IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Speculative Backpropagation for CNN Parallel Training

Sangwoo Park[1], Taeweon Suh[2]
[1, 2]Computer Science and Engineering, Korea University

Corresponding author: Taeweon Suh (e-mail: suhtw@korea.ac.kr).

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00533, Research on CPU vulnerability detection and validation / No.2019-0-01343, Regional strategic Industry convergence security core talent training business).

ABSTRACT The parallel learning in neural networks can greatly shorten the training time. Its prior efforts were mostly limited to distributing inputs to multiple computing engines. It is because the gradient descent algorithm in the neural network training is inherently sequential. This paper proposes a novel CNN parallel training method for image recognition. It overcomes the sequential property of the gradient descent and enables the parallel training with the speculative backpropagation. We found that the *Softmax* and *ReLU* outcomes in the forward propagation for the same labels are likely to be very similar. This characteristic makes it possible to perform the forward and backward propagation simultaneously. We implemented the proposed parallel model with CNNs in both software and hardware, and evaluated its performance. The parallel training reduces the training time by 34% in CIFAR-100 without the loss of the prediction accuracy compared to the sequential training. In many cases, it even improves the accuracy.

INDEX TERMS Deep learning, Parallel training, Speculative backpropagation, Training accelerator, FPGA

I. INTRODUCTION

Artificial neural networks (ANNs) have successfully been applied in various applications such as text recognition [1], image classification [2], and speech recognition [3]. Especially, deep neural networks (DNNs) are drawing attention due to its accuracy and practicability in tandem with the advancement of computing technology. Large scale models in DNN could improve its inference accuracy [4-13]. Ciresan et al [6] reported that the prediction performance could be greatly improved as the number of model parameters such as neurons and layers increases. For example, ResNets [7] with more than 100 layers achieved a 3.57% top-5 error rate on the ImageNet test set.

As a DNN model grows in size, there are a large number of vector-matrix multiplication (VMM) operations for training. The computational complexity of VMM usually grows with $O(n^2)$. Thus the computational complexity of larger networks increases proportionally with the number of layers and parameters. It means that DNN requires a huge amount of time for training. Accordingly, many prior works were aimed at speeding up the training [13-25]. Some studies [13-15] have been pursuing in the direction of the parallel training where the multiple devices undertake each portion of the DNN computations simultaneously. There are also a few works [18, 19] for optimizing synchronizations, which occur when multiple devices process DNN calculations in parallel. There are efforts to develop and utilize DNN hardware accelerators for training. HiSilicon [20] introduced a specialized neural processing unit (NPU) aimed at processing vector and matrix-based computations fast, which are common operations in deep learning. More recently, Cerebras, a startup company, has developed a 16nm wafer-sized processor array [21] for training neural networks. Intel also has introduced a neural network processor dubbed as Spring Crest [21] with a direct proprietary interconnect, with which it avoids passing through external memory for the efficient processing of large neural networks.

For training, the forward propagation should proceed before the backpropagation. It is because the gradient descent algorithm is inherently sequential. The gradient descent algorithm is used in the aforementioned studies [7-15] for the weight update. In this paper, we propose a novel idea of breaking the sequential property of the gradient descent algorithm for CNN parallel training. It enables performing the forward and backward propagations in parallel. The core idea is speculating the forward outcomes for backpropagation. This

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

paper shows that the speculative backpropagation speeds up the training time without the prediction accuracy loss (in some cases, it even improves the accuracy). We implemented the training accelerator in both software and hardware. The experiments show that the parallel training reduces the training time by up to 38%. The hardware accelerator exhibits a superior performance per watt because it only requires a 1.2 % of more memory.

II. RELATED WORK

There are studies on distributed and parallel DNN training using GPUs [18, 19, 22-25]. The parallel training is largely divided into data parallelism and model parallelism; The data parallelism means that different portions of the input are processed in parallel on computing engines where the same network model is deployed to. Since each computing engine is processing the same DNN with weights and related parameters, there is an inevitable synchronization issue. Accordingly, several studies were focusing on synchronization optimization [18, 19]. Zinkevich et al [18] proposed a data-parallel stochastic gradient descent algorithm, where the training data is accessed locally and the communication occurs at the very end. Feng Niu et al [19] studied the lock-free approach by taking advantage of the sparse feature of neural networks. Ahn et al [25] proposed a virtual shared memory framework in parallel distributed deep learning. It enables the memory sharing in remote nodes and improves the communication performance via parameter sharing.

The model parallelism means that different layers of the network are assigned to different cores. Thus, the data transfer occurs after processing assigned layers on each core. Some works [22, 24] combined the model parallelism and the data parallelism together. DistBelief [22] utilizes 512 cores, each of which is assigned a portion of the training data and hidden layers. Then, it performs asynchronous weights update with the centralized parameter server. Krizhevsky [24] proposed applying the data parallelism to convolutional layers and the model parallelism to fully connected layers in CNN.

There are also prior works on the hardware accelerators for speeding up the training [26-29]. Qiu et al [26] proposed a method to reduce the resource consumption of the convolution operation for large-scale image classification and evaluated its performance using FPGAs. Gaunt et al [27] proposed the DNN training accelerator by applying synchronous and asynchronous pipelines for speeding up the training. There are some research works employing low-bit quantization for ANN weights and activations [30, 31]. Such a quantization greatly reduces the model size and computational complexity, making it suitable for hardware implementation. S. Fox et al [32] implemented a training accelerator based on 8-bit integer operations. It processes the forward and backward computations on FPGA with 8-bit integers, while the weight update computation is processed in full-precision on an ARM processor. L. Yang et al [33] implemented a binarized neural network (BNN) on FPGA, which replaces the original binary convolution layer with two parallel binary convolutional layers for fast inference. These previous studies [22-33] follow the sequential order in processing according to the gradient descent algorithm.

There are several studies on breaking the forward and backward dependency for speeding up the training [34-36]. Jaderberg et al [34] proposed removing the locking in backpropagation by employing additional neural networks to approximate gradients. In the backward pass, all neurons use the approximated gradients to update weights, through which it avoids incurring a delay. Nøkland et al [35] broke the local dependencies between successive layers in the backward pass. They used the direct feedback alignment (DFA) where the hidden layers receive the error information from the output layer directly using a random matrix. Our work is different in that the history information (instead of the random matrix in [35]) is used for the weight update, and the additional neural networks in [34] are not required.

