Dynamic Block-Wise Local Learning Algorithm for Efficient Neural Network Training

Gwangho Lee, Sunwoo Lee, and Dongsuk Jeon

***Abstract*— In the backpropagation algorithm, the error calculated from the output of the neural network should backpropagate the layers to update the weights of each layer, making it difficult to parallelize the training process and requiring frequent off-chip memory access. Local learning algorithms locally generate error signals which are used for weight updates, removing the need for backpropagation of error signals. However, prior works rely on large, complex auxiliary networks for reliable training, which results in large computational overhead and undermines the advantages of local learning. In this work, we propose a local learning algorithm that significantly reduces computational complexity as well as improves training performance. Our algorithm combines multiple consecutive layers into a block and performs local learning on a block-by-block basis, while dynamically changing block boundaries during training. In experiments, our approach achieves 95.68% and 79.42% test accuracy on the CIFAR-10 and CIFAR-100 datasets, respectively, using a small fully connected layer as auxiliary networks, closely matching the performance of the backpropagation algorithm. Multiply-accumulate (MAC) operations and off-chip memory access also reduce by up to 15% and 81% compared to backpropagation.**

***Index Terms*— Artificial neural networks, concurrent computing, local learning, supervised learning.**

I. Introduction

Recent advances in neural network optimization techniques have enabled efficient acceleration of the inference process on hardware. Contrarily, training deep neural networks remains a time-consuming and computationally extensive task, making it unsuitable for efficient neural network training processors. However, recent studies on federated learning [1] and transfer learning [2] suggest that they could benefit from on-chip training in edge devices with limited hardware resources. For training deep neural networks, backpropagation is the most common and effective way. Fig. 1 depicts this process for a typical convolutional neural network (CNN) structure. Black arrows represent the forward propagation of inputs, whereas red arrows represent the backpropagation of error signals.

While the backpropagation algorithm demonstrates excellent training performance on various neural networks, it has some inherent inefficiencies from a hardware perspective. First, each layer in the network must wait for the error signals to propagate back from the upper layers to compute the gradient estimates, which makes it impossible to parallelize the training process. Second, the activations of each layer must be temporarily stored in the memory as they are required later for weight update. For deep neural networks, the amount of those activations well exceed available on-chip memory space; hence, they must be transferred to external memory such as dynamic random-access memory (DRAM), resulting in costly off-chip memory access which costs several orders of magnitude more power than on-chip memory access [3], [4].

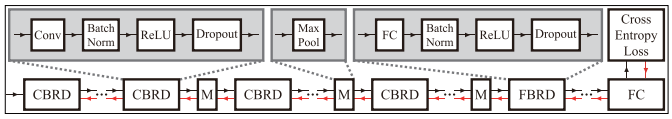


Fig. 1. Overview of CNN training process using backpropagation. CBRD (FBRD) block represents Conv (FC) + BatchNorm + ReLU + Dropout layers.

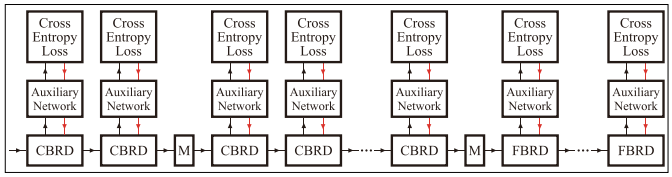


Fig. 2. Layer-wise local learning algorithm (**G**1).

Several alternative training methods have been proposed to overcome these issues [5]–[7]. Local learning algorithms replace the backpropagated errors, with the errors locally generated by the auxiliary networks [8], [9]. Fig. 2 displays an example of a layer-wise local learning algorithm in which weight update is completed within each layer without backpropagating errors between layers. Therefore, local learning algorithms allow for update unlocking and eliminate the drawbacks of the backpropagation algorithm. In other words, they significantly reduce off-chip memory access and enable simultaneous learning of multiple layers in parallel. Furthermore, if the auxiliary networks are small, the number of multiply-accumulate (MAC) operations can be significantly reduced compared to calculating gradients in the main network. Recently, it was reported that a layer-wise local learning algorithm could scale to a large dataset [10], and combining the loss obtained from the similarity measure [11] with the local loss could further improve the performance [12]. In addition, block-wise local learning demonstrated training performance comparable to that of the backpropagation algorithm [13]. However, these approaches necessitate complex auxiliary networks and hence increase MAC operations and required on-chip memory size.

