Master's Thesis of Engineering

Biologically Plausible Learning Algorithm without Weight Transport and Bidirectional Connection

가중치 전송문제와 양방향성 연결 문제를 해결한
뉴로모픽 학습 알고리즘

February 2023

Graduate School of Convergence Science and Technology
Seoul National University
Intelligence and Information Major

Sunghyeon Woo

Biologically Plausible Learning Algorithm without Weight Transport and Bidirectional Connection

가중치 전송문제와 양방향성 연결 문제를 해결한 뉴로모픽 학습 알고리즘

지도교수 전 동 석

이 논문을 공학석사 학위논문으로 제출함

2023 년 1 월

서울대학교 융합과학기술대학원

지능정보융합학과

우 승 현

우승현의 공학석사 학위논문을 인준함

2022 년 12 월

위 원 장: 박 재 흥 (인)

부위원장: 전 동 석 (인)

위 원: 곽 노 준 (인)

Abstract

Recently, the backpropagation is applied in various fields and has achieved great results. However, there are many limitations to do backpropagation in the actual brain. One of the reasons why backpropagation is difficult in the brain is the weight transport problem. In other words, The backpropagation constitutes a forward path and a symmetric feedback path, which is biologically impossible in the real brain. In a recent study, by constructing an asymmetric feedback path and learning similar to forward path, the weight transport problem is solved and high performance close to backpropagation is achieved. However, there is still a problem that biologically rare bidirectional connections occur between forward neurons and feedback neurons in order to train both forward and asymmetric feedback path. In this study, we propose a new learning algorithm that does not occur bidirectional connections while solving the weight transport problem. The proposed learning method also trains asymmetric feedback paths, but removes bidirectional connections by sharing activation across multiple layer during training both forward and asymmetric feedback path. In this algorithm, performance was significantly improved when compared with other learning algorithm that solved the weight transport problem. Furthermore, unlike the existing learning algorithm, it suggested the possibility that learning could be possible without accurate activation information. Consequently, it can reduce training memory because there is no need to store all accurate activation.

keywords: Deep Learning, Biologically Plausible Learning, Weight Transport Problem, Bidirectional Connection, Activation Sharing with Asymmetric Paths

student number: 2020-23071

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1. Introduction

Recently, deep learning has achieved performance comparable to or exceeding that of the human brain in various fields. However, it requires lots of resource and power to train deep and complex neural networks. For example, GPT-3 [1], which is state-of-art model of natural language processing, demands costly GPU clusters and a mount of GPUs consuming $1.1GWh$. On the other hands, the human brain consumes just $20W$ to train neural network consisting of 100 billon neurons and 100 trillion synapse [2]. For these reasons, research on an efficient biologically plausible learning rule that mimics the brain is attracting attention from academia.

Backpropagation(BP) [3], which is essential learning rule of deep learning, can train deep and complex neural networks successfully even outperforming human ability. However, it is hard to implemented in real biological neural systems for several reasons [4–7]. The *weight transport propblem* is one of the important reasons why backpropagation is biologically implausible [8]. In backpropagation, identical forward and feedback paths with same synaptic weights are required. However, it is difficult to be implemented in biological neural networks because vast amounts of information about synaptic weights must be transmitted very quickly along the axon [9].

By propagating feedback paths with random fixed weights, the Feedback Alignment (FA) [10] overcomes the weight transport problem. In feedback alignment, the feedback weights, which were initially different from forward weights, are aligned with the forward weights by training. Therefore, FA trained the network similarly to backpropagation where the forward weight and the feedback weight are identical. Direct Feedback Alignment (DFA) algorithm [11] also uses random fixed feedback weights, but the error from top layers is propagated directly to all layers including non-adjacent ones. In Direct Random Target Projection [12], the errors are locally created for each layer by propagating targets to random fixed weights. Although these

algorithms perform well on small networks, they degrade dramatically when used to more complicated networks, particularly deep convolutional neural networks [13, 14].

The other research suggests that instead of employing random fixed weights in the feedback path, the feedback weights could be trained like forward weights. Applying Reinforcement Learning [15] or Spiking Neural Network [16] to train feedback weights improves classification accuracy over above methods. However, such methods have only been experimented on shallow networks, and it is unclear if they might be used to train deeper neural networks. By transferring just the signs of the forward weights to the feedback path during training, Xiao et al. [17] achieves high performance close to backpropagation in deep neural networks. However, because some information of forward weight must still be transported, it is impossible to declare that the weight transport problem has been perfectly solved.