III. THE NEURAL NETWORK TRAINING

ANN is a group of multiple neurons and each neuron is connected to the next layer neurons through the weights. ANN can be trained to infer the target outputs based on inputs. The convolutional neural network (CNN) is one type of ANNs and is prevalently used in the image recognition. It is because CNN trains the filters capturing the spatial features from an input image. For the neural network training, three sequential operations are performed: forward propagation, backpropagation, and weight update. In the forward propagation process, input data is propagated from the input layer to the output layer; Each neuron computes a weighted sum of the inputs from the connected neurons in its prior layer, and then adds it with a bias, as shown in Eq. (1). Its output goes through an activation function in Eq. (2) that determines data to pass to the next layer. The widely used activation functions are *Rectified Linear Unit* (*ReLU*), *Tanh*, and *Sigmoid*. The ReLU is especially used in many ANNs because it effectively reduces the computation cost. The ReLU propagates zero to the next layer when the input is negative, and otherwise bypasses the input value to the next layer. CNN has pooling layers, which reduce the dimensions of the data by combining the outputs of the neurons. Max pooling selects the maximum value of neurons from its prior layer and propagates it to the next layer. Average pooling takes the average from a cluster of neurons at its prior layer. In the output layer, the *Softmax* function in Eq. (3) is widely used in the deep learning architecture [37-39]. It computes the probability distribution of outcomes (y_z^o).

$$u_j^l = \sum_{i=1}^{N} w_{ij}^l x_i^l + Bias^l \quad (1)$$

$$y_j^l = f(u_j^l) \quad (2)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

$$y_z^o = \frac{e^{u_z^o}}{\sum_{i=1}^{n} e^{u_i^o}} \tag{3}$$

where $1 \le i \le N, 1 \le l \le O$ (i, j, z, k = neuron index, l = layer index)

The backpropagation is used to adjust the weights (w_{ij}^l) by calculating derivatives. This phase begins from the output layer, which is based on Softmax in our work. The derivative in the output layer is expressed in Eq. (4). It calculates the difference between the forward propagation outcome (y_z^o) and target output (t_z). The derivative of the error with respect to the weight is calculated with Eq. (5). The derivative of the ReLU activation function is calculated with Eq. (6), which is zero when the outcome of Eq. (2) is zero, and one otherwise. In the max pooling, the error is passed back to the next layer from the winning neuron which has a maximum value in the forward pass. All the other neurons get a zero gradient. In the average pooling, the error is computed by dividing by the pooling size.

$$\frac{dE}{dy_z^o} = y_z^o - t_z \tag{4}$$

$$\frac{dE}{dy_k^l} = \sum_z w_{kz}^{l+1} \frac{dE}{dy_z^{l+1}} \tag{5}$$

$$\delta_k^l = \frac{dE}{du_k^l} = \frac{dE}{dy_k^l} \frac{dy_k^l}{du_k^l} \tag{6}$$

In the ANN training, the weights are adjusted based on the errors computed in the backpropagation. First, Δw_{ij}^l is calculated by multiplying the result of the backpropagation δ_i^l and the result of the forward propagation y_j^{l-1}, as shown in Eq. (7). Weights (w_{ij}^l) are then updated according to the learning rate η, which determines the degree of learning in Eq. (8). This process is repeated for all the weights.

$$\Delta w_{ij}^l = \delta_i^l y_j^{l-1} \tag{7}$$

$$w_{ij}^l = w_{ij}^l - \Delta w_{ij}^l \eta \tag{8}$$

IV. SPECULATIVE BACKPROPAGATION

In the ANN training, the backpropagation is performed based on the forward propagation outcomes. It means that the backpropagation can be carried out only after the forward propagation is finished. However, if it is feasible to speculate the forward outcomes in advance, the backpropagation can be performed simultaneously with the forward computation. We have found one interesting behavior in the ANN training that makes the speculation possible; The Softmax and ReLU outcomes for the same labels in the temporally near-forward propagations tend to be very similar. Thus, the previous forward outcomes can be used for the current backpropagation. Figure 1 illustrates the comparison between the typical sequential training scheme and the proposed parallel method. In the typical training scheme shown in Figure 1(a), the backward computation is performed only after the forward propagation. In our proposed method shown in Figure 1(b), the forward and backward computations occur at the same time. The backward propagation in Figure 1(b) is based on the accumulated previous forward outcomes, which is detailed in Section IV-A and Section IV-B. To demonstrate its feasibility and practicability, we experimented with two types of CNN models: modified LeNet [9] and VGG16 [10], where the ReLU, Softmax, and average pooling are used. The MNIST handwritten and fashion [40], and the CIFAR-10 and CIFAR-100 [41] were used for the LeNet and VGG16, respectively. We used a *Weight decay*, *Adam* [42], *Dropout* and *Data augmentation* techniques for training the CIFAR datasets to improve the training accuracy. The batch size is 32 for both MNIST and CIFAR.

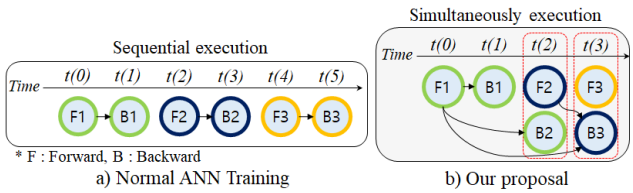

FIGURE 1. a) Backpropagation ONLY after forward propagation b) Simultaneous execution of forward and backward propagations with the speculative backpropagation.

A. SPECULATION OF SOFTMAX OUTCOMES IN OUTPUT LAYER

During the ANN training process, the inputs with the same label are very likely to generate similar Softmax outcomes. To demonstrate the similarity, we use a metric, called the mean-difference. The mean-difference is the average of the differences between Softmax outcomes for the inputs with the same label. For example, assuming that there are four neurons in the output layer, let's say that the current Softmax outcomes for the label 2 are $y_1^O = 0.1$, $y_2^O = 0.7$, $y_3^O = 0.1$, and $y_4^O = 0.1$. In the next training step, if the Softmax outcomes for the same label are $y_1^O = 0.05$, $y_2^O = 0.74$, $y_3^O = 0.14$, and $y_4^O = 0.07$ the mean difference is 0.04 (= $(0.05 + 0.04 + 0.04 + 0.03) / 4$).

