Direct Gradient Calculation: Simple and Variation-Tolerant On-Chip Training Method for Neural Networks

Hyungyo Kim, Joon Hwang, Dongseok Kwon, Jangsaeng Kim, Min-Kyu Park, Jiseong Im, Byung-Gook Park, and Jong-Ho Lee*

On-chip training of neural networks (NNs) is regarded as a promising training method for neuromorphic systems with analog synaptic devices. Herein, a novel on-chip training method called direct gradient calculation (DGC) is proposed to substitute conventional backpropagation (BP). In this method, the gradients of a cost function with respect to the weights are calculated directly by sequentially applying a small temporal change to each weight and then measuring the change in cost value. DGC achieves a similar accuracy to that of BP while performing a handwritten digit classification task, validating its training feasibility. In particular, DGC can be applied to analog hardware-based convolutional NNs (CNNs), which is considered to be a challenging task, enabling appropriate on-chip training. A hybrid method is also proposed that efficiently combines DGC and BP for training CNNs, and the method achieves a similar accuracy to that of BP and DGC while enhancing the training speed. Furthermore, networks utilizing DGC maintain a higher level of accuracy than those using BP in the presence of variations in hardware (such as synaptic device conductance and neuron circuit component variations) while requiring fewer circuit components.

1. Introduction

Recently, neural networks (NNs) have demonstrated superior performance in various fields, such as classification tasks,[1] visual/auditory recognition,[2] and decision-making tasks,[3] and have exhibited the potential of broader applications in the real world.[4] With the increase in the demand for NNs in various fields, issues such as performance, power efficiency, footprint, and training speed of NN hardware, have become a major concern. In particular, power and area constraints are important issues in embedded systems and Internet of Things (IoT) edge computing applications. Various approaches have been proposed to enhance the hardware efficacy of NNs, including specifically designed accelerators and distributed computing methods for deep learning.[5–9] However, approaches based on conventional von Neumann architecture with separate memory and processors have fundamental limitations of consuming time and power for memory access—a “memory wall.”[9,10] To overcome this limitation, neuromorphic systems that use in-memory computing have emerged as promising candidates for next-generation NN hardware.[11] Several neuromorphic systems, such as IBM’s TrueNorth and Intel’s Loihi, that utilize a static random access memory (SRAM) cell as a synaptic device[12,13] have been proposed. However, utilizing an SRAM cell as a synaptic device has limitations, such as requirement of a large number of memory cells for multi-bit weight values and inefficient power consumption.[14] As an alternative hardware platform, neuromorphic systems with analog conductance synaptic devices, such as resistive random access memory,[15–17] phase change random access memory,[18,19] and charge-trap-based floating-gate metal–oxide–semiconductor field-effect transistors (FG-MOSFET) [20–23] have been widely researched to implement NNs with higher density, faster parallel analog computing, and lower power consumption.[14,24,25]

There are two methods to train neuromorphic systems: off-chip and on-chip. In off-chip training, weight values that are obtained through training using a software are transferred to synaptic devices. In comparison, weight values of synaptic devices are updated during runtime by additional monolithic circuitry in on-chip training.[26] The performance of off-chip training is reported to be vulnerable to variations in hardware, especially in neuromorphic systems that use analog conductance synaptic devices.[27] During the process of transferring the pre-acquired weight values to the synaptic devices, converted weight values can be significantly affected by both device-to-device and pulse-to-pulse variations.[28] In the case of on-chip training, the network adapts to the variations in hardware during the training phase. Therefore, on-chip training has drawn significant attention due to its application in neuromorphic systems composed of analog synaptic devices.[29] However, on-chip training that utilizes conventional...
backpropagation (BP) has several drawbacks. These include requirement for additional circuits and memory devices for the implementation of backward path to allow the propagation of error values, and the difficulty of designing the backward path in complex networks.[30] Furthermore, accuracy degradation due to large variations in hardware remains an unsolved challenge for BP.[28]

Therefore, we propose a novel on-chip training method for neuromorphic systems called direct gradient calculation (DGC) to avoid the inclusion of additional circuitry required for backward propagation of error values. DGC is a method that can obtain the gradient values of weights in hardware-based NNs. In this method, the update quantity of weights in the gradient descent method is calculated by sequentially applying a small change to each weight and then measuring the change in cost value. The scope and conditions in which DGC can be used are the same as those of BP. DGC achieves an equally high accuracy as BP with the following advantages: 1) Reduction in the area required for on-chip training of NNs by eliminating the backward path circuitry of BP. 2) Possibility of being used for on-chip training of diverse hardware-based NNs for which designing BP circuitry is challenging. 3) Stronger immunity to device variations than that of BP.

