times 1}$ and the output activation $\boldsymbol{h}_l \in \mathbb{R}^{D_l \times 1}$

into M slices, respectively. The sliced input activations are denoted as $\boldsymbol{h}_{l-1}^{(i)} \in \mathbb{R}^{\frac{D_{l-1}}{M}\times 1}$ and the sliced output activations are denoted as $\boldsymbol{h}_l^{(j)} \in \mathbb{R}^{\frac{D_l}{M}\times 1}$, where $i = 1, 2, ..., M$ and $j = 1, 2, ..., M$. As such, the weights are divided into M^2 slices, namely, $\boldsymbol{W}_l^{(i,j)} \in \mathbb{R}^{\frac{D_l \times D_{l-1}}{M^2}}$. Only the partial gradients $\delta\boldsymbol{W}_l^{(i,j)}$ are calculated, to reduce the training memory overheads in gradient calculation. Similarly, in parameter update, only the partial slice of weights is trained iteratively, to reduce the training memory by a factor of M^2.[2]

TinyFoA, with joint horizontal and vertical layer-wise training, is formulated as follows:

$$\text{Forward:} \qquad \boldsymbol{h}_l = \sigma_l(\boldsymbol{z}_l) = \sigma_l(\boldsymbol{W}_l\boldsymbol{h}_{l-1} + \boldsymbol{b}_l), \quad (1)$$

$$\text{Gradient:} \quad \delta\boldsymbol{W}_l^{(i,j)} = \delta\boldsymbol{h}_l^{(j)} \odot \mathbf{1}_{\boldsymbol{z}_l^{(j)}>0} \otimes (\boldsymbol{h}_{l-1}^{(i)})^T, \quad (2)$$

$$\text{Update:} \qquad \boldsymbol{W}_l^{(i,j)} = \boldsymbol{W}_l^{(i,j)} - \eta\delta\boldsymbol{W}_l^{(i,j)}, \quad (3)$$

where $\boldsymbol{z}_l \in \mathbb{R}^{D_l\times 1}$ is logits and σ_l is the activation function for the layer l. $\delta\boldsymbol{h}_l^{(j)} = (\boldsymbol{B}_l^{(j)})^T\delta\boldsymbol{a}_l = (\boldsymbol{B}_l^{(j)})^T(\boldsymbol{p}_l - \boldsymbol{y})$, where $\boldsymbol{B}_l^{(j)} \in \mathbb{R}^{N_c\times\frac{D_l}{M}}$ and $\boldsymbol{p}_l \in \mathbb{R}^{N_c\times 1}$ is the probability distribution vector of $\boldsymbol{a}_l$ by Softmax activation function. The derivative $\mathbf{1}_{\boldsymbol{z}_l^{(j)}>0}$ is the gradient of $\sigma_l(\boldsymbol{z}_l^{(j)})$, meaning that if $\boldsymbol{z}_l^{(j)} > 0$ then output 1; otherwise, output 0. The operator $\odot$ is the Hadamard product and $\otimes$ is the Kronecker Product. Finally, η is the learning rate.

2.2 Binary Network

To reduce the training memory even further, we continue to binarize the weights. For the hidden layer l, the activations $\boldsymbol{h}_l \in \mathbb{R}^{D_l\times 1}$ is based on binary weights $\boldsymbol{W}_l^b \in \mathbb{R}^{D_l\times D_{l-1}}$. The binary forward pass is as follows:

$$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l^b\boldsymbol{h}_{l-1} + \boldsymbol{b}_l). \quad (4)$$

In the architecture of TinyFoA, e.g., Fully Connected Network (FC) and LC, the weights, require substantially more memory than the intermediate activations. Therefore, the binary forward pass reduces the training memory overheads by $\sim$32 times in the forward pass. We obtain the binarized weights as follows: $\boldsymbol{W}_l^b = \frac{1}{n}\sum_{i=1}^{n}|\boldsymbol{W}_l^i|\cdot\mathrm{Sign}(\boldsymbol{W}_l)$ (Rastegari et al. 2016). Notably, when the weights are binarized, the related computation can be estimated by only addition and subtraction (without multiplication) (Rastegari et al. 2016), resulting in $\sim$2 times speed up because multipliers are the most space and power-hungry components of the digital implementation of neural networks (Courbariaux, Bengio, and David 2015). Moreover, the activation is also binarized in a similar fashion. This allows us to perform the computation with 1-bit XNOR operations combined with pop-count operations, achieving 64$\times$ less computational overheads.