Figure 2 shows the mean-difference distribution during the training process. For 95% of MNIST handwritten in Figure 2 (a), the mean-difference is smaller than 0.02. For roughly 90% of input images in the MNIST and CIFAR, it is smaller than 0.1. Moreover, as the training progresses, the mean-difference becomes smaller and smaller. In the epoch #30 of the Handwritten, Fashion and CIFAR-10, it is smaller than 0.1 for 99.4%, 92.7% and 95.2% of the input data, respectively. Figure 2 (d) reports the mean-difference for CIFAR-100, which requires more training steps than the other datasets. In the epoch #90 of the CIFAR-100, it is smaller than 0.018 for 94.7% of the input data. Note that the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

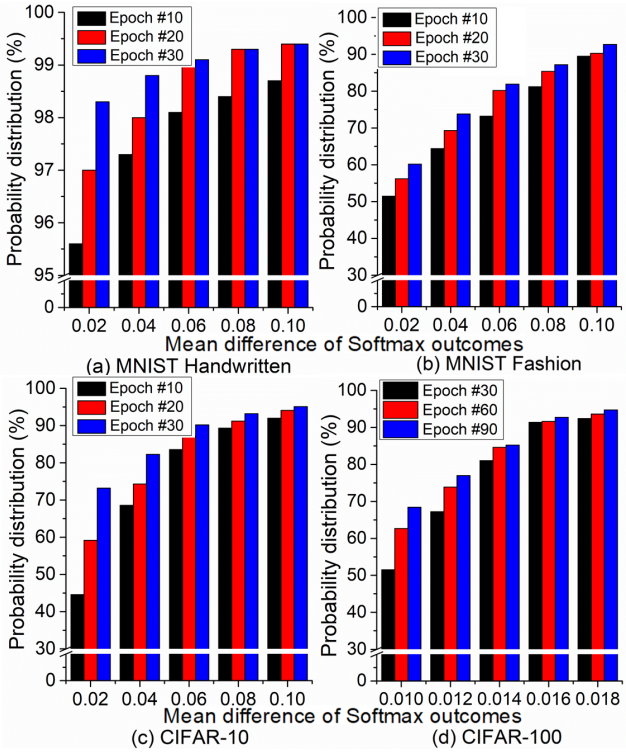

FIGURE 2. Mean difference distribution of Softmax outcomes for the same label inputs; (a) and (b) are for the MNIST dataset, (c) and (d) are for the CIFAR dataset.

mean difference in the x-axis of Figure 2 (d) is relatively small compared with the other datasets.

The similarity of the Softmax outcomes for the same labels means that the past result could be used in the current backpropagation. To take advantage of this behavior, we store the Softmax output per label when the forward propagation finishes. In supervised learning, the label is provided with the input data. Thus, it can be used to read the stored Softmax outcomes for inputs with the same label, and the backward propagation is initiated with the past stored data. Figure 3 shows an example of how the speculative backpropagation is performed in parallel when the input image and its label (a number 7) are provided as an input. At time *t(i)*, the forward propagation processes a current input image. At the same time, the stored data (*red* neurons in Figure 3) for the same label is used for the backpropagation.

Even though the speculated data ($y^{o'}$) is similar to the current Softmax output (y^{o}) in general, it is important to perform speculation more accurately because the difference directly affects the training performance. We found that Eq. (9) considering both the most recent forward outcome and the history of the forward computations helps reduce the difference. In other words, Eq. (9) accumulates all the Softmax outcomes so far. α and β are weights for the most recent outcome and the previous history, respectively. In general, setting α and β to 0.5 works well across all the datasets, and there is room for improvement by adjusting α and β as discussed in Section VII. In the MNIST, a more weight on the

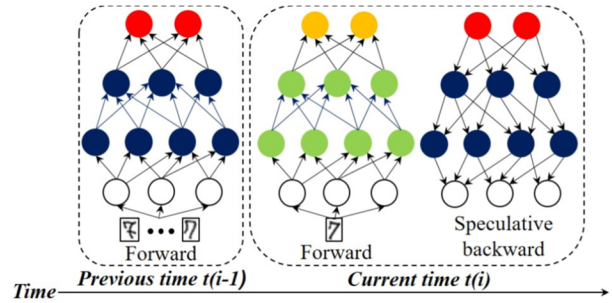

FIGURE 3. Parallel execution of forward and backward computations with speculative backpropagation, which uses the accumulated previous forward outcomes. The *red* neurons are accumulated Softmax outcomes until time *t(i-1)*, and the *blue* neurons are previous ReLU outcomes. These are used for speculative backpropagation at time *t(i)*, instead of using current forward outcomes (*yellow* and *green* neurons). Note that the weights in time *t(i)* are used to perform the speculative backpropagation.

accumulated Softmax outcome ($y^{o'}$) achieves roughly 0.2% smaller mean-difference (α = 1/3, β = 2/3). On the other hand, in the CIFAR, the mean-difference was roughly 0.4% smaller when giving more weight to the most recent Softmax outcome (α = 2/3, β = 1/3).

$$y^{o'} = (y^{o}\alpha) + (y^{o'}\beta) \; where \; (\alpha + \beta = 1) \tag{9}$$

y^{o} = current Softmax outcome, $y^{o'}$ = accumulated Softmax outcome

Nevertheless, there are still cases where there is a large difference between speculated and actual data. In these cases, we treat it as the wrong speculation, and perform the backpropagation again with the current Softmax outcome. More elaborately, the difference between the speculated and the actual data is computed before calculating Eq. (7) for the weight update. It is because the actual one is available at the end of the forward computation. If the difference is larger than a threshold, the speculated execution is nullified and the backpropagation is performed again with the actual data (*yellow* neurons in Figure 3). Thus, the performance of the speculative backpropagation is dependent upon the threshold, and Section VI reports the performance trade-off and sensitivity on thresholds.

B. SPECULATION OF RELU OUTCOMES IN HIDDEN LAYER

ReLU is one of the most widely used activation functions in ANN. It tends to create a sparse ANN, where some neurons are never activated [43]. We take it as an opportunity for parallel training. While training ANN, we found a characteristic that activated neurons in the past are more likely to be activated in the future for inputs with the same label. Table 1 - 4 show our experiment outcomes showing the probability that the same neurons are activated in the previous and current forward propagations, according to the MNIST and CIFAR datasets. As the training progresses, the activation probability of the same neurons increases. In the epoch #30 of the Handwritten, Fashion and CIFAR-10, the activation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

TABLE I
MNIST HANDWRITTEN (LENET)

Epochs	Activation probability of the same neurons			
	Conv. layer #1	Conv. layer #2	Conv. layer #3	Fully connected layer #1
10	81.5%	74.7%	82.1%	85.6%
20	82.8%	77.3%	82.4%	85.2%
30	84%	78.1%	85.9%	86.2%