This article is organized as follows. Section 2 presents the underlying principle and algorithm details of DGC and provides the hardware implementation of the proposed training method. Section 3 provides verification of the training feasibility of DGC, an analysis of the network accuracy depending on the magnitude of a small change applied to the weights, verification of DGC on a deeper and more complex network, and the hardware variation tolerance of DGC compared with BP.

2. Algorithm Details and Hardware Implementation

2.1. Limitations of On-Chip BP

The gradient descent method updates the weight values of a network by calculating the gradient values of a cost function with respect to the weights with the aim of minimizing the cost value for a given training dataset.[31] For a cost function $C$, weight $W_i$, and learning rate $\eta$, the update quantity of the weight is calculated as follows:

$$\Delta W_i = -\eta \frac{dC}{dW_i}$$  \hspace{1cm} (1)

where $i$ represents the index of the specific weight among the $N$ total weights of the network.

For BP, the gradient values of each weight are calculated by propagating the error value $\delta$ backward using the chain rule. [32] However, there are two major issues when implementing BP in a hardware system. First, although on-chip BP training has stronger immunity to variations in hardware compared with that of off-chip training, variations in hardware and noise could still affect the accuracy of the network, because the update quantities pass through several sequential layers.[28] In the case of BP, it is assumed that the hardware implementation of equations, involving summation, multiplication, and other operations, such as activation functions, is ideal, and device-to-device uniformity is preserved. However, discrepancies between the ideal and the real hardware environment cannot be completely corrected by BP, because devices in the backward path also exhibit variations (not ideal) in their characteristics. Second, the backward path requires a considerable amount of hardware resources. The gradient values of each weight are obtained by multiplying the error value of the postsynaptic neuron and the activation value of the presynaptic neuron, which requires multiplier elements as many as the number of weights. Also, memory devices are required for each neuron to store the activation value, which are necessary to propagate the error values backward.[10]

2.2. DGC Method

We propose a novel method for obtaining the weight gradient values. With DGC, the weight gradient values are determined directly by measuring the rate of change in the cost value with respect to the change in weight value instead of propagating the error backward. Because DGC does not require a backward path or derivative function implementation, the above-mentioned problems related to BP can be overcome. The essence of DGC can be summarized as follows:

$$\frac{dC(\theta; W_i)}{dW_i} = \lim_{\delta W \rightarrow 0} \frac{\delta C(\theta; W_i)}{\delta W} \approx \frac{C(\theta; W_i + \delta W) - C(\theta; W_i)}{\delta W}$$  \hspace{1cm} (2)

where $\theta$ represents other parameters in the network except $W_i$. In this article, $\delta W$ is the temporal weight change for the gradient calculation, and $\Delta W$ is the weight update value. If the value of $\delta W$ is sufficiently small, the measured gradient value for each weight would be similar to that of the actual gradient value. This enables the weights to be updated toward the minimum cost value.

The complete sequence of learning a single training datum in a network with $N$ weights using BP and two DGC methods (using different updating processes) are presented in Figure 1. In Equation (2), if $\delta W$ is infinitesimally small and the update is simultaneously performed after calculating the gradients for all the weights, DGC will update all the weights identically to BP, as shown in Figure 1a,b. However, retaining all the calculated gradients of the weights requires a significant amount of memory space. To avoid this problem, we propose that each weight be updated one by one after calculating the gradient of that weight instead of simultaneously updating all the weights, as shown in Figure 1c. Updating weights simultaneously and one by one is referred to as DGC-S and DGC-O, respectively. The validation of updating the weight values one by one is presented in Section 3.1.