Weight Mirrors and modified Kolen-Pollack algorithms have recently been successful in training deep convolutional neural networks on big datasets with remarkable performance near to backpropagation by transporting only neuronal information, not explicit weight. These algorithms, however, need a *bidirectional connection*, which is a pair of connections between forward and feedback neurons to exchange neuronal information with each other bidirectionally. This connection could be found in some biological organisms [18–21]. However, general biological neural networks are consist of unidirectional connections between neurons and the opposite directional transfer of neuronal information is only possible by pass through multiple layers indirectly [22].

The goal of our paper is to achieve high performance close to backpropagation while solving the weight transport problem without bidirectional connection. Therefore, we propose Activation Sharing, which updates the weight of current layer using the input activation of preceding layer, while backpropagation uses input activation of current layer for weight updates. By employing this learning strategy on both forward and asymmetric feedback path, the weight transport problem can be solved without requiring bidirectional paths, and deep convolutional networks like ResNet-34 can

be effectively trained. Furthermore, because shared activation is repeatedly used for updating weights of several layers during training, the approach could considerably minimize memory consumption and be implemented efficiently on hardware.

Chapter 2. Related work

2.1 Backpropagation and weight transport problem

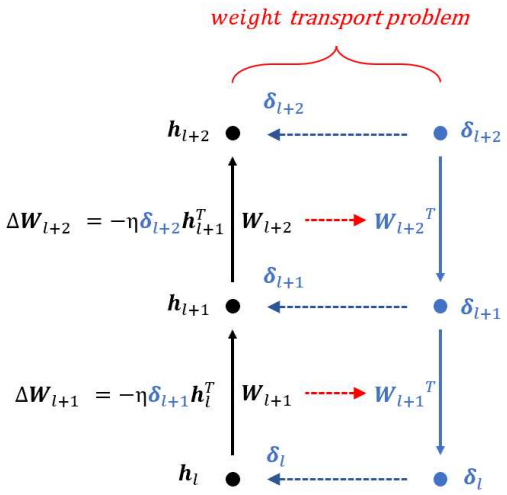


Fig. 2.1: Backpropagation

To understand the weight transport problem, we first explain backpropagation [3], a general training algorithm in deep learning. In forward path, the input data are propagated to multiple layers, resulting in output activations of layers:

$$\boldsymbol{h}_{l+1} = \phi(\boldsymbol{W}_{l+1}\,\boldsymbol{h}_l + \boldsymbol{b}_{l+1}) \tag{2.1}$$

where $\boldsymbol{h}_l$, $\boldsymbol{W}_l$, ϕ and $\boldsymbol{b}_l$ denotes the output activation, the weight matrix, nonlinear function like ReLU [23], and the bias vector of the hidden layer l in the forward path, respectively. When interpreted from a biological point of view, $\boldsymbol{h}$ could represent neural firing rates, $\boldsymbol{W}$ could be synaptic weights between neurons, $\boldsymbol{b}$ could be interpreted as bias currents, and ϕ could represent nonlinearities in neurons [24].

Total output of the network with loss function is compared with target and the errors are calculated in the direction of reducing total network loss. This errors are

propagated in backward path as follow:

$$\boldsymbol{\delta}_l = \phi^{'}(\boldsymbol{h}_l)\, \boldsymbol{W}_{l+1}^T\, \boldsymbol{\delta}_{l+1} \tag{2.2}$$

where $\boldsymbol{\delta}$ and $\phi^{'}$ represents the backpropagated error and the derivative of the nonlinear function ϕ, respectively.

Finally, the weights are updated by

$$\Delta \boldsymbol{W}_{l+1} = -\eta\, \boldsymbol{\delta}_{l+1}\, \boldsymbol{h}_l^T \tag{2.3}$$

where η is the learning rate. This backpropagation, which is based on mathematical gradient descent [25], trained the network successfully on various tasks. However, the weights of feedback path in equation (2.2) have to be equal to that of forward path in equation (2.1). Therefore, the errors must be propagated in the opposite direction of the forward path, which is not possible in biological systems with solely unidirectional paths. It is also biologically impossible a completely identical backward path that is separated from the forward path to exist like Fig. 2.1 because massive amounts of weight transport must occur very quickly between neurons to realize this [9]. This biological implausibility caused by the forward weight and the feedback weight being the equal is defined as a *weight transport problem*.