To obtain the gradient $\delta\boldsymbol{W}_l^{(i,j)}$, we further modify the gradient $\delta\boldsymbol{W}_l^{b,(i,j)}$, by preserving the gradient information and canceling the gradient when $\boldsymbol{W}_l^{(i,j)}$ is too large to avoid performance degradation (Hubara et al. 2016). The gradient $\delta\boldsymbol{W}_l^{(i,j)}$ is formulated as follows:

$$\begin{aligned}\delta\boldsymbol{W}_l^{(i,j)} &= \delta\boldsymbol{W}_l^{b,(i,j)} \odot \mathbf{1}_{|\boldsymbol{W}_l^{(i,j)}|\le 1} \quad (5)\\ &= \delta\boldsymbol{h}_l^{(j)} \odot \mathbf{1}_{\boldsymbol{z}_l^{(j)}>0} \otimes (\boldsymbol{h}_{l-1}^{b,(i)})^T \odot \mathbf{1}_{|\boldsymbol{W}_l^{(i,j)}|\le 1},\end{aligned}$$

where $\delta\boldsymbol{h}_l^{(j)} = (\boldsymbol{B}_l^{(j)})^T(\boldsymbol{p}_l - \boldsymbol{y}) \odot \mathbf{1}_{|\boldsymbol{h}_l^{(j)}|\le 1}$; $\mathbf{1}_{|\boldsymbol{h}_l^{(j)}|\le 1} = 1$ if $|\boldsymbol{h}_l^{(j)}| \le 1$ and $\mathbf{1}_{|\boldsymbol{h}_l^{(j)}|\le 1} = 0$ otherwise. Moreover, $\mathbf{1}_{|\boldsymbol{W}_l^{(i,j)}|\le 1}$ outputs 1 if $|\boldsymbol{W}_l^{(i,j)}| \le 1$; otherwise $\mathbf{1}_{|\boldsymbol{W}_l^{(i,j)}|\le 1}$ outputs 0. Note that only $\boldsymbol{W}_l^{(i,j)}$ and $\boldsymbol{b}_l^{(j)}$ are trainable.

2.3 Motivation and Biological Plausibility

TinyFoA focuses primarily on FC and LC (i.e., Convolutional Neural Network (CNN) without weight sharing), since weight sharing in CNN is not biologically plausible (Pogodin et al. 2021). In the biologically plausible architecture, e.g., FC and LC, the parameters, partially the weights, consume substantially more memory than the intermediate activations. This finding contrasts with CNN-based networks trained with BP (such as MobileNetV2 (Sandler et al. 2018) and ResNet-50 (He et al. 2016)), where activation memory is the primary bottleneck (Cai et al. 2020).

To address this issue, we propose TinyFoA, built on joint horizontal and vertical layer-wise training, where it focuses on training only parts of a single layer, without weight sharing at the architecture level. Although memory-reducing binarizing techniques incorporated in DNN training exist in the literature (Courbariaux, Bengio, and David 2015; Rastegari et al. 2016; Hubara et al. 2016), these techniques are mainly designed for the BP training scheme and have not been verified on forward-only algorithms—binarizing activations result in greater loss of classification performance (as presented in Section 4) and significantly higher error (as illustrated in Fig. 3) for PEPITA and FF compared to TinyFoA.

TinyFoA considers the biologically implausible issues of weight transport (Burbank and Kreiman 2012) in the standard pass, non-locality (Whittington and Bogacz 2019), update locking (Jaderberg et al. 2017), and frozen activities (Liao, Leibo, and Poggio 2016) that BP, PEPITA, and FF have not completely solved. TinyFoA avoids weight symmetry in the standard pass by the fixed random matrix in horizontal layer-wise training. TinyFoA performs the training strictly locally by introducing an auxiliary classifier in horizontal layer-wise training, without the need for the information from the top to the bottom layers. Additionally, TinyFoA does not suffer from the issues of update locking and frozen activities because the weight update in TinyFoA does not need to wait for all the dependent layers in the forward pass to complete execution, without storing the intermediate activations. This is not only among the layers but also within one single layer, thanks to the vertical layer-wise training, which is a unique property among the forward-only learning techniques. Concretely, TinyFoA only trains parts of a single layer based on $\delta\boldsymbol{h}_l^{(j)}$, which is directly mapped from $\delta\boldsymbol{a}_l$ by $\boldsymbol{B}_l^{(j)}$ without

[2]For Locally Connected Network (LC), the number of kernel is divided into M slices, incurring M times the training memory overheads saved.

Setting	Benchmark	MNIST	CIFAR-10	CIFAR-100	MIT-BIH	Forward	Gradient	Update	Comp
Baseline	DRTP	4.50±0.03	52.01±0.13	79.86±0.13	12.99±0.18	48.06	48.03	96.06	96.0
	PEPITA	5.02±0.06	52.09±0.09	78.91±0.18	19.57±0.18	32.04	16.02	32.01	96.0†
	FF	1.46±0.07	47.38±0.25	85.76±0.18	10.86±0.25	16.04	16.04	32.01	96.0*
	TinyFoA-V-BA-BW	1.62±0.08	45.12±0.12	74.57±0.51	9.77±0.49	**16.02**	**16.02**	**32.01**	96.0
	BP	1.33±0.04	43.62±0.33	72.22±0.43	8.25±0.46	48.06	96.06	96.06	96.0
+ Binary Weights	DRTP+BW	6.43±0.09	53.23±0.14	85.72±0.41	24.02±0.26	1.56	48.03	96.06	48.0
	PEPITA+BW	6.39±0.15	53.86±0.16	80.31±0.06	24.69±0.29	1.04	16.02	32.01	48.0†
	FF+BW	3.13±0.05	53.47±0.14	93.13±0.19	12.11±0.48	0.54	16.04	32.01	48.0*
	TinyFoA-V-BA	1.84±0.02	46.07±0.24	74.33±0.17	9.88±0.86	**0.52**	**16.02**	**32.01**	48.0
	BP+BW	1.45±0.03	44.66±0.31	71.88±0.06	8.34±0.11	1.56	49.56	96.06	48.0
+ Binary Activations	DRTP+BW+BA	39.49±7.60	69.61±1.19	97.06±0.30	43.22±7.25	1.53	48.00	96.06	1.5
	PEPITA+BW+BA	89.94±0.24	90.00±0.00	99.00±0.00	32.14±0.92	1.02	16.00	32.01	1.5†
	FF+BW+BA	90.47±0.48	90.00±0.00	98.97±0.05	70.17±1.38	0.51	16.00	32.01	1.5*
	TinyFoA-V	2.06±0.03	50.32±0.24	81.30±0.42	12.01±0.26	**0.50**	**16.00**	**32.01**	1.5
	BP+BW+BA	3.11±0.03	54.44±1.32	83.02±0.51	11.63±0.02	1.53	49.53	96.06	1.5
+ Vertical Training	**TinyFoA**	2.10±0.09	50.38±0.17	80.08±0.15	11.69±0.69	**0.50**	**4.01**	**8.00**	1.5
	BP+BW+BA+V	2.95±0.14	51.32±0.07	80.79±0.16	11.49±0.34	1.53	12.41	24.03	1.5