TABLE 2
MNIST FASHION (LENET)

Epochs	Activation probability of the same neurons			
	Conv. layer #1	Conv. layer #2	Conv. layer #3	Fully connected layer #1
10	94.5%	86.9%	87.2%	85.9%
20	94%	84.4%	84.6%	83.2%
30	95.1%	85.1%	86.1%	84.1%

TABLE 3
CIFAR-10 (VGG16)

Epochs	Activation probability of the same neurons			
	Conv. layer #4	Conv. layer #8	Conv. layer #12	Fully connected layer #1
10	65.7%	70.2%	94%	80.1%
20	66.4%	78.8%	97.2%	84.2%
30	67.8%	78.9%	98.4%	84.6%

TABLE 4
CIFAR-100 (VGG16)

Epochs	Activation probability of the same neurons			
	Conv. layer #4	Conv. layer #8	Conv. layer #12	Fully connected layer #1
30	73%	70.2%	89.8%	80%
60	77.8%	72.5%	93.7%	86.2%
90	80.1%	72.4%	95%	88.8%

probabilities of the same neurons are 82.7%, 88.8%, and 83.7% on average, respectively for all convolution layers. In the epoch #90 of the CIFAR-100, the activation probability of the same neurons is 87.8% on average. The activation probabilities of the first few convolution layers for the CIFAR are relatively lower than the MNIST. It is because the data augmentation is applied only for CIFAR datasets, and it randomly changes the training input to increase the diversity of data available for training. In a fully connected layer, the activation probabilities of the same neurons are more than 80% for all datasets. This characteristic makes it possible to speculatively perform the backpropagation.

Equation (6) in the backpropagation computes the derivative of the activation function. Unlike other activation functions such as tangent and sigmoid, ReLU has a slightly different derivative property. The derivative of tangent and sigmoid requires a real number computed in the forward step. However, the derivative computation of the ReLU only demands whether the neuron is activated or not in the forward phase. Thus, if the binary result (activated or non-activated) of ReLU in the forward step can be speculated, its derivative in Eq. (6) can be performed simultaneously with the forward propagation. Based on this observation and the high activation probability of the same neurons, Eq. (6) can be computed speculatively. We store the ReLU outputs ($f(u_j^l)$) per label in the hidden layers when the forward propagation finishes. Then, the speculative backpropagation is performed using the stored values (*blue* neurons in Figure 3). As mentioned in Section IV-A, if the speculation turns out to be wrong, then it normally performs the backpropagation with the current ReLU outcomes (*green* neurons in Figure 3).

C. SPECULATION OF DERIVATIVE OUTCOMES IN POOLING LAYERS

There are two common pooling methods in CNNs; Max pooling and Average pooling. In the max pooling, the gradient is passed back to the next layer to the winning neuron in the forward pass. All the other neurons get zero gradients. Therefore, the position of the winning neuron should be predicted for the speculative backpropagation. In the average pooling, the output gradient is calculated by dividing the input gradient by the pooling size. Thus, it does not require any speculation. Our work uses the average pooling and leaves the max pooling as future work. Thus far, all the ingredients for the speculative backpropagation for the CNNs have been addressed.

V. IMPLEMENTATION AND OPTIMIZATION OF HW PARALLEL TRAINING

This section reports the implemented hardware design of LeNet with the proposed parallel training.

A. EXPERIMENTAL ENVIRONMENT

For prototyping the hardware implementation, we used a ZCU102 FPGA board, which is based on Zynq UltraScale+ MPSoC with a 4GB DDR4. The UltraScale+ is composed of the processing system (PS) and programmable logic (PL) sections: The PS has a quad-core Cortex-A53 processor operating at 1.5 GHz. The PL is configured with the hardware accelerator in our work. SDSoC 2019.1v [44], a CAD tool from Xilinx, is used for hardware and software implementation. It provides the capability of automating the system-level integration for C/C++/OpenCL code, targeting the Zynq programmable SoCs. The system-level integration includes the software-to-hardware translation, its device driver generation, and kernel creation; Users can specify software functions to be translated to hardware in SDSoC.

B. IMPLEMENTATION AND OPTIMIZATION

We implemented LeNet with the average pooling using the C language. The ReLU and Softmax are used in the hidden and output layers, respectively. We designed two C functions performing the forward propagation and the speculative backpropagation. Then, the C functions were translated to hardware by the SDSoC. Two directives are used to generate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

TABLE 5
RESOURCE UTILIZATION AND EXECUTION TIME OF HARDWARE ACCELERATORS ON ZYNQ ULTRASCALE+

Hardware Accelerators	*#BRAMs (18Kbit)*	*#DSPs (48E)*	*#FFs*	*#LUTs*	*Maximum Frequency*	*Execution Cycle at 100MHz*
Forward propagation	62 (3%)	31 (1%)	28,938 (5%)	32,713 (11%)	273MHz	4,465,266 cycles
Speculative backpropagation	206 (11%)	5 (~0%)	29,634 (5%)	26,795 (9%)	287MHz	3,278,366 cycles
Optimized speculative backpropagation	210 (11%)	8 (~0%)	31,402 (5%)	29,073 (10%)	274MHz	4,298,658 cycles

the target hardware: *#pragma async* to generate two different hardware operating in parallel and *#pragma pipeline* to create the pipelining for maximal throughput. The synthesized accelerator operates at 100MHz, and the S/W portion in the application is processed on the Cortex-A53 in the PS.

Table 5 shows the resource utilization and performance of implemented hardware. The forward computation takes roughly 44ms on UltraScale+, and there is a 1,186,900-cycle difference in the execution latency between the forward propagation and speculative backpropagation (4,465,266 and 3,278,366 cycles, respectively). The difference comes from the fact that the forward propagation performs more computations than the one in the backpropagation. Specifically, the forward propagation requires computations in Eq. (1) from the input layer to the first hidden layer, whereas there is no computation in the opposite direction in the backpropagation. Since the weight update requires both forward and backward outcomes as in Eq. (7), this difference makes the weight update wait for the forward computation to complete. Thus, the backpropagation hardware becomes idle for 1,186,900 cycles.

There is an opportunity to balance the latencies and consequently reduce the overall training time. All the weight updates in Eq. (7) require the forward propagation outcomes, except the weights between the input and the first hidden layers; Specifically, all weights are updated based on $\Delta w_{ij}^{l} = \delta_i^{l} \times y_j^{l-1}$ where y_j^{l-1} is the forward propagation outcome and δ_i^{l} is the backpropagation outcome. However, the weights between the input and the first hidden layer are updated based on $\Delta w_{ij}^{1} = \delta_i^{1} \times y_j^{0}$, where y_j^{0} is an input image.