2.2 Training with random fixed feedback weights

To overcome weight transport problem, Lillicrap et al. [10] developed Feedback Alignment which propagates errors to random fixed weights, not synaptic weights equal to forward weights like backpropagation. The error propagation in Feedback Alignment is as follow:

$$\boldsymbol{\delta}_l = \phi^{'}(\boldsymbol{h}_l)\, \boldsymbol{B}_{l+1}^T\, \boldsymbol{\delta}_{l+1} \tag{2.4}$$

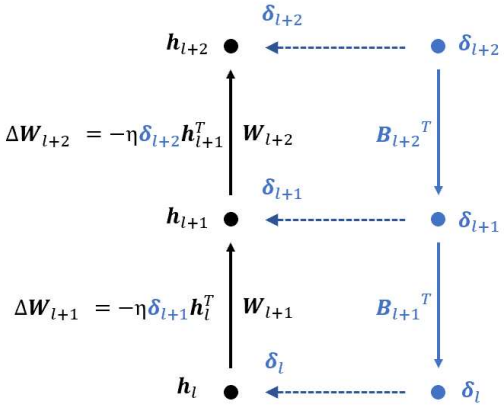


Fig. 2.2: Feedback Alignment

The B denotes random fixed weights in backward path. In this case, the weight transport problem is solved because there is no need to transport information of synaptic weights as shown in Fig.2.2. Lilicrap et al. explained that the synaptic weights in forward path are aligned to feedback weights during training, and therefore the training process of Feedback Alignment is close to that of backpropagation. They also proved that the loss of simple network converges to minimum value in Feedback Alignment by using Barbalat's lemma [26].

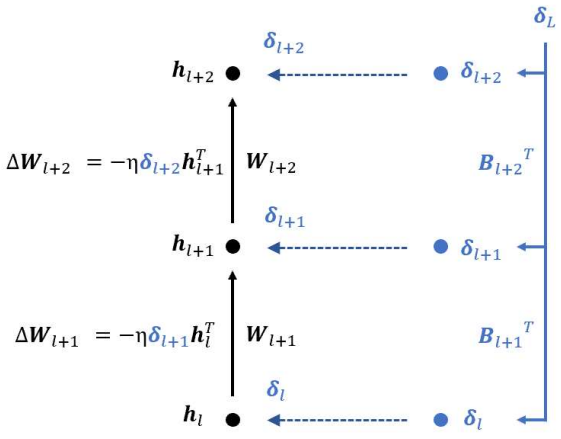


Fig. 2.3: Direct Feedback Alignment

By extending this idea, Direct Feedback Alignment [11], which also uses random fixed weights for backward propagation but propagates errors to lower layer directly is proposed. As shown in Fig.2.3, the back-propagated error is depicted as follow:

$$\boldsymbol{\delta}_l = \phi^{'}(\boldsymbol{h}_l)\, \boldsymbol{B}_{l+1}^{T}\, \boldsymbol{\delta}_L \tag{2.5}$$

where L is the total output of the network. The Direct Feedback Alignment not only solves weight transport problem, but also relaxes the structural constraint of biological neural networks that synaptic connections in feedback path should exist only between adjacent layers.

By employing random fixed weights in the feedback path, the approaches above completely address the weight transport problem and achieve high training accuracy in simple linear networks. However, they cause severe performance degradation in deep convolutional networks. [13, 14, 17, 27].

2.3 Training with feedback weight update

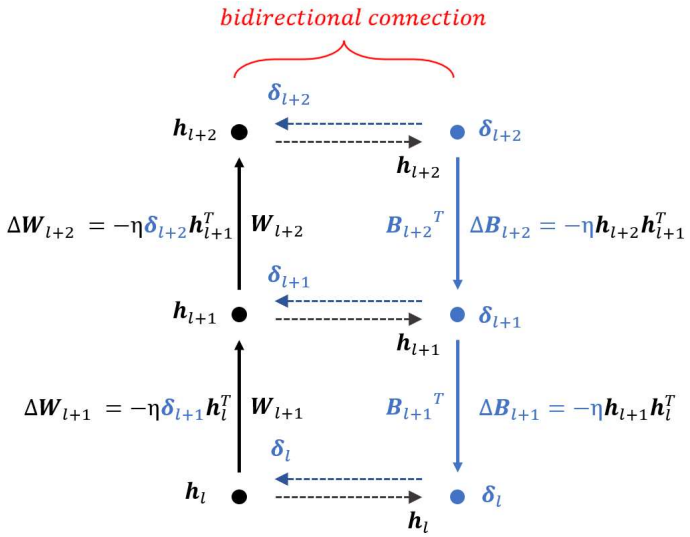


Fig. 2.4: Weight Mirror

Because employing random fixed weights in backward propagation results in poor

performance when training deep convolutional neural networks, many approaches for training the feedback weights have recently been presented. [15, 16, 28]. For example, Akrout et al. proposed the Weight Mirror (WM) and modified Kolen-Pollack (KP) algorithms which train the feedback weights and achieve high performance even in deep convolutional neural networks [24].