Table 1: Error (%) of classification performance and the dynamic training memory overheads as well as computation (Comp). The unit for memory and computation is MB. Forward represents the forward pass; Gradient represents gradient calculation; Update represents parameter update. H and V stand for horizontal and vertical layer-wise training, respectively. BW is binary weights and BA is binary activations. † means the normalized computation for equal comparison, i.e., considering a four-hidden-layer network for PEPITA computation. * means the computation of one pass but FF (multi-pass) needs N_c times passes.

locking, hence without the need to utilize all the related activations, instead, only the partial activations, for instance, only $\boldsymbol{h}_l^{(j)}$ and $\boldsymbol{h}_{l-1}^{(i)}$ from Equation (2).

3 Experimental Setup

To evaluate our proposed TinyFoA, we use several datasets: the MNIST dataset of handwritten digits (LeCun 1998), the CIFAR-10 dataset of object recognition (Krizhevsky 2009), and the CIFAR-100 dataset of object recognition (Krizhevsky 2009). Additionally, we extend our evaluation to a real-world medical IoT application, where memory overheads are a major constraint for on-device learning (Huang, Abtahi, and Aminifar 2023; Aminifar et al. 2024; Huang, Abtahi, and Aminifar 2024), specifically, cardiac arrhythmia classification based on the MIT-BIH Arrhythmia ECG Dataset (Mark et al. 1982).

For the dynamic training memory in Random-Access Memory (RAM), we focus on layer activations, DNN parameters (weights and biases), and their corresponding gradients, excluding scratch buffers to maintain a general analysis, because scratch buffer usage varies with the low-level implementation of the workload (Pau and Aymone 2023). Additionally, since PyTorch does not support explicit fine-grained memory management, we rely on the theoretically calculated training memory overheads for comparison in our experiments similar to (Cai et al. 2020). Moreover, our analysis emphasizes fine-grained individual training memory for forward pass, gradient calculation, and parameter update, rather than the peak RAM usage, for dynamically managed RAM usage. For measuring computation, we consider the metric–#Floating Point Operations (FLOPs)$\times$Bytes–in the inference phase.

Our experiments utilize balanced datasets, with classification performance assessed using error, i.e., the total number of incorrectly classified inputs divided by the total number of inputs. The mean error with the standard deviation is averaged over three independent runs. A three-hidden-layer network is considered for PEPIAT (because PEPITA with deeper networks exhibits a decrease in performance (Pau and Aymone 2023; Srinivasan et al. 2023)) and a four-hidden-layer for other algorithms. All these algorithms undergo training on a server equipped with 2 $\times$ 16-core Intel (R) Xeon (R) Gold 6226R (Skylake) Central Processing Units (CPUs) and 1 NVIDIA Tesla T4 Graphics Processing Card (GPU).

4 Results

In this section, we evaluate TinyFoA in terms of classification performance and dynamic training memory overheads, compared to BP (Rumelhart, Hinton, and Williams 1986), DRTP (Frenkel, Lefebvre, and Bol 2021), PEPITA (Dellaferrera et al. 2022), and FF (Hinton 2022).

4.1 Fully Connected Without Weight Sharing

In this section, we extensively evaluate TinyFoA and compare TinyFoA with BP and other forward-only algorithms in the sequential forward pass, gradient calculation, and parameter update. Table 1 reports the classification performance in error metric, the dynamic training memory overheads, and the computation in inference of TinyFoA, compared to DRTP, PEPITA, FF, and BP.