However, both DGCs need more time compared with that of BP to train an NN on real hardware. This is because the computation on the forward path in the case of DGCs has to be repeated more times than that of BP. Hence, DGC can be considered as an algorithm that enables on-chip training with a lower area requirement and realizes on-chip training of complex NNs (verified in Section 3.3) at the expense of the training time. Because the training mechanism of DGC completely differs from that of BP, the actual hardware of the whole circuitry would be necessary to
compare the exact training speed and power consumption of each method. However, a complete circuitry of neuromorphic systems with analog synaptic devices that utilize BP has not been developed yet. Therefore, a comparison between DGC and BP with the exact training speed and power consumption would be impossible in the current state. Instead, a qualitative comparison of these methods regarding circuit and memory requirements and training speed is shown in Table 1 using big-O notation. Because the neurons are placed at one side of the synapse array, the number of neurons is \( O(\sqrt{N}) \).

### 2.3. Hardware Implementation of DGC

In a neuromorphic system with analog synaptic devices, a single weight requires a differential pair of synaptic devices with a conductance of \( G_+ \) and \( G_- \) to represent both positive and negative weights, respectively.\(^\text{[13]}\) In Figure 2a, the odd and even rows of each output \( (V_i) \) represent \( G_+ \) and \( G_- \), respectively. To implement DGC, we propose a conventional \( n \)-channel FG-MOSFET operating in the ohmic region as a synaptic device. The conductance of this device is represented as follows

\[
G = K(V_{GS} - V_{TH})
\]

where \( K \) is the device parameter, \( V_{GS} \) is the external gate voltage, and \( V_{TH} \) is the threshold voltage. The hardware weight corresponding to the software weight in this architecture is as follows

\[
W = \frac{R_2R_{ref1}}{R_1}(G_+ - G_-)
\]

where \( R_1 \), \( R_2 \), and \( R_{ref1} \) are shown in Figure 2a. Each neuron circuit consists of three operational amplifiers: U1, U2, and U3, where U1 summates the currents from the \( G_- \) devices. Note that \( R_{ref2} \) should be sufficiently small to ensure that the output of U1 is unsaturated. Subsequently, U2 summates the currents from the \( G_+ \) devices and subtracts the output of U1 from it. The neuron circuit exhibits a hard tanh activation function due to the supply voltage limitations of U2. Finally, U3 reduces the voltage by a ratio of \( R_3/R_1 \). Because the reduced voltage is the input of the subsequent layer, \( R_3/R_1 \)

---

**Table 1. Comparison of training methods.**

| Method | Peripheral circuit requirement | Memory device requirement | Training speed |
|--------|--------------------------------|--------------------------|---------------|
| BP     | \( O(N) \)                     | \( O(\sqrt{N}) \)         | \( O(1) \)    |
| DGC-S  | \( O(\sqrt{N}) \)              | \( O(N) \)                | \( O(N) \)    |
| DGC-O  | \( O(\sqrt{N}) \)              | \( O(1) \)                | \( O(N) \)    |
Figure 2. Hardware implementation of DGC. a) Circuit diagram of a fully connected layer utilizing DGC. The column lines represent the output of the \( l \)–1th layer neurons (which enters the fully connected layer as input), whereas the row lines represent the output of the \( l \)th layer neurons. FG-MOSFETs are utilized as synaptic devices. b) Two weight change mechanisms in an FG-MOSFET synaptic device. The weight can be changed by changing the gate or threshold voltage. c) Time evolution of one weight value (with corresponding \( V_{GS} \) and \( V_{TH} \)) and the cost function value. The colored bars represent the same phases as in Figure 1. Weights are temporarily changed by the gate voltage change, whereas weights are permanently updated by modifying the threshold voltage. d) Cost value perturbation detector and weight update circuit schematic. The initial cost value (\( \delta C \)) is stored in \( C_1 \), whereas the perturbed cost value (\( \delta C \)) is stored in \( C_2 \). The resulting cost value change are shown in Figure 2c. The changes in \( V_{GS} \) and \( V_{TH} \) (corresponding to a single weight) and the resulting cost value change are shown in Figure 2c. Here, other weight values remain unchanged while updating the single weight. For convenience, it is assumed that the \( G_+ \) value of the weight is fixed, and only the \( G_- \) value is updated. In this case, a program pulse with a voltage of \( V_{PGM} \) is applied to the gate terminal of the target synaptic device to increase \( V_{TH} \) and the resulting cost value change are shown in Figure 2c. The update quantity of \( V_{TH} \) is proportional to \( \delta C/\delta W \). Contrastively, if the cost value changes when a positive \( \delta W \) is applied in the forward phase. During the updating phase, the weight values of the synaptic devices are updated through a permanent change in \( V_{TH} \). This is achieved by applying a program (or erase) pulse to the FG-MOSFETs with \( G_+ \) or \( G_- \) conductance while maintaining \( V_{GS} \) to a constant bias voltage during the forward path.\(^{[28]} \) The program (or erase) pulse stores or removes the charge in the floating gate, resulting in a permanent change in \( V_{TH} \).