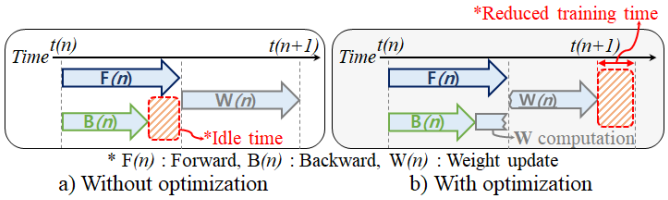

FIGURE 4. a) Without optimization, there is a latency imbalance between forward and backward processes, b) With optimization, it reduces the overall training time by balancing the forward and backward computations.

Thus, when the backpropagation is completed early, Δw_{ij}^{1} can be computed during otherwise idle time. Figure 4 compares without and with the optimized scheme. In Figure 4(a), the weight update computation is performed only after the forward propagation. In the optimized version shown in Figure 4(b), the weight update of the first layer is computed right after the backpropagation. The last row of Table 5 reports the optimized version and shows a similar execution latency to the forward computation, which reduces the overall training time.

VI. EVALUATION

This section compares the performance of the parallel training accelerator of LeNet against the baseline. The baseline is a training accelerator without speculation. The baseline accelerator was implemented with SDSoC as well. We also implemented the VGG16 using C language with OpenMP and CUDA framework for the parallel training of the CIFAR-10 and CIFAR-100, and evaluated their performance using a 32-core AMD 3970X based machine with 64GB main memory and GeForce 2080Ti GPU. We report the training performance according to the thresholds mentioned in Section IV-A.

Figure 5 reports the accuracies with speculative backpropagation according to thresholds. Figure 5(a) shows accuracies for the Handwritten. When the threshold is set to 0.1, the parallel training was more accurate than the baseline; the accuracy is 99.1%, which is even superior to the baseline

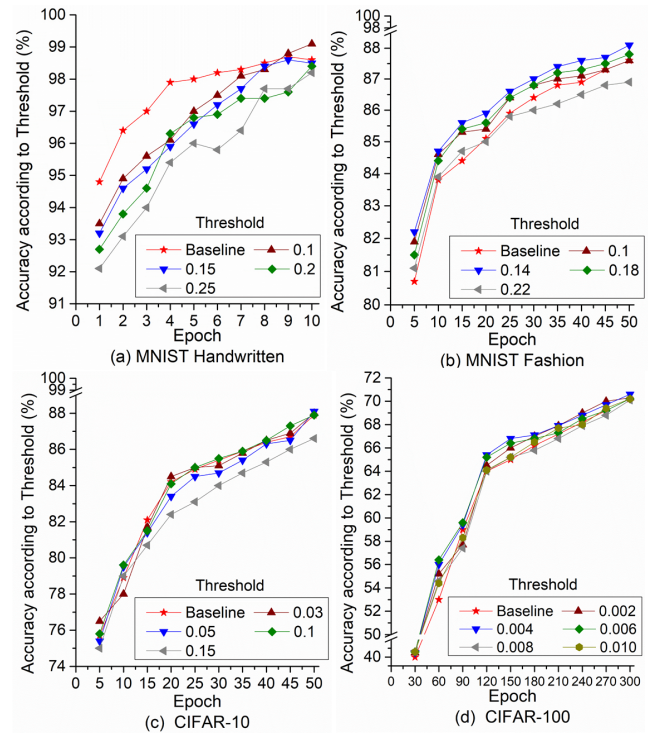

FIGURE 5. Accuracies with speculative backpropagation scheme according to thresholds. The threshold is the speculation allowance range, which is the difference between the speculated and the actual Softmax outcomes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

TABLE 6
MNIST HANDWRITTEN (LENET)

Thresholds	Miss speculation rate				
	Epoch #2	Epoch #4	Epoch #6	Epoch #8	Epoch #10
0.25	9.9%	6.3%	5.6%	3.8%	2.6%
0.2	10.8%	6.9%	5.9%	4.1%	3.1%
0.15	11.5%	7.3%	5.8%	4.2%	3.2%
0.1	13.7%	8.4%	6.5%	4.9%	4.1%

TABLE 7
MNIST FASHION (LENET)

Thresholds	Miss speculation rate				
	Epoch #10	Epoch #20	Epoch #30	Epoch #40	Epoch #50
0.22	26.6%	24.8%	23.1%	23.0%	22.3%
0.18	30.1%	29.6%	27.5%	26.5%	25.1%
0.14	35.3%	32%	31.7%	29.2%	27.7%
0.1	39.2%	36.8%	34.9%	32.1%	30%

TABLE 8
CIFAR-10 (VGG16)

Thresholds	Miss speculation rate				
	Epoch #10	Epoch #20	Epoch #30	Epoch #40	Epoch #50
0.15	8.4%	7.5%	5.7%	4.8%	2.8%
0.1	13.3%	11.1%	9%	6.4%	3.6%
0.05	28.5%	23.3%	17.9%	13.4%	9.4%
0.03	43.1%	36%	28.8%	21.7%	16.2%

TABLE 9
CIFAR-100 (VGG16)

Thresholds	Miss speculation rate				
	Epoch #60	Epoch #120	Epoch #180	Epoch #240	Epoch #300
0.01	8.9%	2.8%	2.2%	2.1%	2.1%
0.008	10.0%	3.3%	2.1%	2.3%	2.2%
0.006	12.1%	4.8%	3.4%	3.6%	3.4%
0.004	17.2%	7.6%	5.7%	6.0%	5.9%
0.002	30.0%	15.2%	12.2%	12.8%	12.6%

98.6% (0.5% improvements). Figure 5(b) shows the accuracies for the Fashion. With the threshold set to 0.14, there is also an accuracy enhancement by 0.5% to 88.1% compared with the baseline 87.6%. Figure 5(c) shows the training accuracies for the CIFAR-10. When the threshold is set to 0.05, the accuracy is improved by 0.2% to 88.1% compared with the baseline 87.9%. Figure 5(d) shows the training accuracies for the CIFAR-100. With the threshold set to 0.01, there is no accuracy loss compared with the baseline (70.2%). When the threshold is set to 0.004, the accuracy is even improved by 0.4% to 70.6%.