First, in the baseline setting, we compare TinyFoA-V-BA-BW (i.e., TinyFoA without the vertical training, without binary activations, and without binary weights) with DRTP, PEPITA, FF, and BP. DRTP has substantially higher dynamic training memory overheads and a considerably higher error compared to TinyFoA-V-BA-BW. PEPITA achieves comparable training memory overheads in gradient calculation and parameter update compared to TinyFoA-V-BA-BW, but it shows degradation in classification performance and higher training memory overheads in the forward pass. Similarly, FF attains comparable dynamic training memory overheads to TinyFoA-V-BA-BW, but also has a higher error compared to TinyFoA-V-BA-BW, except for MNIST. Finally, BP achieves the lowest error among the baseline algorithms in four datasets; however, BP occupies $\sim$3 times, $\sim$6 times, and $\sim$3 times higher memory than TinyFoA-V-BA-BW in the forward pass, gradient calculation, and parameter update, respectively.

Second, let us consider binary weights for TinyFoA (i.e., TinyFoA-V-BA). We also binarize the weights for BP, and for other forward-only algorithms, i.e., DRTP+BW, PEPITA+BW, FF+BW. All of these algorithms reduce the dynamic training memory by $\sim$32 times in the forward pass; BP+BW also reduces the dynamic training memory by $\sim$1.9 times in the step of gradient calculation, because the gradient in BP is based on error backpropagation by the symmetric weights. DRTP+BW has substantially higher dynamic training memory overheads and a considerably higher error compared to TinyFoA-V-BA. FF+BW has comparable dynamic training memory overheads but higher error compared to TinyFoA-V-BA. PEPITA+BW also shows a degradation in classification performance compared to TinyFoA-V-BA. BP+BW achieves the lowest error among these algorithms in four datasets; however, BP+BW still occupies $\sim$3 times, $\sim$3 times, and $\sim$3 times higher memory than TinyFoA-V-BA in the sequential training steps.

Third, we continue to binarize the activations, i.e., TinyFoA-V. We also binarize the weights and activations for BP, and for other forward-only algorithms, i.e., DRTP+BW+BA, PEPITA+BW+BA, FF+BW+BA. Because of the FC architectures, i.e., where weights occupy greatly more memory overheads than the corresponding activations, the dynamic training memory overheads will almost be kept but with a $\sim$64 times computation speed up, because the computation can now be performed using 1-bit XNOR operations combined with pop-count operations (Rastegari et al. 2016). The computation in inference for all algorithms is reduced by $\sim$64 times. Moreover, with binary weights and activations simultaneously, all of these algorithms experience degradation in classification performance; the forward-only algorithms, such as DRTP+BW+BA, PEPITA+BW+BA, and FF+BW+BA, have a significantly worse degradation when compared to TinyFoA-V. Taking MNIST as an example, when we consider binary activations, the error of DRTP+BW+BA increases from 6.43% to 39.49%; the error of PEPITA+BW+BA increases from 6.39% to 89.94%; the error of FF+BW+BA increases from 3.13% to 90.47%. On the contrary, TinyFoA only experiences a slight increase in the error, i.e., from 1.84% to 2.06%, while BP+BW+BA achieves an error of 3.11%. Overall, TinyFoA-V stands out among the state-of-the-art forward-only algorithms when binarizing both weights and activations.

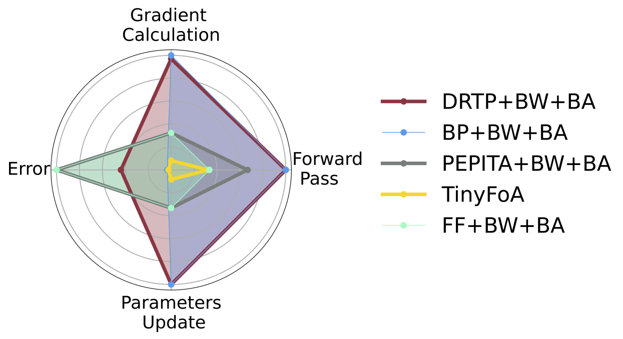


Figure 3: The error and training memory comparison between TinyFoA and BP+BW+BA, DRTP+BW+BA, PEPITA+BW+BA, and FF+BW+BA on MNIST. TinyFoA occupies the smallest area.

Finally, we further consider the vertical layer-wise training in TinyFoA and introduce it also to BP (i.e., BP+BW+BA+V) and conduct the comparison, where the default slice M is set to 2. The dynamic training memory of gradient calculation and update parameters is reduced by $\sim$4 times for TinyFoA and BP+BW+BA+V. Moreover, TinyFoA not only demonstrates less training memory overheads in the sequential training steps but also achieves on par performance with BP+BW+BA+V on all four datasets.

Fig. 3 summarizes the results and shows that TinyFoA occupies the smallest area compared to BP and other forward-only algorithms on MNIST. These results demonstrate the effectiveness of TinyFoA among the state-of-the-art forward-only algorithms and BP.

4.2 Ablation Study

In this section, we first investigate the individual effect of our proposed schemes including horizontal and vertical layer-wise training, binary weights, and binary activations. Then, we extend our proposed TinyFoA to LC architectures and compare TinyFoA against BP. Next, we investigate the effects of different slice sizes M for on-device learning and the effect of reducing the number of exchanging data in horizontal and vertical layer-wise training.