Conversely, \( \delta W \) should be applied temporarily during the forward path in DGC. However, several problems arise if \( \delta W \) is controlled by changing \( V_{TH} \). First, applying a program (or erase) pulse to the device requires a considerable amount of power. Second, repeated program/erase pulses degrade the endurance of the device. Finally, the weight value should return to its original value after the gradient calculation. However, restoring the original \( V_{TH} \) value is a challenging task due to the asymmetric characteristic of \( V_{TH} \) modulation.\(^{[14]} \) These problems can be eliminated by changing \( V_{GS} \) instead of \( V_{TH} \), which is considered a more efficient way of applying \( \delta W \). The conductance (\( G \)) depends on \( V_{GS} \) and \( V_{TH} \) symmetrically, as shown in Equation (3). Thus, changing \( V_{GS} \) is equivalent to changing \( V_{TH} \) by the same amount (in terms of \( G \) change), as shown in Figure 2b by \( \circ \) and \( \circ \), respectively.

It should be noted that changing \( V_{GS} \) is considerably rapid compared with changing \( V_{TH} \) during gradient calculation. Applying a small change to the weight value can be implemented by simply adding a small voltage (\( \delta V \)) to the read bias (\( V_r \)) at the gate terminal of the target synaptic devices, marked as the thick, red terminals in Figure 2a. The changes in \( V_{GS} \) and \( V_{TH} \) (corresponding to a single weight) and the resulting cost value change are shown in Figure 2c. Here, other weight values remain unchanged while updating the single weight. For convenience, it is assumed that the \( G_+ \) value of the weight is fixed, and only the \( G_- \) value is updated. In this case, a program pulse with a voltage of \( V_{PGM} \) is applied to the gate terminal of the target synaptic device to increase \( V_{TH} \) and the resulting cost value change are shown in Figure 2c. The update quantity of \( V_{TH} \) is proportional to \( \delta C/\delta W \). Contrastively, if the cost value decreases with positive \( \delta W \), an erase pulse should be applied to the synaptic device to decrease \( V_{TH} \) and the weight value.

Figure 2d presents the cost value perturbation detector and the weight update circuit schematic. The initial cost value (\( C \)) is stored in \( C_1 \), whereas the perturbed cost value (\( C + \delta C \)) due to the applied \( \delta W \) is stored in \( C_2 \) by controlling the switches. The voltage subtractor calculates the cost value change (\( \delta C \)).
which is converted to an update pulse through a pulse width modulation (PWM) circuit.[35] As mentioned earlier, DGC-S requires a large number of memory devices to temporarily store the weight update values, whereas DGC-O does not. Note that a full hardware implementation of DGC requires a significantly reduced number of devices compared with BP. Especially, considerably fewer memory devices are required for DGC-O. Furthermore, DGC is more suitable for mini-batch learning. Every neuron requires as many memory devices as the mini-batch size in BP. In contrast, DGC only requires one additional capacitor (C3) for mini-batch learning, which stores the accumulated capacitor (C3) for mini-batch learning.