In this brief, we propose a novel block-wise local learning algorithm with both low computational complexity and high training performance. The algorithm employs simple auxiliary networks but delivers training performance similar to backpropagation on the target datasets.

II. Proposed Local Learning Algorithm

A. Conventional Block-Wise Local Learning Algorithm

Layer-wise local learning algorithms attach additional networks ("auxiliary network") to each layer in the neural network. The parameters of each layer are updated through the cross-entropy loss

Manuscript received March 10, 2021; revised June 8, 2021; accepted July 11, 2021. Date of publication July 26, 2021; date of current version August 30, 2021. This work was supported in part by the National Research Foundation of Korea under Grant NRF-2019R1C1C1004927 and Grant NRF-2021M3F3A2A01037633 and in part by the Information Technology Research Center Support Program supervised by the Institute for Information & Communications Technology Planning & Evaluation under Grant IITP-2021-2018-0-01421. *(Corresponding author: Dongsuk Jeon.)*

The authors are with the Graduate School of Convergence Science and Technology, Seoul National University, Seoul 08826, South Korea (e-mail: djeon1@snu.ac.kr).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3097341.

Digital Object Identifier 10.1109/TVLSI.2021.3097341

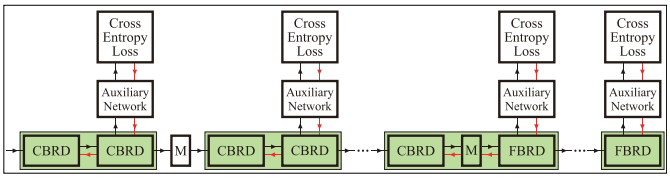


Fig. 3. Block-wise local learning algorithm with block size = 2 (**G**2).

obtained from the output of the corresponding auxiliary network and the training labels (see Fig. 2). Instead of minimizing the global objective function, learning takes place in a way to minimize the objective function for each layer. This still allows upper layers to learn more complex features by combining the features learned from the previous layers. In general, these algorithms suffer performance degradation compared to backpropagation. This tendency becomes stronger as the neural network gets deeper [9] or when the neural network is trained on more complex datasets [14]. Since each layer is trained independently, adding more layers to the neural network does not necessarily improve training performance.

Block-wise local learning divides the neural network into blocks composed of multiple layers and trains each block using the error signals locally generated by the auxiliary networks [7], [8]. Similar to layer-wise local learning, the loss calculated for each auxiliary network backpropagates only through the layers inside the block. This approach shows a large performance boost over layer-wise local learning since more layers are now grouped together during training. Increasing the number of layers per block can improve the training performance by enhancing the representation power of each block. In other words, the direction of the weight update in each layer will be more accurate if there are more layers in the block and the error signals are backpropagated through the layers within a block.

From now on, we refer to the block-wise local learning as **G**#. The number at the end (#) represents the number of layers in each block. For instance, **G**1 corresponds to the layer-wise local learning algorithm. Fig. 3 shows the structure of **G**2.

B. Proposed Dynamic Block-Wise Local Learning Algorithm

Block-wise local learning algorithm offers more reliable training, but the training process is still entirely contained within each block. Therefore, no information is shared between blocks during training, which is the main reason that it requires a complex auxiliary network for high training performance. We address this issue by dynamically alternating the scope of blocks during training; we propose to change block boundaries continuously in the training process so that the key information can be inherently transferred between layers, as backpropagation explicitly does. Specifically, we move back and forth the block boundaries by a fixed step with a predefined period (e.g., epoch) while maintaining the number of layers in each block. We denote this method as **G**#**S**#. The number after S (Shift) indicates the number of layers by which the block boundary shifts in each epoch. Fig. 4 shows **G**2**S**1 as an example. In each epoch, different sets of auxiliary networks are activated for error signal generation. Note that some layers in the front that do not belong to any block in each epoch are frozen and not trained.