Individual Schemes We present the error with individual schemes for DRTP, PEPITA, FF, TinyFoA, and BP in FC, as shown in Table 2. When compared to the baseline from Table 1, considering only binary weights (e.g., TinyFoA-V-BA) results in a higher error for all the algorithms on MNIST, CIFAR-10, and MIT-BIH datasets, and reduces the dynamic training memory by $\sim$32 times in the forward pass. Moreover, considering only binary activations (TinyFoA-V-BW) leads to a higher error for all the algorithms on four datasets, but reduces the computation in inference by a factor of 2. Finally, considering only vertical layer-wise training (TinyFoA-BW-BA) results in a comparable error to the BP (i.e., BP+V) and reduces the dynamic training memory by $\sim$4 times in gradient calculation and parameter update compared to the baseline.

Setting	Benchmark	MNIST	CIFAR-10	CIFAR-100	MIT-BIH	Forward	Gradient	Update	Comp
Binary Weights	DRTP+BW	6.43$\pm$0.09	53.23$\pm$0.14	85.72$\pm$0.41	24.02$\pm$0.26	1.56	48.03	96.06	48.0
	PEPITA+BW	6.39$\pm$0.15	53.86$\pm$0.16	80.31$\pm$0.06	24.69$\pm$0.29	1.04	16.02	32.01	48.0†
	FF+BW	3.13$\pm$0.05	53.47$\pm$0.14	93.13$\pm$0.19	12.11$\pm$0.48	0.54	16.04	32.01	48.0*
	TinyFoA-V-BA	1.84$\pm$0.02	46.07$\pm$0.24	74.33$\pm$0.17	9.88$\pm$0.86	**0.52**	**16.02**	**32.01**	48.0
	BP+BW	1.45$\pm$0.03	44.66$\pm$0.31	71.88$\pm$0.06	8.34$\pm$0.11	1.56	49.56	96.06	48.0
Binary Activations	DRTP+BA	32.52$\pm$5.15	66.51$\pm$0.28	94.81$\pm$0.37	40.82$\pm$7.18	48.03	48.00	96.06	48.0
	PEPITA+BA	89.99$\pm$0.13	90.33$\pm$0.46	99.00$\pm$0.00	79.94$\pm$0.13	32.02	16.00	32.01	48.0†
	FF+BA	90.36$\pm$0.62	76.20$\pm$1.43	98.05$\pm$0.09	75.33$\pm$4.29	16.01	16.00	32.01	48.0*
	TinyFoA-V-BW	1.83$\pm$0.05	50.36$\pm$0.21	81.20$\pm$0.07	11.47$\pm$0.37	**16.01**	**16.00**	**32.01**	48.0
	BP+BA	5.22$\pm$0.58	75.73$\pm$3.87	97.57$\pm$0.29	62.25$\pm$13.60	48.03	96.03	96.06	48.0
Vertical Training	**TinyFoA-BW-BA**	1.64$\pm$0.06	44.77$\pm$0.06	73.83$\pm$0.45	9.86$\pm$0.42	**16.02**	**4.02**	**8.00**	96.0
	BP+V	1.33$\pm$0.06	43.21$\pm$0.17	71.17$\pm$0.42	8.62$\pm$0.38	48.06	24.06	24.03	96.0

Table 2: Error (%) of individual schemes on the baseline and the dynamic training memory overheads as well as computation.

	MNIST	Forward	Gradient	Update
TinyFoA-V-BA-BW	1.35$\pm$0.03	2.21	2.21	3.61
TinyFoA-V-BA	1.37$\pm$0.05	0.46	2.21	3.61
TinyFoA-V	2.46$\pm$0.11	0.07	1.82	3.61
TinyFoA	2.86$\pm$0.10	0.07	0.92	1.81
BP+BW+BA+V	2.59$\pm$0.11	0.17	2.66	4.97
BP+BW+BA	2.43$\pm$0.20	0.17	5.14	9.93
BP+BW	1.37$\pm$0.06	0.71	5.67	9.93
BP	1.35$\pm$0.06	5.52	10.48	9.93

Table 3: Error (%) of classification performance and the dynamic memory overheads for LC.

Locally Connected Without Weight Sharing In this section, we extend TinyFoA to LC architecture and compare TinyFoA with BP in the sequential forward pass, gradient calculation, and parameter update. Table 3 reports the classification error and the dynamic training memory overheads.

In the sequential training steps: (1) forward pass: TinyFoA-V-BA reduces the training memory to 0.46 MB, while BP+BW reduces the training memory to 0.71 MB; (2) gradient calculation: TinyFoA-V reduces the training memory to 1.82 MB and BP+BW+BA reduces the training memory to 5.14 MB; (3) parameter update: TinyFoA reduces the training memory to 1.81 MB, while BP+BW+BA+V reduces the training memory to 4.97 MB. Overall, TinyFoA has a comparable error to BP+BW+BA+V and occupies $\sim$2.4 times, $\sim$2.9 times, and $\sim$2.7 times less memory than BP+BW+BA+V in the sequential training steps.

Different Slices in Vertical Layer-Wise Training We increase the slices M from the default 2 to 4, 8, and 16 for TinyFoA in FC and LC. The training memory of TinyFoA in the step of gradient calculation and parameter update is $\frac{1}{M}$ and $\frac{1}{M^2}$ of the training memory without vertical layer-wise training for LC and FC, respectively. We plot the error versus $\frac{1}{M}$, as illustrated in Fig. 4 (a) and (b). For FC, different M results in a comparable error on MNIST and CIFAR-10. For LC, the error will be decreased slightly if we increase the M.