In Weight Mirror, the weight update in feedback path is as bellow:

$$\Delta \boldsymbol{B}_{l+1} = -\eta \, \boldsymbol{h}_{l+1} \, \boldsymbol{h}_l^T \tag{2.6}$$

The output activations of layers l and $l+1$ is used for calculating weight update $\boldsymbol{W}_{l+1}$ while the error propagation of Weihtg Mirror is equal to that of Feedback Alignment according to equation (2.4). Akrout et al. proved that $\Delta \boldsymbol{B}_{l+1}$ is proportional to W_{l+1}, and therefore the feedback weights $\boldsymbol{B}$ is aligned with the forward weights $\boldsymbol{W}$ although there is no weight transport.

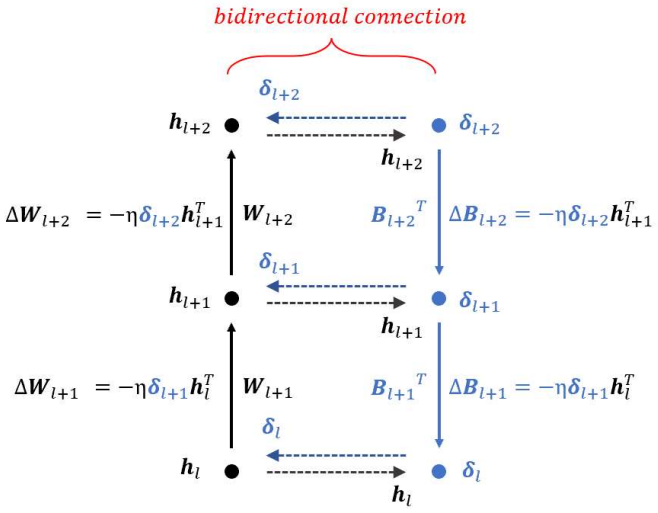


Fig. 2.5: Modified Kolen and Pollack

On the other hand, the modified Kolen and Pollack algorithm is depicted as follow:

$$\Delta \boldsymbol{B}_{l+1} = \Delta \boldsymbol{W}_{l+1} = -\eta \, \boldsymbol{\delta}_{l+1} \, \boldsymbol{h}_l^T \tag{2.7}$$

The forward and feedback weights could be updated in the same direction using this algorithm, and the feedback weight converges to forward weight during training.

Above methods perform close to backpropagation in deep convolutional neural networks on ImageNet. The weight transport problem is also solved by not directly transferring information about synaptic weights such as weight changes $\Delta \boldsymbol{W}$ and $\Delta \boldsymbol{B}$; instead, neuronal information in forward path ($\boldsymbol{h}$) and that of feedback path ($\boldsymbol{\delta}$) are exchanged as shown in Fig.2.5. However, they require a *bidirectional connection*, which is connection between forward and feedback neurons for transferring information bidirectionally (the connection between $\boldsymbol{h}$ and $\boldsymbol{\delta}$ in Fig.2.5).

2.4 Biologically implausible bidirectional connection

The bidirectional connections incur strict dependency between two neurons; one-to-one paring. This one-to-one paring would exist in some simple biological systems like C-elegans [18–21]. However, most of biological neural networks do not have this strict two-step dependencies, but have relaxed multi-step dependencies [29]. Furthermore, this strict two-step dependencies between two neurons would be a fatal structural constraint in constructing biological neural networks. To solve these issues, the object of our study is solving weight transport problem without bidirectional connections.

Chapter 3. Forward Alignment

we proposed Forward Alignment, which approximates forward path while prior studies only focus on approximating feedback path by random weights. In prior studies, the accurate information of activation is required to update both forward and feedback path. Therefore, this information has to be transported to feedback path and it causes bidirectional connection as shown in Fig. 2.5. By approximating forward path, we could relax the necessity for accurate neuronal information for updating weights and then remove bidirectional connection. In this section, we will first introduce the Forward Alignment algorithm. Afterwards, the validity of the Forward Alignment algorithm will be verified mathematically and experimentally.

3.1 Forward Alignment: Approximating Forward Path

For the convenience of explanation, suppose that a simple linear network is trained. In this case, we neglect nonlinear function ϕ in equation (2.1-2.3). For example, the equation (2.2) is transformed to $\boldsymbol{h}_{l+1} = \boldsymbol{W}_{l+1}\,\boldsymbol{h}_l$. To train this network, the error $\boldsymbol{\delta}$ and the activation $\boldsymbol{h}$ is required for updating the weights as shown in equation (2.3). In backpropagation, these values are calculated very accurately: the activations are obtained by propagating the input to layers in forward path and the errors are calculated by propagating the identical layers reversely as shown in Fig. (3.1a) and equations below:

$$\boldsymbol{h}_{l+1} = \boldsymbol{W}_{l+1}\,\boldsymbol{h}_l \tag{3.1}$$

$$\boldsymbol{\delta}_{l+2} = \boldsymbol{W}_{l+3}^{T}\,\boldsymbol{\delta}_{l+3} \tag{3.2}$$

$$\Delta\boldsymbol{W}_{l+2} = -\eta\,\boldsymbol{\delta}_{l+2}\,\boldsymbol{h}_{l+1}^{T} \tag{3.3}$$