3. Results and Discussion

We designed a fully connected NN (FCNN) of size 64-64-10 and a convolutional NN (CNN) for the modified National Institute of Standards and Technology (MNIST) database classification task. A Python simulation was conducted to verify and analyze the proposed method. The training dataset for one epoch was composed of 60,000 training images, and the accuracy was evaluated using the ratio of correctly classified images out of 10,000 test images. The cross-entropy function was adopted as a cost function, and the networks were trained for 50 epochs with a mini-batch size of 100. The mini-batch gradient descent method was adopted for this study. Gradient descent methods with momentum, such as RMSProp or Adam,[36,37] were not used, as they require additional memory and circuitry in hardware. Furthermore, optimization techniques, such as dropout,[38] were not used due to their complexity when implemented with analog circuits. Optimal learning rates were obtained for each network utilizing each method, and time-based decay was adopted as the learning rate schedule. Each network had the same optimal learning rate for all methods, so the same learning rate was used for training. To reflect the limits of synaptic conductances and voltage values in the hardware, the absolute values of the weights were constrained to be less than or equal to $W_{\text{max}} = 1$, and the hard tanh function was adopted as the activation function in every layer.

3.1. Trainability Validation

The simulation was conducted to train FCNNs using three different training methods: BP, DGC-S, and DGC-O. The simulation was conducted with an identical initial weight value set for each method, and the simulation was repeated five times with different initial weight value sets. The training curves of BP, DGC-S, and DGC-O are presented in Figure 3. The accuracies and cost values were averaged along the epochs and indicated by symbols. Deviations of the accuracy and cost values are indicated by error bars. The different initial weight value sets of each simulation cause the network to converge to different local minima, leading to deviations in accuracy. The average accuracies of DGC-S and DGC-O (95.05% and 95.06%, respectively) were similar to that of BP (95.03%). DGC-S has almost the same convergence speed as BP in terms of epochs, because they have identical update sequences. However, in the early epochs, DGC-O achieves higher accuracy (lower cost) than that of BP, which means faster convergence. This is expected to be mainly due to the frequent updates in DGC-O. Frequent updates enable weights to be updated with a lowered cost, resulting in a faster convergence speed for DGC-O than that for BP and DGC-S.

Figure 4a shows the weight matrices of the first fully connected layer in the 50-epoch-trained networks utilizing each method. The matrices were obtained under one simulation condition out of five simulations with different sets of initial weight values. All three networks using each method started with the same set of initial weight values and eventually converged with similar accuracy but at different weight values. Both BP and DGC-S converged at similar but slightly different weight values, because $\delta W$ was set to 0.01, which is not infinitesimally small. However, because DGC-O has a different weight update process compared with that of BP and DGC-S, the weight values where DGC-O converges were significantly different from those of BP and DGC-S.

The true positive rate (TPR), positive predictive value (PPV), true negative rate (TNR), and modified confusion entropy (MCEN) were also measured for these three FCNNs using 10,000 test images.[39] Figure 4b,c shows that the values of the TPR and PPV of BP and DGC-S were different from those of DGC-O for some labels due to convergence at different weight values. However, as shown in Figure 4d, all three methods exhibit similar TNRs. Furthermore, as shown in Figure 4e, the three methods exhibit similar MCENs, which is an indicator of the overall performance of a classifier, as it considers the degree of confusion of an image with respect to another. Similarly, MCEN also verified that the three methods have an equal performance.

The feasibility of training with DGC using two different updating processes was verified by the results mentioned previously. However, as described in Section 2.2, DGC-O has the advantage of a lower memory requirement compared with that of DGC-S. In this article, we focus on the hardware implementation of DGC. Accordingly, we adopt the updating process of DGC-O for the general DGC method. From this point, DGC will be used to represent DGC-O.
3.2. Accuracy Dependency on the WCR

We define the term weight change ratio (WCR) as follows

\[ \text{WCR} = \frac{\delta W}{W_{\text{max}}} \]  

(5)

Figure 5 shows the accuracy of the FCNN after 50 epochs as a function of the WCR in DGC. For a WCR greater than 0.1, the accuracy significantly decreases, as the WCR increases. However, for a WCR less than 0.1, the accuracy of the network saturates, as the WCR decreases. This is because, referring to Equation (2), as \( \delta W \) decreases, the measured gradient value converges to the ideal value. However, in practical hardware implementations, extremely small values of \( \delta W \) would be indistinguishable from external noise. In addition, the cost value change \( (\delta C) \) could also be intractable due to the resolution limit of the cost value perturbation detector circuit. Therefore, we consider 0.01 as a reasonable WCR for practical applications of DGC. Accordingly, the default value of the WCR is set to 0.01 for the remaining simulations in this article.