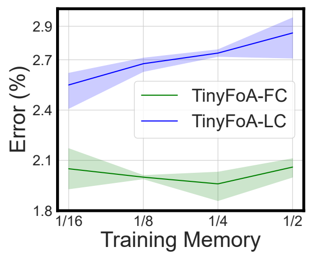
On-Device Learning When M is set to 16, the training memory overheads of FC are 0.5 MB, 0.06 MB, and 0.13 MB and the training memory overheads of LC are 0.07 MB, 0.11 MB, and 0.23 MB for forward pass, gradient calculation, and parameter update, respectively. The Microcontrollers (MCUs)-STM32F7 Series (STMicroelectronics 2024)-feature an ARM Cortex-M7 core, 512KB of RAM, and up to 2MB of Flash Memory (Flash), supporting the training of TinyFoA on the device.

Reducing The Number of Data Exchange In addition to the training memory in RAM, the data exchange between RAM and Flash is investigated here. We consider the number of exchanging the partial slice of weights between RAM and Flash. The partial slice of weights is iteratively updated in different epochs; different partial slices of weights are updated in different epochs. Therefore, the whole training will experience E times of data exchanges (E is the training epochs). Instead, if we perform the data exchange every $\frac{E}{F}$ epochs, the whole training will experience only F times of data exchanges. In this ablation study, we reduce the number of exchanging data from E to F. As shown in Fig. 5 (a) and (b), we reduce the number of exchanging data for TinyFoA in FC and LC with $F = 2$, without any major loss in classification performance on MNIST and CIFAR-10.

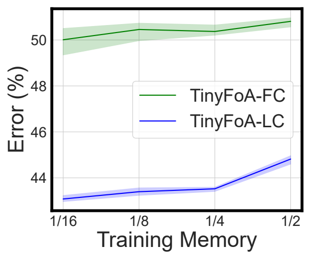
5 Related Work

BP (Rumelhart, Hinton, and Williams 1986) is considered as the state-of-the-art training algorithm and includes the standard forward pass, backward pass (gradient calculation), and parameter update. In the forward pass, as shown in Fig. 2 (a), BP needs to store the parameters (weights and biases) and activations from *all* layers; in the backward pass, BP also needs to retain the parameters and activations from *all* layers because calculating the gradients of weights requires symmetric weights and activations from top to bottom layers; in the parameter update, *all* the layers are updated, utilizing the full-precision parameters and gradient from *all* layers.

Forward-only algorithms are proposed to be the biologically plausible alternatives to BP. Forward-only algorithms reduce memory overheads by replacing the backward pass

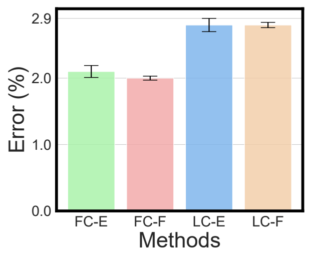


(a) MNIST

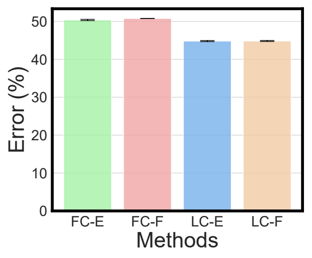


(b) CIFAR-10

Figure 4: Error (%) of vertical layer-wise training with the different number of slices for TinyFoA (The solid line is the mean value and the shaded area is the standard deviation).



(a) MNIST



(b) CIFAR-10

Figure 5: Error (%) of reducing the number of data exchange for TinyFoA, where the mean value and the standard deviation are also reported. E means E times of data exchanges; F means F times of data exchange, where E is the number of epochs and F is 2 by default.

with forward passes. For instance, FA (Lillicrap et al. 2016) and DFA (Nøkland 2016) exploit fixed random matrix to replace the symmetric weights, thus reducing the training memory required for storing weights in gradient calculation. DRTP (Frenkel, Lefebvre, and Bol 2021) introduces the use of a target proxy, without the need for storing the weights in the step of gradient calculation. However, in the step of parameter update, DRTP still needs to store the parameters, the fixed random matrix, and activations from *all* layers because all the layers update the parameters according to the official implementation (Frenkel 2021). Moreover, DRTP shows significant performance degradations compared to BP.

PEPITA (Dellaferrera et al. 2022) replaces the backward pass with a modulated forward pass, adopting a layer-wise training manner. PEPITA eliminates the need to store the entire symmetric weights, while only requiring training memory overheads from a single layer with the *maximum* size in gradient calculation and parameter update (where *max* denotes the maximum required memory overheads among every two consecutive layers). Despite this, PEPITA still needs to store *all* the parameters and activations in the forward pass. Although PEPITA-TL (Srinivasan et al. 2023) is proposed to only store the parameter and activations from a single layer with the *maximum* size in the forward pass, PEPITA-TL exhibits a major degradation in accuracy compared to the original PEPITA (Dellaferrera et al. 2022)—for example, the test accuracy on MNIST is decreased to 92.78% in PEPITA-TL.