Consider a K-layer neural network consisting of convolutional and fully connected layers. If we define the parameters of the ith layer as θ_i, the activation x_i of the ith layer can be expressed as $x_i = f_i(x_{i-1}; \theta_i)$. Given that the auxiliary network corresponding to the ith layer is A_i, the prediction z_i can be obtained as $z_i = A_i(x_i; \phi_i)$, where ϕ_i represents the parameters of A_i. Assuming that the neural network is trained with **G**X**S**Y algorithm, all blocks constituting the

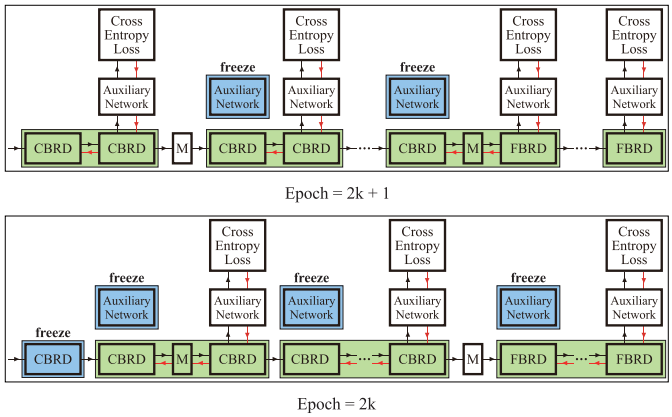


Fig. 4. Dynamic block-wise local learning algorithm with block size = 2 and shift = 1 (**G**2**S**1).

Algorithm 1 Proposed Algorithm (**G**X**S**Y)

Require: Learning rate ϵ, neural network with K layers
Initialize: Parameters Θ_{1K}, Φ_{XK}
▷ $\Theta_{1K} = \{\theta_i \mid 1 \le i \le K\}, \Phi_{XK} = \{\phi_j \mid X \le j \le K\}$
Ensure: $X > Y$
$s \leftarrow 0$
while training **do**
  Freeze all layers up to the s_{th} layer
  for $t \leftarrow 1, 2, 3, \ldots, T$ **do**
    Sample minibatch from the dataset $\{x_0^{(t)}, y^{(t)}\}$
    $i \leftarrow 1$
    for $j = X + s, 2X + s, 3X + s, \ldots, K$ **do**
      $x_j^{(t)} \leftarrow F_{ij}(x_{i-1}^{(t)}; \Theta_{ij})$ ▷ $F_{ij} = f_j \circ f_{j-1} \circ \cdots \circ f_i$
      $z_j^{(t)} \leftarrow A_j(x_j^{(t)}; \phi_j)$
      Compute gradient $\hat{\mathbf{g}} \leftarrow \nabla_{\Theta_{ij}, \phi_j} \mathcal{L}(z_j^{(t)}, y^{(t)}; \Theta_{ij}, \phi_j)$
      Apply update $\Theta_{ij} \leftarrow \Theta_{ij} - \epsilon \hat{\mathbf{g}}_{\Theta_{ij}}, \phi_j \leftarrow \phi_j - \epsilon \hat{\mathbf{g}}_{\phi_j}$
      $i \leftarrow j + 1$
    end for
  end for
  $s \leftarrow s + Y$
  if $s \ge X$ **then**
    $s \leftarrow 0$
  end if
end while

neural network will always contain X layers except for the last block. Select a block that covers from the ith layer to the jth layer. Then, the prediction z_j of this block can be expressed as

$$z_j = A_j(f_j(f_{j-1} \circ \cdots \circ f_i(x_{i-1}; \theta_i, \ldots, \theta_{j-1}); \theta_j); \phi_j). \quad (1)$$

The layers in this block are trained using the following objective function:

$$\min_{\theta_i, \ldots, \theta_j, \phi_j} \mathcal{L}(z_j, y; \theta_i, \ldots, \theta_j, \phi_j). \quad (2)$$

Unlike backpropagation, when calculating the gradient estimate to update the weights of a layer, only the parameters within the block that the layer belongs to and its auxiliary network are needed. The layers that share the same objective vary every epoch due to the change in block scopes; similarly, the auxiliary network corresponding to each block continues to change. The formal description of the proposed algorithm can be found in Algorithm 1. In the conventional block-wise learning algorithms (**G**#), each block is