3.3. DGC for CNNs

To verify the performance of DGC in deeper and more complex networks, we designed a CNN with the structure of a variant of LeNet-5,[40] as shown in Figure 6a. All convolutional layers had a stride of 1, and zero padding was not used. Here, we also propose a hybrid method that uses both BP and DGC. In a CNN that uses BP or DGC, all the weight gradients in the convolutional layers and fully connected layers are obtained by each method. However, in the hybrid method, the gradients of the weights in the convolutional layers are obtained by DGC, whereas the gradients of the weights in the fully connected layer are obtained by BP. The weights of the convolutional layers are sequentially updated by DGC, and then, the weights in the fully connected layer are simultaneously updated by BP during the updating phase of one image set, as shown in Figure 6b.

Three networks with the same structure were trained using BP, DGC, and the hybrid method. The simulation was conducted for five different initial weight value sets, and the cost values and accuracies were averaged along the epochs, as in Section 3.1. It can be observed from Figure 6c that the accuracies of DGC and the hybrid method are 98.04% and 97.98%, respectively, which
are similar to the accuracy of BP, i.e., 98.00%. It should be noted that DGC enables on-chip training in convolutional layers. This is remarkable, because an on-chip training method suitable for the convolutional layers of CNN is absent, to the best of our knowledge.

The FCNN in Section 3.1 achieves an accuracy of 95% using 4810 total weights, whereas the CNN achieves ≈3% higher accuracy using 2380 total weights. Thus, the CNN, which adopts DGC and is capable of on-chip training, can achieve high accuracy in edge devices that require small area occupancy. Furthermore, the hybrid method efficiently combines BP and DGC to enable on-chip training of CNNs while enhancing the training speed compared with that of standalone DGC. In the hybrid method, only convolutional layers, which are intractable in hardware that utilizes BP, can be trained by DGC.

Especially for CNN architectures with a small proportion of weights in the convolutional layers and a large proportion of weights in the fully connected layers (such as VGGNet), the hybrid method is expected to reduce the training time significantly, because the training time required by DGC is proportional to the number of weights. The hybrid method is also expected to be an efficient method of training a network in a reasonable time for large images (such as the CIFAR-10/100 dataset), providing a practical solution for on-chip training of hardware-based CNNs.

3.4. Hardware Variation Tolerance

In hardware-based systems, the physical parameters of the device, such as the width, length, thickness, doping concentration, and mobility, can randomly deviate from the ideal target values. Hence, the variations in $G$, $R_{\text{ref1}}$, and the voltage scaling gain described in Section 2.3 are modeled as a random normal distribution of $N(1, \sigma^2)$. The standard deviation ($\sigma$) of the random normal distribution varied from 0 to 0.4 in 0.1 increments in the simulation, and each method was simulated five times for one $\sigma$ value. Figure 7 demonstrates the dependency of the accuracy on hardware variations in 50-epoch-trained FCNNs utilizing BP and DGC. As the random variations increase, DGC exhibits a smaller performance degradation than BP. For $\sigma = 0.4$, the accuracies of networks utilizing BP and DGC decrease by 2.41% and 1.08%%, respectively. In the case of BP, variations in hardware affect the circuit in both forward and backward paths; whereas in DGC, the variation affects only the circuit in the forward path. As a result, DGC is approximately twice as tolerant to variation in hardware compared with that of BP.