Additionally, FF (Hinton 2022), introduced by Hinton, extends the exploration of forward-only algorithms. FF (Hinton 2022) substitutes the forward and backward passes of BP with two forward passes, performing sequential forward pass, gradient calculation, and parameter update in a layer-wise manner. Nevertheless, the positive samples and the negative samples are required to be generated, i.e., requiring twice the number of activations required in training memory.

Although plenty of memory-reducing techniques are available for DNN, such as pruning (Dong and Yang 2019), quantization (Yao et al. 2022), tensor decomposition (Novikov et al. 2015), distillation (Huang et al. 2022), and neural architecture search (Chau et al. 2022), these methods are primarily for reducing inference memory rather than training memory (Han, Mao, and Dally 2016). Additionally, techniques specifically aimed at reducing training memory, such as mixed-precision training (Lee, Sharma, and Aiken 2023), activation data recomputation (Gomez et al. 2017), migration (Jin et al. 2018), compression (Choukse et al. 2020), low-rank gradient descent (Huang et al. 2023), are also available. However, these memory-reducing techniques are predominantly designed for BP-based DNN, rather than the forward-only algorithms. Notably, for PEPITA and FF, binary activations lead to a significant loss in classification performance, as we show empirically in our results in Section 4.

6 Conclusion

In this paper, we propose a memory-efficient forward-only algorithm built on joint horizontal and vertical layer-wise training with binarization, called TinyFoA. TinyFoA outperforms the state-of-the-art forward-only algorithms such as DRTP, PEPITA, and FF in terms of classification performance and training memory overheads. TinyFoA with joint horizontal and vertical layer-wise training and binary weights reduces the training memory overheads by $\sim 96\times$, $\sim 24\times$, and $\sim 12\times$ in the forward pass, gradient calculation, and parameter update, respectively, while maintaining a comparable classification performance to the standard BP training algorithm. TinyFoA is orthogonal to techniques such as mixed-precision training (Lee, Sharma, and Aiken 2023), activation data recomputation (Gomez et al. 2017), migration (Jin et al. 2018), compression (Choukse et al. 2020), low-rank gradient descent (Huang et al. 2023), and can be used together with such techniques to further reduce the dynamic training memory overheads.

Limitations Forward-only algorithms are still in the early stages of development. As a result, in our current work, we do not explore forward-only algorithms for advanced model architectures such as Transformers (Vaswani et al. 2017) and Large language models (LLMs) (Brown et al. 2020; Touvron et al. 2023; Anil et al. 2023; Achiam et al. 2023). The extension of our proposed forward-only approach to advanced model architectures will remain as future work.

Acknowledgments

This research has been partially supported by the Swedish Wallenberg AI, Autonomous Systems and Software Program (WASP), Swedish Research Council, Swedish Foundation for Strategic Research, and ELLIIT Strategic Research Environment. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council (VR) through grant agreement no. 2022-06725.

References

Achiam, J.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Aminifar, A.; Huang, B.; Fahliani, A. A.; and Aminifar, A. 2024. LightFF: Lightweight Inference for Forward-Forward Algorithm. In *27th European Conference on Artificial Intelligence, ECAI-2024*, 1728–1735. IOS Press.

Anil, R.; et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Baghersalimi, S.; Teijeiro, T.; Aminifar, A.; and Atienza, D. 2023. Decentralized federated learning for epileptic seizures detection in low-power wearable systems. *IEEE Transactions on Mobile Computing*.

Bi, M.; Chandrasekharan, S.; and Gniady, C. 2013. {IAMEM}:{Interaction-Aware} Memory Energy Management. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, 267–278.

Bi, M.; Duan, R.; and Gniady, C. 2010. Delay-hiding energy management mechanisms for dram. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 1–10. IEEE.

Brown, T.; et al. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc.

Burbank, K. S.; and Kreiman, G. 2012. Depression-biased reverse plasticity rule is required for stable learning at top-down connections. *PLoS computational biology*, 8(3): e1002393.

Cai, H.; Gan, C.; Zhu, L.; and Han, S. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 11285–11297. Curran Associates, Inc.

Chau, T.; Dudziak, L. u.; Wen, H.; Lane, N.; and Abdelfattah, M. 2022. BLOX: Macro Neural Architecture Search Benchmark and Algorithms. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 30851–30864. Curran Associates, Inc.

Choukse, E.; Sullivan, M. B.; O'Connor, M.; Erez, M.; Pool, J.; Nellans, D.; and Keckler, S. W. 2020. Buddy compression: Enabling larger memory for deep learning and hpc workloads on gpus. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 926–939. IEEE.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28.

Dellaferrera, G.; et al. 2022. Error-driven input modulation: solving the credit assignment problem without a backward pass. In *International Conference on Machine Learning*, 4937–4955. PMLR.

Dong, X.; and Yang, Y. 2019. Network Pruning via Transformable Architecture Search. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Frenkel, C. 2021. Direct Random Target Projection (DRTP) - PyTorch-based implementation. https://github.com/ChFrenkel/DirectRandomTargetProjection [Accessed: 18-May-2024].

Frenkel, C.; Lefebvre, M.; and Bol, D. 2021. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in neuroscience*, 15: 629892.