While a lower threshold improves the accuracy, it increases the number of miss speculations because the speculation allowance range becomes smaller. Tables 6-9 show the miss speculation rates according to thresholds. In Table 6 for Handwritten. the miss speculation rate in epoch #10 is 4.1%, with the threshold set to 0.1. In Table 7 for Fashion, the miss speculation rate in epoch #50 is 27.7%, with the threshold of 0.14. Table 8 is for CIFAR-10. When the threshold is set to 0.05, the miss speculation rate in epoch #50 is 9.4%. Table 9 is for CIFAR-100. With the threshold set to 0.004, the miss speculation rate in epoch #300 is 5.9%. As the epoch progresses, the miss speculation rate decreases in all datasets. This is because, as the network gets trained, the difference in Softmax outcomes for the same label becomes smaller, as shown in Figure 2.

Note that the miss speculation rates for CIFAR-10 and CIFAR-100 are smaller than MNIST fashion. We believe that it comes from the Adam optimization technique, which takes advantage of the momentum by using the moving average of the gradients instead of the gradient itself like stochastic gradient descent (SGD). The training with considering the previous gradients could lead to the smaller differences in Softmax outcomes for inputs with the same label. Compared with the training without Adam, it reduces the miss speculation rate by 7.1% and 8.9% with the threshold set to 0.1 and 0.004 for the CIFAR-10 and CIFAR-100, respectively.

The miss speculation rate in turn influences the execution time for the parallel training. It is because the proposed method performs the normal training upon a miss speculation. Figure 6 shows normalized execution times of the parallel training over the baseline according to thresholds. With the thresholds set to 0.25 and 0.22 for Handwritten and Fashion, the training times were reduced by 1.38x and 1.28x over the baseline, respectively. With the threshold set to 0.15 for CIFAR-10, the performance is enhanced by 1.34x over the baseline. When the threshold is set to 0.01 for CIFAR-100, the performance is improved by 1.35x over the baseline. The last

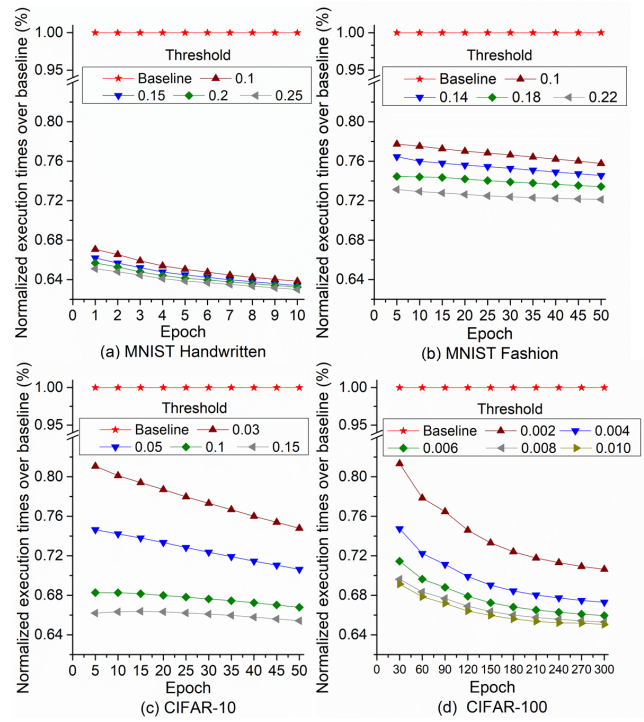

FIGURE 6. Normalized execution times of parallel training over baseline (baseline is the training without speculation)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

TABLE 10
OVERALL RESOURCE UTILIZATION AND POWER CONSUMPTION OF TRAINING ACCELERATORS ON ZYNQ ULTRASCALE+

Hardware Accelerators	*#BRAMs (18Kbit)*	*#DSPs (48E)*	*#FFs*	*#LUTs*	*Power Consumption*
Baseline accelerator	258 (28.34%)	36 (1.43%)	131,184 (23.93%)	91,756 (33.41%)	4.624W
Parallel accelerator	258 (28.34%)	36 (1.43%)	131,184 (23.93%)	91,841 (33.48%)	4.636W

epoch benefits comparably more from the speculative execution due to the lowest miss speculation rate in all datasets.

Table 10 shows the resource utilizations and power consumption of the hardware accelerators on Ultrascale+. In terms of the resource utilization, there is almost no difference between the parallel accelerator and the baseline. This is because the parallel accelerator is composed of roughly the same components as the baseline; Both accelerators should perform the two operations (forward and backward propagations) anyway. In tandem with the similar resource utilizations, the power consumption is also similar in both accelerators. Therefore, the parallel accelerator offers a better performance per watt than the baseline, due to its higher accuracy with a faster training speed. The amount of resources in Table 10 is a little higher than the one in Table 5. For example, the parallel accelerator utilizes a 23.9% of FFs and 33.4% of LUTs on Zynq UltraScale+. It is because the hardware components such as AXI interconnects and DMA engines are required for the full system integration and operation.

VII. DISCUSSION

There are some design considerations when architecting and designing a parallel training accelerator. The performance of the parallel accelerator depends on the speculation accuracy. It comes from the similarity of the Softmax and ReLU outcomes for the same label. The proposed equation (Eq. (9)) reflects the history of Softmax outcomes. In Equation (9), giving more weight to the accumulated Softmax means that it could reflect the history well but does not train sensitively with the recent input. Conversely, giving more weight to the current Softmax means that it could train sensitively with the recent input but the history would not be properly reflected. Therefore, we believe that the proper balancing is important. With setting both α and β to 0.5, the speculative backpropagation works well for all datasets. To find out additional room for improvement, we experimented with different pairs of α and β. Table 11-13 summarize the

TABLE 11
THE PARALLEL TRAINING PERFORMANCE ACCORDING TO DIFFERENT PAIRS OF α AND β FOR MNIST HANDWRITTEN: THE NORMALIZED EXECUTION TIMES AND ACCURACIES ARE FOR THE EPOCH #10 WITH THE THRESHOLD 0.1.