As explained in Chapter 2, the equation (3.1-3.3) represent forward path, feedback path, and weight update, respectively. To solve weight transport problem, recent studies approximate feedback path (i.e. approximating equation (3.2)). For instance, approximate errors $\tilde{\boldsymbol{\delta}}$ is calculated by using random fixed weight $\boldsymbol{B}$ in Feedback Alignment as below:

$$\boldsymbol{h}_{l+1} = \boldsymbol{W}_{l+1}\,\boldsymbol{h}_l \tag{3.4}$$

$$\tilde{\boldsymbol{\delta}}_{l+2} = \boldsymbol{B}_{l+3}^T\,\tilde{\boldsymbol{\delta}}_{l+3} \neq \boldsymbol{\delta}_{l+2} \tag{3.5}$$

$$\Delta\boldsymbol{W}_{l+2} = -\eta\,\tilde{\boldsymbol{\delta}}_{l+2}\,\boldsymbol{h}_{l+1}^T \neq -\eta\,\boldsymbol{\delta}_{l+2}\,\boldsymbol{h}_{l+1}^T \tag{3.6}$$

Although the forward path is accurately calculated by equation (3.4) like backpropagation, the feedback path is approximated by using random fixed weight $\boldsymbol{B}$ instead of accurate value $\boldsymbol{W}$ as shown in equation (3.4) and Fig. (3.1b). Therefore, the weights are updated by approximated errors $\tilde{\boldsymbol{\delta}}$ in equation (3.6).

Approximate feedback path using random fixed feedback weights $\boldsymbol{B}$ might be utilized to train the network according to the Feedback Alignment. In the same way, we assumed the network might be trained even though we approximate forward path by using random fixed forward weight $\boldsymbol{C}$ instead of actual forward weights $\boldsymbol{W}$ (i.e. approximating equation (3.1)). In other words, approximate activations $\tilde{\boldsymbol{h}}$ also could be used for weight updates just as approximate approximate errors $\tilde{\boldsymbol{\delta}}$ are used for weight updates in Feedback Alignment. We named this algorithm *Forward Alignment*.

$$\boldsymbol{h}_{l+1} = \boldsymbol{W}_{l+1}\,\boldsymbol{h}_l \tag{3.7}$$

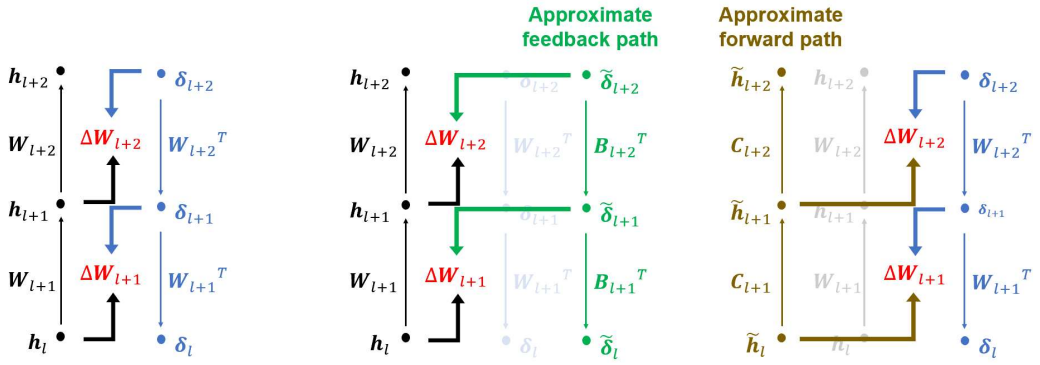
$$\tilde{\boldsymbol{h}}_{l+1} = \boldsymbol{C}_{l+1}\,\tilde{\boldsymbol{h}}_l \neq \boldsymbol{h}_{l+1} \tag{3.8}$$

$$\boldsymbol{\delta}_{l+2} = \boldsymbol{W}_{l+3}^T\,\boldsymbol{\delta}_{l+3} \tag{3.9}$$

$$\Delta\boldsymbol{W}_{l+2} = -\eta\,\boldsymbol{\delta}_{l+2}\,\tilde{\boldsymbol{h}}_{l+1}^T \neq -\eta\,\boldsymbol{\delta}_{l+2}\,\boldsymbol{h}_{l+1}^T \tag{3.10}$$

Note that the input propagates through forward weights $\boldsymbol{W}$ to make output of the network as equation (3.7). Simultaneously, approximate activations $\tilde{\boldsymbol{h}}$ are calculated by using random fixed forward weights $\boldsymbol{C}$ in equation (3.8). Afterwards, the errors are calculated accurately (equation (3.9)) and the weights are updated by using approximate activations $\tilde{\boldsymbol{h}}$ and errors $\boldsymbol{\delta}$ in equation (3.10).