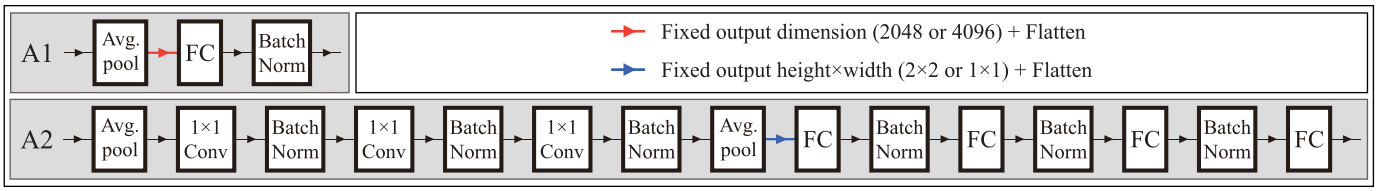


Fig. 5. Two auxiliary network structures used in the experiments.

trained independently during the entire training process and the layers do not receive the error signals backpropagated from upper layers that belong to a different block. Instead, they have to rely only on the error signals locally generated by their own auxiliary network. This increases the probability that the change in the weight due to the gradient does not convert the feature space appropriately considering upper layers, resulting in performance degradation.

Contrarily, in the proposed algorithm with dynamic block boundary (**G#S#**), while decoupled learning is performed in each block, it is guaranteed that two adjacent layers have a chance to be trained together in some of the epochs due to changing block boundaries, enabling network-wise training as the backpropagation algorithm does. Also, since each layer learns using multiple auxiliary networks, the generalization ability of the neural network can be improved.

III. Experiments

To validate the performance of the proposed learning algorithm, we applied conventional and our training algorithms to deep neural network training on widely used datasets such as CIFAR-10 and CIFAR-100. For experiments, we selected VGGNets and VGG-like CNNs [12] that have been employed in previous works for fair comparisons. Since local learning algorithms experience large performance degradation when applied to deep neural networks, we employed the deepest VGG networks for a more reliable comparison. The algorithms were implemented on PyTorch using Nvidia RTX 2080 Ti and Titan X GPUs.

A. Auxiliary Network Structure

Selecting an optimal auxiliary network structure is very important for local learning, since the training performance largely depends on the structure of the auxiliary networks. In general, auxiliary networks with good representation capability (e.g., deep CNN) tend to provide better training performance. However, even a single 3×3 convolutional layer in the auxiliary networks would incur a large amount of computational overhead, undermining the advantages of local learning. Hence, it is critical to minimize the size and complexity of the auxiliary networks, while maintaining the training performance. To observe how the training performance of each algorithm varies with the size and complexity of auxiliary networks, we employ two different auxiliary networks in Fig. 5 for experiments.

A1 is a simple auxiliary network with a single fully connected layer, which was employed in prior works [12], [15], but has an additional batch normalization layer at the end. To be attached to layers with different dimensions, the input to A1 is changed to a fixed dimension through average pooling and flattened. The batch normalization layer was added because it showed better generalization performance in local learning algorithms with a negligible increase in MAC operations and parameters. A2 is a relatively large network that was used in [13], but was slightly modified to support the datasets used in our experiments: when the activation size is smaller than 8×8, the second average pooling is ignored. A2 has 4×4 average pooling layer followed by three 1×1 convolution layers and the activation is reduced to 2×2 by average pooling and flattened to pass through four fully connected layers. Table I displays the maximum and the average number of MAC operations and parameters of the auxiliary networks of VGG-16 when applying layer-wise local learning algorithm. The auxiliary networks' MAC operations account for 0.1% and 11.2% of the total MAC operations when using A1 and A2, respectively, and this ratio remains similar for other models. Also, the number of parameters increases by 1.8% and 123.1% for A1 and A2, respectively, confirming that the A1 structure incurs significantly lower computational overhead. Furthermore, our algorithm (**G2S1**) requires only 0.75% more parameters than conventional block-wise local learning (**G2**) for VGG-16 trained on CIFAR-10, suggesting that the overhead of moving block boundaries is ignorable.