4. Conclusion

We have proposed a novel on-chip training method for NNs that is simple and tolerant of variation in circuit implementation—DGC. Prior research on on-chip training, which adapts to a nonideal circuit environment during training, mostly utilizes BP as a training algorithm. However, BP requires an additional backward path, which increases circuit area, memory requirement, and complicates
the circuit design when implemented in hardware. Furthermore, accuracy degradation of hardware BP due to large variation in hardware needs to be solved. DGC, which can substitute conventional BP, is a method for obtaining the gradient values of weights in hardware-based NNs. Unlike BP, DGC uses the cost value change corresponding to the weight value change in the forward path to obtain the gradient values. Hence, it does not require a backward path. The validity of DGC, especially with updating weights one by one, was verified in the MNIST database classification example. The accuracy of DGC in the FCNN and CNN, 95.06% and 98.04%, is similar to that of BP, 95.03% and 98.00%, respectively. On-chip training in hardware CNNs is considerably difficult due to the complexity of the BP circuit design. Therefore, utilizing DGC is an appropriate approach. In addition, DGC is suitable for mini-batch learning compared with that of BP. Moreover, DGC shows greater tolerance to variations in hardware for FCNN compared with BP. In the case of a normal variation with a standard deviation of 0.4, the accuracy of DGC decreases by 1.08%p, whereas the accuracy of BP decreases by 2.41%p. The main issue related to DGC is the relatively slow training speed. If this issue can be resolved, DGC is expected to be an efficient on-chip training method for various types of NNs. As one of the solutions, a hybrid method that uses both DGC and BP for on-chip training CNNs can effectively reduce the training time compared with that of standalone DGC. Due to the simplicity of BP in fully connected layers compared with that of convolutional layers, BP circuitry of the hybrid method is significantly simpler than that of standalone BP. Thus, the hybrid method can efficiently make up for the shortcomings of DGC and BP. As CNNs have superior performance in various fields compared with that of other NNs, the results in this article enable neuromorphic systems that can achieve software-comparable performance with less chip area and power consumption.

Acknowledgements
H.K. and J.H. contributed equally to this work. This work was supported by the Technology Innovation Program (20009972) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea), and in the Brain Korea 21 Plus project in 2021.

Conflict of Interest
The authors declare no conflict of interest.

Data Availability Statement
Research data are not shared.

Keywords
analog synaptic devices, backpropagation, hardware-based neural networks, on-chip training