To justify the proposed Forward Alignment, we will prove that the loss of a simple linear network converges to zero when learning through Forward Alignment in the next section. After that, we will verify the validity of this algorithm through experiments.



(a) Backpropagation (b) Feedback Alignment (c) Forward Alignment

Fig. 3.1: Overview of weight update process in different algorithms.

3.2 Mathematical proof of Forward Alignment

Lillicrap et al. used Barbalet's lemma for mathematical proof of Feedback Alignment [26]. Similarly, we also utilize Barbalet's lemma to prove our Foward Alignment also can converge the loss of total network to zero in some conditions. First, consider a simple linear network $\boldsymbol{h} = \boldsymbol{W}_1\boldsymbol{x}$, $\boldsymbol{y} = \boldsymbol{W}_2\boldsymbol{h}$ where the column vectors $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{h}$ denotes input, output, and output activation of hidden layer, respectively. In this case, n_x, n_y, and n_h denote dimension of $\boldsymbol{x}$, $\boldsymbol{h}$, and $\boldsymbol{y}$, respectively. $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ represent the weight matrices. The desired target function of the network is $\boldsymbol{T}$ where $\mathring{\boldsymbol{y}} = \boldsymbol{T}\boldsymbol{x}$. We apply the Forward Alignment to train above network by approximating activations

$\boldsymbol{h}$ to $\tilde{\boldsymbol{h}} = \boldsymbol{C}\,\boldsymbol{x}$, where $\boldsymbol{C}$ is random fixed forward weights weight.

3.2.1 Mathematical proof that loss converges to zero in Forward Alignment

We newly define linear target function as follow.

$$\boldsymbol{E} = \boldsymbol{T} - \boldsymbol{W}_2\boldsymbol{W}_1 \tag{3.11}$$

In this case, we can represent the error as $\boldsymbol{\delta} = \boldsymbol{E}\,\boldsymbol{x}$. The weight changes $\Delta\boldsymbol{W}_1$ and $\Delta\boldsymbol{W}_2$ are expressed as

$$\Delta\boldsymbol{W}_2 = \eta\,\boldsymbol{\delta}\,\tilde{\boldsymbol{h}}^T = \eta\,\boldsymbol{E}\,\boldsymbol{x}\,\boldsymbol{x}^T\,\boldsymbol{C}^T \tag{3.12}$$

$$\Delta\boldsymbol{W}_1 = \eta\,\boldsymbol{W}_2^T\boldsymbol{\delta}\,\boldsymbol{x}^T = \eta\,\boldsymbol{W}_2^T\boldsymbol{E}\,\boldsymbol{x}\,\boldsymbol{x}^T \tag{3.13}$$

where η represents the learning rate. We can assume that weight changes $\Delta\boldsymbol{W}_1$ and $\Delta\boldsymbol{W}_2$ could be expressed as expected value when weight change is converged to expected value after lots of training.

$$\Delta\boldsymbol{W}_2 = \eta\,[\boldsymbol{E}\,\boldsymbol{x}\,\boldsymbol{x}^T\,\boldsymbol{C}^T] = \eta\,\boldsymbol{E}\,\boldsymbol{C}^T \tag{3.14}$$

$$\Delta\boldsymbol{W}_1 = \eta\,[\boldsymbol{W}_2^T\boldsymbol{E}\,\boldsymbol{x}\,\boldsymbol{x}^T] = \eta\,\boldsymbol{W}_2^T\boldsymbol{E} \tag{3.15}$$

For sake of simplicity, we assume that the input $\boldsymbol{x}$ would be independent and identical distribution random variables whose mean and standard variation are 0 and 1, respectively. In this case, $[\boldsymbol{x}\boldsymbol{x}^T] = \boldsymbol{I}$ where $\boldsymbol{I}$ is the identity matrix. The discrete update process could be approximated to continuous time dynamics when a learning rate is very small as follow.