TABLE I
MAC Operations and Parameters of the Auxiliary Networks

Network	MAC (max)	MAC (avg.)	Parameters (max)	Parameters (avg.)
A1	61,440	57,344	20,480	19,115
A2	11,827,200	6,098,629	3,352,576	1,664,236
Layers of VGG-16	75,497,472	41,969,254	2,359,296	1,085,555

TABLE II
CIFAR-10 Test Accuracy Comparison (%)

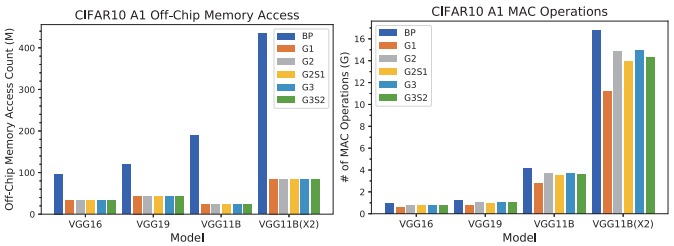
Model	BP	**G1** (A1 / A2)	**G2** (A1 / A2)	**G2S1** (A1 / A2)	**G3** (A1 / A2)	**G3S2** (A1 / A2)
VGG16	93.99	89.87 / 92.36	91.44 / 93.81	92.26 / 94.25	92.84 / 93.91	93.68 / 94.23
VGG19	93.85	89.71 / 92.26	91.57 / 93.74	92.26 / 93.98	93.15 / 93.99	93.66 / 94.15
VGG11B	95.08	92.41 / 92.06	93.60 / 94.40	93.95 / 94.97	94.59 / 95.07	95.11 / 95.58
VGG11B(×2)	95.46	93.00 / 92.72	94.36 / 94.61	94.68 / 95.39	94.97 / 95.40	95.68 / 95.75

B. CIFAR-10

We first trained neural networks on CIFAR-10 using different training algorithms. The input dimension of the fully connected layer in A1 was set to 2048 for VGG-16 and VGG-19, and 4096 for VGG-11B and VGG-11B (×2). The dropout rate was 0.05 for VGG-16 and VGG-19, 0.2 for VGG-11B, and 0.25 for VGG-11B (×2). We trained the models using SGD for 400 epochs with a momentum of 0.9 and a weight decay factor of 5×10^{-4}.

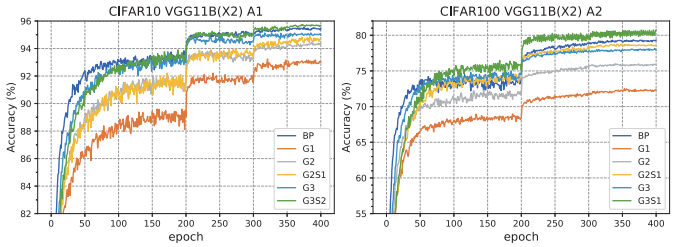
From Table II, it can be seen that the overall performance of each algorithm follows the trend of **G1** < **G2** < **G2S1** < **G3** < **G3S2** $\approx$ BP. This suggests that applying the proposed dynamic block assignment (**G#S#**) does boost the performance of conventional block-wise local learning (**G#**) and closely matches or even outperforms backpropagation using a simple A1 auxiliary network. Using A2 auxiliary network showed better training performance in most cases, but our algorithm exhibits narrower performance gap between the two cases compared to the conventional block-wise local learning, confirming its high generalization performance.

Off-chip memory access count and MAC operations of local learning algorithms when using A1 are shown in Fig. 6. Those values



(a) (b)

Fig. 6. (a) Off-chip memory access. (b) MAC operation comparisons for CIFAR-10.



(a) (b)

Fig. 7. Example training curves of VGG-11B (×2) on (a) CIFAR-10 and (b) CIFAR-100.

TABLE III

CIFAR-100 TEST ACCURACY COMPARISON (%)

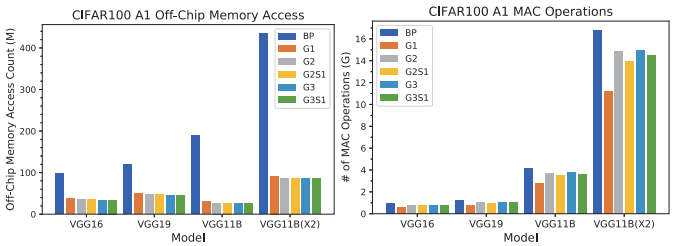
Model	BP	**G**1 (A1 / A2)	**G**2 (A1 / A2)	**G2S**1 (A1 / A2)	**G**3 (A1 / A2)	**G3S**1 (A1 / A2)
VGG16	72.15	67.57 / 65.22	69.95 / 69.18	72.32 / 71.23	72.67 / 69.38	74.99 / 74.81
VGG19	73.36	68.66 / 64.0	70.09 / 67.36	71.69 / 69.86	72.3 / 69.13	74.82 / 74.46
VGG11B	77.4	72.88 / 69.39	75.46 / 73.35	76.7 / 76.12	76.81 / 76.25	79.03 / 79.52
VGG11B(×2)	79.32	74.44 / 72.34	77.25 / 75.81	77.45 / 78.59	78.8 / 77.89	79.42 / 80.76