Hyperparameters value	$\alpha = 0.1$ $\beta = 0.9$	$\alpha = 0.2$ $\beta = 0.8$	$\alpha = 0.3$ $\beta = 0.7$	$\alpha = 0.4$ $\beta = 0.6$	$\alpha = 0.5$ $\beta = 0.5$	$\alpha = 0.6$ $\beta = 0.4$	$\alpha = 0.7$ $\beta = 0.3$	$\alpha = 0.8$ $\beta = 0.2$	$\alpha = 0.9$ $\beta = 0.1$	$\alpha = 1.0$ $\beta = 0.0$
Likelihood (<0.1)	93.2%	96.8%	98.5%	98.3%	98.3%	98.3%	98.3%	97.9%	97.7%	97.4%
Normalized execution times of parallel training over baseline	1.338x	1.353x	1.36x	1.358x	1.358x	1.359x	1.358x	1.356x	1.356x	1.354x
Accuracies with the validation set	98%	98.4%	99%	98.9%	99%	98.9%	98.9%	98.2%	97.7%	97%

TABLE 12
THE PARALLEL TRAINING PERFORMANCE ACCORDING TO DIFFERENT PAIRS OF α AND β FOR CIFAR-10: THE NORMALIZED EXECUTION TIMES AND ACCURACIES ARE FOR THE EPOCH #50 WITH THE THRESHOLD 0.1.

Hyperparameters value	$\alpha = 0.1$ $\beta = 0.9$	$\alpha = 0.2$ $\beta = 0.8$	$\alpha = 0.3$ $\beta = 0.7$	$\alpha = 0.4$ $\beta = 0.6$	$\alpha = 0.5$ $\beta = 0.5$	$\alpha = 0.6$ $\beta = 0.4$	$\alpha = 0.7$ $\beta = 0.3$	$\alpha = 0.8$ $\beta = 0.2$	$\alpha = 0.9$ $\beta = 0.1$	$\alpha = 1.0$ $\beta = 0.0$
Likelihood (<0.1)	81.9%	85%	89.6%	93.5%	93.6%	93.6%	94%	93.2%	93%	92.2%
Normalized execution times	1.285x	1.302x	1.312x	1.334x	1.335x	1.335x	1.336x	1.331x	1.33x	1.327x
Accuracies	87.6%	87.7%	87.8%	87.9%	88.1%	88%	88%	87.9%	87.4%	87%

TABLE 13
THE PARALLEL TRAINING PERFORMANCE ACCORDING TO DIFFERENT PAIRS OF α AND β FOR CIFAR-100: THE NORMALIZED EXECUTION TIMES AND ACCURACIES ARE FOR THE EPOCH #300 WITH THE THRESHOLD 0.01.

Hyperparameters value	$\alpha = 0.1$ $\beta = 0.9$	$\alpha = 0.2$ $\beta = 0.8$	$\alpha = 0.3$ $\beta = 0.7$	$\alpha = 0.4$ $\beta = 0.6$	$\alpha = 0.5$ $\beta = 0.5$	$\alpha = 0.6$ $\beta = 0.4$	$\alpha = 0.7$ $\beta = 0.3$	$\alpha = 0.8$ $\beta = 0.2$	$\alpha = 0.9$ $\beta = 0.1$	$\alpha = 1.0$ $\beta = 0.0$
Likelihood (<0.01)	94.1%	96.7%	97.6%	97.6%	97.8%	98%	98.1%	97.9%	97.8%	97.5%
Normalized execution times	1.33x	1.342x	1.346x	1.346x	1.348x	1.349x	1.35x	1.349x	1.347x	1.345x
Accuracies	69.7%	69.7%	69.8%	69.8%	70.1%	70.1%	70.2%	69.8%	69.6%	69.3%

* With $\alpha = 0$ and $\beta = 1$, almost all speculations fail because the accumulated Softmax outcome ($y^{o'}$) is not changed according to Eq. (9).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

performance differences. The likelihood in Tables means the probability that the mean-difference is less than 0.1 and 0.01. As shown, the performance differences are negligible according to different pairs of α and β ; The likelihood is the highest when α = 0.3 and β = 0.7 for MNIST and when α = 0.7 and β = 0.3 for CIFAR-10 and CIFAR-100. However, it reduces the training time roughly by 0.2% with the similar accuracies. Therefore, it would be optional to tune α and β for small improvements using optimization techniques such as Bayesian optimization.

Another important factor to consider is the trade-off between the training time and accuracy. It depends on the thresholds, i.e., speculation allowance range. If it is more sensitive to accuracy, the threshold should be set to a minimum. On the other hand, if sensitive to the training time, it should be set to a higher value. According to our experiments, the reciprocal of the number of classes in the dataset can be a good candidate for the threshold. Figure 5 and Figure 6 report the performance for the datasets. With the threshold of 0.1 for the datasets with 10 classes (MNIST and CIFAR-10), there are accuracy improvements by 0.5% and 0.1% for MNIST handwritten and CIFAR-10, respectively, and there is no accuracy loss in case of the MNIST fashion. The training times were reduced by 1.35x, 1.24x and 1.32x over the baseline for MNIST handwritten, fashion and CIFAR-10, respectively. With the threshold set to 0.01 for the dataset with 100 classes (CIFAR-100), it reduces the training time by 1.35x with no accuracy loss.

We found that the performance of the speculative backpropagation is closely related to the optimization techniques in training. In our experiments, when training VGG16 with Adam, the miss speculation rate was reduced by 7.1% and 8.9% with CIFAR-10 and CIFAR-100, respectively. It means that the optimization techniques can shorten the training time with less miss speculation. The parallel training also tends to show a better accuracy compared with the baseline. We believe that it is because the speculative backpropagation has an effect of regularizing the model. A common type of regularization is to inject noises during the training process. The noise is typically added to inputs, weights, gradients, and even activation functions. This kind of technique is frequently adopted in many applications for the performance improvement [45]. Our technique has a similar effect of adding noises to gradients and activation function by speculation.

Our proposed approach breaks the sequential flow of forward and backward propagations and enables the parallel execution. There are two approaches in implementing the parallel training: software-based training with GPUs and hardware-based training with the accelerator. When training with GPUs, our method requires higher power consumption and more memory at a certain point in time in return for the faster training. It is because the forward and backward passes are performed simultaneously. When implementing the hardware accelerator, the memory access for loading weights has a huge impact on the training time [46, 47]. For the fair comparison of the training time with the parallel accelerator, we implemented the baseline accelerator with forward and backward passes with separate BRAMs in hardware. Our method enables the efficient utilization of the forward and backward hardware resources. Nevertheless, the parallel accelerator requires some memory overhead because the Softmax outputs and 1-bit ReLU outputs should be stored per label. For example, it requires an additional 0.71MB of memory for CIFAR-10, which is only a 1.2% of the memory required for storing weights.