Gomez, A. N.; Ren, M.; Urtasun, R.; and Grosse, R. B. 2017. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hinton, G. 2022. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*.

Huang, B.; Abtahi, A.; and Aminifar, A. 2023. Lightweight machine learning for seizure detection on wearable devices. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1–2. IEEE.

Huang, B.; Abtahi, A.; and Aminifar, A. 2024. Energy-Aware Integrated Neural Architecture Search and Partitioning for Distributed Internet of Things (IoT). *IEEE Transactions on Circuits and Systems for Artificial Intelligence*.

Huang, S.; Hoskins, B. D.; Daniels, M. W.; Stiles, M. D.; and Adam, G. C. 2023. Low-Rank Gradient Descent for Memory-Efficient Training of Deep In-Memory Arrays. *J. Emerg. Technol. Comput. Syst.*, 19(2).

Huang, T.; You, S.; Wang, F.; Qian, C.; and Xu, C. 2022. Knowledge Distillation from A Stronger Teacher. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 33716–33727. Curran Associates, Inc.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information*

Processing Systems, NIPS'16, 4114–4122. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510838819.

Jaderberg, M.; Czarnecki, W. M.; Osindero, S.; Vinyals, O.; Graves, A.; Silver, D.; and Kavukcuoglu, K. 2017. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, 1627–1635. PMLR.

Jin, H.; Liu, B.; Jiang, W.; Ma, Y.; Shi, X.; He, B.; and Zhao, S. 2018. Layer-centric memory reuse and data migration for extreme-scale deep learning on many-core architectures. *ACM Transactions on Architecture and Code Optimization (TACO)*, 15(3): 1–26.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, Toronto, ON, Canada.

LeCun, Y. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Lee, W.; Sharma, R.; and Aiken, A. 2023. Training with Mixed-Precision Floating-Point Assignments. *Transactions on Machine Learning Research*.

Liao, Q.; Leibo, J.; and Poggio, T. 2016. How important is weight symmetry in backpropagation? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Lillicrap, T. P.; Cownden, D.; Tweed, D. B.; and Akerman, C. J. 2016. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1): 13276.

Mark, R.; Schluter, P.; Moody, G.; Devlin, P.; and Chernoff, D. 1982. An annotated ECG database for evaluating arrhythmia detectors. In *IEEE Transactions on Biomedical Engineering*, volume 29, 600–600.

Niu, S.; Miao, C.; Chen, G.; Wu, P.; and Zhao, P. 2024. Test-Time Model Adaptation with Only Forward Passes. In *Forty-first International Conference on Machine Learning*.

Nøkland, A. 2016. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. P. 2015. Tensorizing neural networks. *Advances in neural information processing systems*, 28.

Patterson, D.; Gonzalez, J.; Hölzle, U.; Le, Q.; Liang, C.; Munguia, L.-M.; Rothchild, D.; So, D. R.; Texier, M.; and Dean, J. 2022. The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink. *Computer*, 55(7): 18–28.

Patterson, D.; Gonzalez, J.; Le, Q.; Liang, C.; Munguia, L.-M.; Rothchild, D.; So, D.; Texier, M.; and Dean, J. 2021. Carbon Emissions and Large Neural Network Training. arXiv:2104.10350.

Pau, D. P.; and Aymone, F. M. 2023. Suitability of Forward-Forward and PEPITA Learning to MLCommons-Tiny benchmarks. In *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, 1–6. IEEE.

Pogodin, R.; Mehta, Y.; Lillicrap, T.; and Latham, P. E. 2021. Towards biologically plausible convolutional networks. *Advances in Neural Information Processing Systems*, 34: 13924–13936.

Rahmani, A. M.; Gia, T. N.; Negash, B.; Anzanpour, A.; Azimi, I.; Jiang, M.; and Liljeberg, P. 2018. Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach. *Future Generation Computer Systems*, 78: 641–658.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542. Springer.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *nature*, 323(6088): 533–536.

Sabry, F.; Eltaras, T.; Labda, W.; Alzoubi, K.; and Malluhi, Q. 2022. Machine learning for healthcare wearable devices: the big picture. *Journal of Healthcare Engineering*, 2022(1): 4653923.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Seo, J.-S. 2022. Special Topic on Energy-Efficient Compute-in-Memory With Emerging Devices. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 8(2): iii–v.

Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; and Xu, L. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5): 637–646.

Sopic, D.; Aminifar, A.; and Atienza, D. 2018. e-glass: A wearable system for real-time detection of epileptic seizures. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. IEEE.

Srinivasan, R. F.; Mignacco, F.; Sorbaro, M.; Refinetti, M.; Cooper, A.; Kreiman, G.; and Dellaferrera, G. 2023. Forward Learning with Top-Down Feedback: Empirical and Analytical Characterization. In *The Twelfth International Conference on Learning Representations*.

STMicroelectronics. 2024. STM32F7 Series. Accessed: 2024-08-14.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Whittington, J. C.; and Bogacz, R. 2019. Theories of error back-propagation in the brain. *Trends in cognitive sciences*, 23(3): 235–250.

Yao, Z.; Yazdani Aminabadi, R.; Zhang, M.; Wu, X.; Li, C.; and He, Y. 2022. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 27168–27183. Curran Associates, Inc.