are estimated using the methodology in [9], which assumes on-chip memory can hold all activations and parameters of a single block and its corresponding auxiliary network. Compared to backpropagation, off-chip memory access is reduced by 66%, 64%, 88%, and 81% in VGG-16, VGG-19, VGG-11B, and VGG-11B (×2), respectively. MAC operations are decreased by 33%, 12%, 17%, 11%, and 15% when using **G**1, **G**2, **G2S**1, **G**3, and **G3S**2 regardless of the neural network structure. Using A2 results in lower efficiency due to its larger size, but our approach exhibits similar reduction in MAC operations and off-chip memory access.

C. CIFAR-100

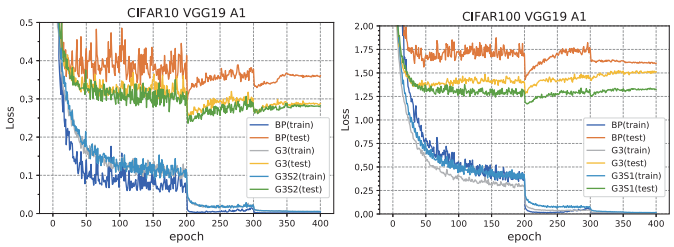
We trained the same models on the CIFAR-100 dataset to observe scalability. The input dimension of the fully connected layer in A1 is 2048 for VGG-16 and VGG-19, and 4096 for VGG-11B and VGG-11B (×2). The dropout rate was 0.05 for VGG-16 and VGG-19, and 0.1 for VGG-11B and VGG-11B (×2). We trained the models using SGD for 400 epochs with a momentum of 0.9 and weight decay factor of 5×10^{-4}.

Training results in Table III can be summarized as: **G**1 < **G**2 < **G2S**1 < **G**3 < BP < **G3S**1. The trend is similar to that of CIFAR-10, but **G3S**1 showed better performance than backpropagation in all cases regardless of the type of auxiliary networks and the models. Note that **G3S**1 exhibits better performance than **G3S**2 for CIFAR-100, so **G3S**1 is used in CIFAR-100 experiments. Similar to CIFAR-10 training, applying dynamic block boundary enhances the training performance in all cases. Fig. 7 displays the training curves, where our algorithm shows reliable training, while closely matching or outperforming backpropagation in all cases.



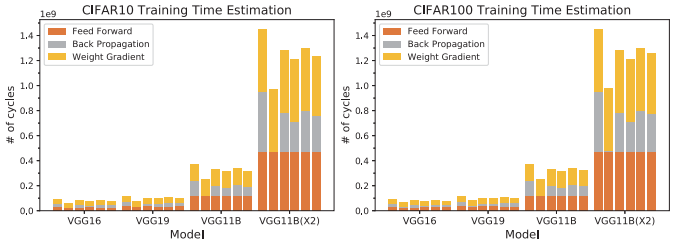
(a) (b)

Fig. 8. (a) Off-chip memory access. (b) MAC operation comparisons for CIFAR-100.



(a) (b)

Fig. 9. Train and test loss convergence of VGG-19 with A1 on (a) CIFAR-10 and (b) CIFAR-100.



(a) (b)

Fig. 10. Training time estimation on ASIC for algorithms in Fig. 8 on (a) CIFAR-10 and (b) CIFAR-100.

Fig. 8 compares off-chip memory access and MAC operations. Off-chip memory access decreases by 64%, 61%, 86%, and 80%, and MAC operations are reduced by 32%, 12%, 16%, 11%, and 14% in VGG-16, VGG-19, VGG-11B, and VGG-11B (×2) for **G3S**1, compared to backpropagation.