Received: April 5, 2021
Revised: May 27, 2021
Published online: July 5, 2021

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, in Communication of the ACM, Vol. 60, Association for Computing Machinery (ACM), New York 2017, p. 84.
[2] A. Graves, A. Mohamed, G. Hinton, in 2013 IEEE Int. Conf. Acoustics, Speech, and Signal Processing, IEEE, Piscataway, NJ, 2013, p. 6645.
[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Nature 2016, 529, 484.
[4] Y. LeCun, Y. Bengio, G. Hinton, Nature 2015, 521, 436.
[5] Y. Chen, T. Krishna, J. S. Emers, V. Sze, IEEE J. Solid-State Circuits 2017, 52, 127.
[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P-I. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Haggman, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, et al., in Proc. Int. Symp. Computer Architecture, Association for Computing Machinery (ACM), Toronto, 2017, p. 1.
[7] J. Chen, K. Li, K. Bilal, X. Zhou, K. Li, P. S. Yu, IEEE Trans. Parallel Distrib. Syst. 2019, 30, 965.
[8] J. Chen, K. Li, Q. Deng, K. Li, P. S. Yu, IEEE Trans. Ind. Inf. 2019, https://doi.org/10.1109/TII.2019.2909473.
[9] S. A. McKee, in Proc. 1st Conf. Comput. Front., Association for Computing Machinery (ACM), Ischia, 2004, p. 162.
[10] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, M. Horowitz, in Proc. 37th Annual Int. Symp. Computer Architecture, Association for Computing Machinery (ACM), New York, 2010, p. 37.
[11] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, Y. Xie, in Proc. 53rd Annual Design Automation Conf., Vol. 173, Association for Computing Machinery (ACM), Austin, 2016, p. 1.
[12] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Tabu, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, D. S. Modha, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 2015, 34, 1537.
[13] V. Srinivas, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaiakutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, A. Yang, H. Wang, IEEE Micro 2018, 38, 82.
[14] S. Mandal, A. El-Amin, K. Alexander, B. Rajendran, R. Jha, Sci. Rep. 2014, 4, 5333.
[15] K. Moon, E. Cha, D. Lee, J. Jang, J. Park, H. Hwang, in Proc. Int. Symp. VLSI Technology, Systems and Application, IEEE, Hsinchu, 2016, p. 1.
[16] B. H. Lee, H. Hwang, in IEDM Technical Digest, Washington, 2013.
[17] H. Y. Lee, P. S. Chen, T. Y. Wu, Y. S. Chen, C. C. Wang, P. J. Tzeng, C. H. Lin, F. Chen, M. J. Tsai. in 2008 IEEE Int. Electron Device Meeting, IEEE, San Francisco, 2008.
[18] G. W. Burr, R. M. Shelby, S. Sidler, C. di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayana, K. Virwani, E. U. Giacometti, B. N. Kurdi, H. Hwang, IEEE Trans. Electron Devices 2015, 62, 3498.
[19] D.-H. Kang, D.-H. Ahn, K.-B. Kim, J. F. Webb, K.-W. Yi, J. Appl. Phys. 2003, 23, 3536.
[20] S. Oh, C.-H. Kim, S. Lee, J. Kim, J.-H. Lee, Nanotechnology 2019, 30, 435206.
[21] Y.-T. Seo, M.-K. Park, J.-H. Bae, B. G. Park, J.-H. Lee, J. Nanosci. Nanotechnol. 2020, 20, 4292.
[22] J. Lu, S. Young, I. Arel, J. Holleman, IEEE J. Solid-State Circuits 2014, 50, 270.
[23] S. Brink, S. Nease, P. Hasler, S. Ramakrishnan, R. Wunderlich, A. Basu, B. Degnan, IEEE Trans. Biomed. Circuits Syst. 2012, 7, 71.
[24] C. Riggert, M. Ziegler, D. Schroeder, W. H. Krautschneider, H. Kohlstedt, Semicond. Sci. Technol. 2014, 29, 104011.
[25] D. Kuzum, S. Yu, H.-S. P. Wong, Nanotechnology 2013, 24, 382001.
[26] S. Yu, Proc. IEEE 2018, 106, 260.
[27] C. Merkel, D. Kudithipudi, in 2015 28th Int. Conf. VLSI Designs, IEEE, Bangalore, 2015.
[28] D. Kwon, S. Lim, J.-H. Bae, H. Kim, Y.-T. Seo, S. Oh, J. Kim, K. Yeom, B. G. Park, J.-H. Lee, Front. Neurosci. 2020, 14, 423.
[29] R. Hasan, T. M. Taha, C. Yakopcic, in 2017 Int. Joint Conf. Neural Networks, IEEE, Anchorage, 2017, p. 3527.
[30] S.-T. Lee, S. Lim, N. Choi, J.-H. Bae, D. Kwon, H.-S. Kim, B.-G. Park, J.-H. Lee, J. Nanosci. Nanotechnol. 2020, 20, 4138.
[31] N. Qian, Neural Netw. 1999, 12, 145.
[32] D. Rumelhart, G. Hinton, R. Williams, Nature 1986, 323, 533.
[33] S. Lim, J.-H. Bae, J.-H. Eum, S. Lee, C.-H. Kim, D. Kwon, B. G. Park, J.-H. Lee, Neural Comput. Appl. 2018, 31, 8101.
[34] P.-Y. Chen, B. Lin, I.-T. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J.-S. Seo, Y. Cao, S. Yu, in Proc. IEEE/ACM Int. Conf. Computer Aided Designs, IEEE, Austin 2015.
[35] S.-T. Lee, J.-H. Lee, Front. Neurosci. 2020, 14, 571292.
[36] T. Tieleman, G. Hinton, COURSERA: Neural Netw. Mach. Learn. 2012, 4, 26.
[37] D. P. Kingma, J. Ba, in 2015 Int. Conf. Learn. Represent., San Diego, CA, 2015, 434.
[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, J. Mach. Learn. Res. 2014, 15, 1929.
[39] R. Delgado, J. D. Núñez-González, PloS one 2019, 14.
[40] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Proc. IEEE 1998, 86, 2278.
[41] K. Simonyan, A. Zisserman, (Preprint) arXiv, arXiv:1409.1556, submitted: Apr., 2015.
[42] P. Narayanan, M. Leuchtenburg, J. Kina, P. Joshi, P. Panchapakesan, C. O. Chui, C. A. Moritz, in 2010 IEEE 25th Int. Symp. Defect Fault Tolerance VLSI Systems, IEEE, Kyoto, 2010, p. 24.