$$\dot{\boldsymbol{W}}_2 = \boldsymbol{E}\,\boldsymbol{C}^T \tag{3.16}$$

$$\dot{\boldsymbol{W}}_1 = \boldsymbol{W}_2^T \boldsymbol{E} \tag{3.17}$$

where $\dot{W}$ represents dW/dt. By merging equations (3.16) and (3.17),

$$\dot{\boldsymbol{W}}_1 \boldsymbol{C}^T = \boldsymbol{W}_2^T \boldsymbol{E} \boldsymbol{C}^T = \boldsymbol{W}_2^T \dot{\boldsymbol{W}}_2 \tag{3.18}$$

$$\boldsymbol{W}_1 \boldsymbol{C}^T = \int \boldsymbol{W}_2^T \dot{\boldsymbol{W}}_2 + \boldsymbol{r} = \frac{1}{2} \boldsymbol{W}_2^T \boldsymbol{W}_2 + \boldsymbol{r} \tag{3.19}$$

where $\boldsymbol{r}$ is a constant.

Theorem. *If the weight changes are calculated by*

$$\dot{\boldsymbol{W}}_2 = \boldsymbol{E}\, \boldsymbol{C}^T$$

$$\dot{\boldsymbol{W}}_1 = \boldsymbol{W}_2^T \boldsymbol{E}$$

where $\boldsymbol{r}$ in equation (3.19) is zero and left pseudoinverse matrix of $\boldsymbol{C}$ exists (i.e., $\boldsymbol{C}^+ \boldsymbol{C} = \boldsymbol{I}$), then a loss of the network converge to zero.

$$\lim_{t \to 0} \boldsymbol{E} = 0$$

In above theorem, $\boldsymbol{C}^+$ means the Moore-Penrose pseudoinverse of matrix $\boldsymbol{C}$. Because the theorem assumed that the condition $\boldsymbol{C}^+ \boldsymbol{C} = \boldsymbol{I}$ is satisfied, the components of $\boldsymbol{C}$ would be independent and $\boldsymbol{C}$ and the number of rows should be larger than that of columns (i.e., $n_h > n_x$). In addition, the condition $\boldsymbol{r} = 0$ means that $\boldsymbol{W}_1 \boldsymbol{C}^T = \frac{1}{2} \boldsymbol{W}_2^T \boldsymbol{W}_2$ by equation (3.19). By using these conditions and Barbalet's lemma below, the theorem could be proved mathematically.

Lemma 1 (Barbalat's Lemma)**.** *If the following conditions are satisfied:*

1. *M is lower bounded,*
2. *$\dot{M}$ satisfies negative semi-definite, and*
3. *$\dot{M}$ is uniformly continuous,*

then $\dot{M} \to 0$ when $t \to \infty$.

To utilize this lemma, we represent the matrix M as

$$M = tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}) \tag{3.20}$$

Before proving the theorem, we would prove that $\dot{M}$ converges to zero by confirming whether M satisfies three conditions of Barbalat's lemma. Afterwards, we prove the theorem that loss of the network ($\|\mathbf{E}\|$) converges to zero.

Condition 1. M is lower bounded.

Proof. $M = \|\boldsymbol{E}\boldsymbol{C}^T\|$ because $\boldsymbol{E}$ and $\boldsymbol{C}$ are not imaginary, but real valued. Therefore, M is lower bounded ($M \geq 0$). □

Condition 2. $\dot{M}$ satisfies negative semi-definite.

Proof.

$$\begin{aligned}
\dot{M} &= \frac{d}{dt}tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) \\
&= tr(\dot{\boldsymbol{E}}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) + tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\dot{\boldsymbol{E}}^T) \\
&= 2tr(\dot{\boldsymbol{E}}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) \\
&= 2tr((-\dot{\boldsymbol{W}}_2\boldsymbol{W}_1 - \boldsymbol{W}_2\dot{\boldsymbol{W}}_1)\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) \\
&= -2tr(\dot{\boldsymbol{W}}_2\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) - 2tr(\boldsymbol{W}_2\dot{\boldsymbol{W}}_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) \\
&= -2tr(\boldsymbol{E}\boldsymbol{C}^T W_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) - 2tr(\boldsymbol{W}_2\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T)
\end{aligned}$$

by applying equation (3.16) and (3.17). Then, the first term $2tr(\boldsymbol{E}\boldsymbol{C}^T W_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T)$ is calculated as follows.

$$
\begin{aligned}
& 2tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) \\
= \;& tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) + tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{W}_1^T\boldsymbol{C}\boldsymbol{E}^T) \\
= \;& \frac{1}{2}(tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) + tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T)) \\
= \;& tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T)
\end{aligned}
$$

by using the condition $\boldsymbol{W}_1\boldsymbol{C}^T = \frac{1}{2}\boldsymbol{W}_2^T\boldsymbol{W}_2$ where $\boldsymbol{r} = 0$ in equation (3.18). Consequently, we can achieve

$$
\begin{aligned}
& \frac{d}{dt}tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) \\
=& -2tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) - 2tr(\boldsymbol{W}_2\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T) \\
=& -tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) - 2tr(\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2) \qquad (3.21) \\
=& -\|\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\|^2 - 2\|\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\|^2 \qquad (3.22) \\
\leq \;& 0
\end{aligned}
$$

by applying the commutative law to second trace term. Therefore, $\dot{M}$ is negative semi-definite. □

Condition 3. $\dot{M}$ is uniformly continuous.