D. Comparison With Backpropagation

In Fig. 9, all algorithms have similar convergence speed for both train and test losses. While backpropagation shows slightly lower train loss, our algorithm (**G3S**1) achieves lower test loss, which better reflects the model accuracy. In addition, a smaller gap between training and test losses of our algorithm suggests its superior generalization capability. To demonstrate the speedup of training process using our algorithm, we experimentally trained neural networks using multiple GPUs. For multi-GPU training, we run independent processes on each GPU using PyTorch distributed package. NCCL backend is used to move layer parameters and SGD optimizer states between GPUs since the block boundaries continue to change. Simulations show 2.2× faster training in VGG-16 using five GPUs and 2.4× faster training in VGG-19 using six GPUs compared to backpropagation using a single GPU.

Additionally, we estimated training speed using a training processor ASIC in [16]. **G3S** delivers up to 20% speedup compared to BP as shown in Fig. 10. Note that this processor is designed for edge

device and has limited on-chip memory space; hence, the training speed is improved mainly by MAC operation reduction and the result is consistent with Fig. 6(b). A larger improvement is expected if our algorithm is used on a processor with sufficient on-chip memory size [17], [18].

IV. Conclusion

In this brief, we proposed a dynamic block-wise local learning algorithm that continuously alters block boundaries during training. By sharing critical information between layers regardless of the block boundaries, it has been experimentally shown that the algorithm noticeably improves training performance. Our approach demonstrates competitive training performance for deep neural networks using a simple auxiliary network. Our approach achieves 14% and 80% lower MAC operations and external memory access than backpropagation for CIFAR-100 dataset with similar or better training performance, suggesting that it has a high potential for efficient hardware implementation of deep neural network training.

Acknowledgment

The EDA tool was supported by the IC Design Education Center.

References

[1] K. Stewart and Y. Gu, "One-shot federated learning with neuromorphic processors," 2020, *arXiv:2011.01813*. [Online]. Available: http://arxiv.org/abs/2011.01813

[2] M. A. Hussain and T.-H. Tsai, "Memory access optimization for on-chip transfer learning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 4, pp. 1507–1519, Apr. 2021.

[3] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep./Oct. 2011.

[4] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2010, pp. 363–374.

[5] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," 2017, *arXiv:1707.04585*. [Online]. Available: http://arxiv.org/abs/1707.04585

[6] H. Zhuang, Y. Wang, Q. Liu, S. Zhang, and Z. Lin, "Fully decoupled neural network learning using delayed gradients," 2019, *arXiv:1906.09108*. [Online]. Available: http://arxiv.org/abs/1906.09108

[7] Z. Huo, B. Gu, and H. Huang, "Training neural networks using features replay," 2018, *arXiv:1807.04511*. [Online]. Available: http://arxiv.org/abs/1807.04511

[8] M. Jaderberg *et al.*, "Decoupled neural interfaces using synthetic gradients," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1627–1635.

[9] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers Neurosci.*, vol. 12, p. 608, Aug. 2018.

[10] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Greedy layerwise learning can scale to ImageNet," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 583–593.

[11] N. Kriegeskorte, M. Mur, and P. A. Bandettini, "Representational similarity analysis—Connecting the branches of systems neuroscience," *Frontiers Syst. Neurosci.*, vol. 2, p. 4, Nov. 2008.

[12] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4839–4850.

[13] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled greedy learning of CNNs," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 736–745.

[14] S. Bartunov, A. Santoro, B. A. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," 2018, *arXiv:1807.04587*. [Online]. Available: http://arxiv.org/abs/1807.04587

[15] Q. Teng, K. Wang, L. Zhang, and J. He, "The layer-wise training convolutional neural networks using local loss for sensor-based human activity recognition," *IEEE Sensors J.*, vol. 20, no. 13, pp. 7265–7274, Jul. 2020.

[16] J. Park, S. Lee, and D. Jeon, "A 40 nm 4.81 TFLOPS/W 8b floating-point training processor for non-sparse neural networks using shared exponent bias and 24-way fused multiply-add tree," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 1–3.

[17] Y. Jiao, L. Han, and X. Long, "Hanguang 800 NPU—The ultimate AI inference solution for data centers," in *Proc. IEEE Hot Chips Symp. (HCS)*. Washington, DC, USA: IEEE Computer Society, Aug. 2020, pp. 1–29.

[18] Z. Huo, B. Gu, and H. Huang, "Decoupled parallel backpropagation with convergence guarantee," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2098–2106.