VIII. CONCLUSION

In this paper, we proposed a novel CNN parallel training architecture for image recognition. It takes advantage of the characteristics that the Softmax and ReLU outcomes in the forward propagation for the same labels are likely to be very similar. We implemented the parallel training accelerator in hardware on Zynq UltraScale+ and optimized it further for balancing the latencies for forward and backward computations. The speculative backpropagation shows different performance characteristics according to the thresholds, which incur trade-offs between the accuracy and the training speed. With the threshold set to 0.1 for the Handwritten, it shows a superior performance to the baseline, in both the training time (38% reduction) and the accuracy (0.5% improvement). With the threshold set to 0.14 for the Fashion, it also provides a noticeable performance in both the training time (26% reduction) and the accuracy (0.5% improvement). With the thresholds set to 0.1 for the CIFAR-10, the training time has been reduced by 33% without the accuracy loss. With the thresholds set to 0.004 for the CIFAR-100, it shows a performance enhancement to the baseline, in both the training time (34% reduction) and the accuracy (0.4% improvement). In the future, we plan to extend our work to other CNNs such as ResNet [48] and other datasets such as ImageNet [49]. We also plan to apply our method to transfer learning as well.

ACKNOWLEDGEMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00533, Research on CPU vulnerability detection and validation / No.2019-0-01343, Regional strategic Industry convergence security core talent training business).

REFERENCES

[1] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008: ACM, pp. 160-167.R.

[2] D. Claudiu Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," *arXiv preprint arXiv:1003.0358,* 2010.

[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on audio, speech, and language processing,* vol. 20, no. 1, pp. 30-42, 2011.

[4] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review,* vol. 33, no. 1-2, pp. 1-39, 2010.

[5] E. Xing and Q. Ho, "A new look at the system, algorithm and theory foundations of large-scale distributed machine learning," in *Tutorial in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15), Sydney, Australia*, 2015, pp. 10-13.

[6] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint arXiv:1202.2745,* 2012.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.

[8] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011: Omnipress, pp. 265-272.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[11] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1-9.

[12] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215-223.

[13] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. A. Gibson, and E. P. Xing, "On model parallelization and scheduling strategies for distributed machine learning," in *Advances in neural information processing systems*, 2014, pp. 2834-2842.

[14] J. K. Kim *et al.*, "STRADS: a distributed framework for scheduled model parallel machine learning," in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016: ACM, p. 5.

[15] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation* 2016, pp. 265-283.

[16] De Luca, P., Galletti, A., Giunta, G., & Marcellino, L. (2020, June). Accelerated Gaussian Convolution in a Data Assimilation Scenario. In *International Conference on Computational Science* (pp. 199-211). Springer, Cham.

[17] Fisher, M., & Gürol, S. (2017). Parallelization in the time dimension of four-dimensional variational data assimilation. *Quarterly Journal of the Royal Meteorological Society*, 143(703), 1136-1147.

[18] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595-2603.

[19] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693-701.

[20] A. Ignatov *et al.*, "Ai benchmark: Running deep neural networks on android smartphones," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 0-0.

[21] R. Merritt, "Startup Spins Whole Wafer for AI," 2019. [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1335043#.

[22] J. Dean *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223-1231.

[23] T. Chen *et al.*, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274,* 2015.

[24] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997,* 2014.

[25] S. Ahn, J. Kim, E. Lim, and S. Kang, ''Soft memory box: A virtual shared memory framework for fast deep neural network training in distributed high performance computing,'' *IEEE Access*, vol. 6, pp. 26493–26504, 2018.

[26] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016: ACM, pp. 26-35.

[27] A. L. Gaunt *et al.*, "AMPNet: asynchronous model-parallel training for dynamic neural networks," *arXiv preprint arXiv:1705.09786,* 2017.

[28] W. Vanderbauwhede and K. Benkrid, *High-performance computing using FPGAs*. Springer, 2013.

[29] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Computer Architecture News,* vol. 42, no. 3, pp. 13-24, 2014.

[30] M. Courbariaux and Y. Bengio, (Feb. 2016). "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1." [Online]. Available: https://arxiv.org/ abs/1602.02830.

[31] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160, 2016.*

[32] S. Fox, J. Faraone, D. Boland, K. Vissers and P. H. W. Leong, "Training Deep Neural Networks in Low-Precision with High Accuracy Using FPGAs," *2019 International Conference on Field-Programmable Technology (ICFPT)*, Tianjin, China, 2019, pp. 1-9.

[33] L. Yang, Z. He, and D. Fan, "A fully on-chip binarized convolutional neural network FPGA implementation with accurate inference," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED '18. New York, NY, USA: ACM, 2018, pp. 50:1–50:6.

[34] M. Jaderberg *et al.*, "Decoupled neural interfaces using synthetic gradients," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017: JMLR. org, pp. 1627-1635.

[35] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 1037-1045.

[36] Z. Huo, B. Gu, Q. Yang, and H. Huang, "Decoupled parallel backpropagation with convergence guarantee," *arXiv preprint arXiv:1804.10574,* 2018.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[38] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence,* vol. 39, no. 12, pp. 2481-2495, 2017.

[39] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400,* 2013.

[40] Y. L. a. C. Cortes. "MNIST handwritten digit database." http://yann.lecun.com/exdb/mnist/.

[41] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Citeseer, 2009.*

[42] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980, 2014.*

[43] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, 2013, vol. 30, no. 1, p. 3.

[44] V. Kathail, J. Hwang, W. Sun, Y. Chobe, T. Shui, and J. Carrillo, "Sdsoc: A higher-level programming environment for zynq soc and ultrascale+ mpsoc," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016: ACM, pp. 4-4.

[45] NOH, Hyeonwoo, et al. "Regularizing deep neural networks by noise: Its interpretation and optimization." In *Advances in Neural Information Processing Systems.* 2017. p. 5109-5118.

[46] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning", Proc. 19th Int. *Conf. Archit. Support Program. Lang. Oper. Syst*., pp. 269-284, 2014.

[47] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor", Proc. *42nd Annu. Int. Symp. Comput. Archit*., pp. 92-104, 2015.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[49] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. "Imagenet: A large-scale hierarchical image database." In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3040849, IEEE Access

Sangwoo Park received the B.S. degree in Computer Science and Engineering from Soongsil University, Seoul, South Korea, in 2017, and the M.S. degree in Computer Science and Engineering from Korea University, Seoul, South Korea, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering.

Taeweon Suh received the B.S. degree in Electrical Engineering from Korea University, Seoul, South Korea, in 1993, the M.S. degree in Electronics Engineering from Seoul National University in 1995, and the Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2006. He is currently a Professor with the Department of Computer Science and Engineering, Korea University.