If the derivative function $\dot{f}$ is bounded, the function f is uniformly continuous. Therefore, we would prove the $\ddot{M}$ is finite to prove that the Condition 3. is satisfied. Before proving this, we first prove the condition W_2 is bounded because this condition is used to prove that $\ddot{M}$ is finite.

Condition 3.1. W_2 is bounded.

Proof. we define u as

$$u = tr(\boldsymbol{W}_2\,\boldsymbol{W}_2^T) \geq 0$$

Then,

$$\begin{aligned}
\dot{u} &= tr(\dot{\boldsymbol{W}}_2\boldsymbol{W}_2^T) + tr(\boldsymbol{W}_2\dot{\boldsymbol{W}}_2^T) \\
&= 2tr(\dot{\boldsymbol{W}}_2\boldsymbol{W}_2^T) \\
&= 2tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T) \\
&= 2tr((\boldsymbol{T} - \boldsymbol{W}_2\boldsymbol{W}_1)\boldsymbol{C}^T\boldsymbol{W}_2^T) \\
&= 2tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T) - 2tr(\boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{W}_2^T) \\
&= 2tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T) - 2tr(\boldsymbol{W}_2(\frac{1}{2}\boldsymbol{W}_2^T\boldsymbol{W}_2)\boldsymbol{W}_2^T) \\
&= 2tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T) - tr(\boldsymbol{W}_2\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{W}_2^T)
\end{aligned}$$

In this case, $\boldsymbol{W}_2\boldsymbol{W}_2^T$ is symmetric matrix which has an $n_o \times n_o$ dimension. Therefore. we can do diagonalization of $\boldsymbol{W}_2\boldsymbol{W}_2^T$. Therefore, if the dominant eigenvalue of this matrix is λ,

$$u = tr(\boldsymbol{W}_2\,\boldsymbol{W}_2^T) \leq n_o\lambda$$

By applying this inequality to second term of $\dot{u}$, we can attain

$$\begin{aligned}
tr(\boldsymbol{W}_2\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{W}_2^T) &= \|\boldsymbol{W}_2\boldsymbol{W}_2^T\|^2 \\
&\geq \lambda^2 \\
&\geq (\frac{u}{n_o})^2
\end{aligned}$$

Therefore, $\dot{u}$ is expressed as below.

$$\begin{aligned}
\dot{u} =& 2tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T) - tr(\boldsymbol{W}_2\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{W}_2^T) \\
\leq& 2tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T) - (\frac{u}{n_o})^2
\end{aligned}$$

We can apply Caushy-Schwarz to first term of above inequality as follow.

$$tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T)^2 \leq tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{T}^T)\, tr(\boldsymbol{W}_2^T\boldsymbol{W}_2) = u\|\boldsymbol{T}\boldsymbol{C}^T\|^2$$

Therefore, when $u > \|\boldsymbol{T}\boldsymbol{C}^T\|^2$, the above inequality is transformed to

$$tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T)^2 \leq u\|\boldsymbol{T}\boldsymbol{C}^T\|^2 \leq u^2.$$

In other words, $tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T) \leq u$. Therefore, we can achieve

$$\dot{u} \leq 2tr(\boldsymbol{T}\boldsymbol{C}^T\boldsymbol{W}_2^T) - (\frac{u}{n_o})^2 < 2u - \frac{u^2}{n_o^2}$$

In this inequality, we can confirm that $\dot{u} < 0$ when $u > 2n_o$. By merging above $u > \|\boldsymbol{T}\boldsymbol{C}^T\|^2$, we can attain

$$u \leq \boldsymbol{T}\boldsymbol{C}^T + 2n_o$$

at any time t. Therefore, $0 \leq u \leq \boldsymbol{T}\boldsymbol{C}^T + 2n_o$ which means $\boldsymbol{W}_2$ is bounded.

□

Condition 3.2. $\ddot{M}$ is finite.

Proof. Using equation (3.21) in Lemma 3, we can calculate $\ddot{M}$ as

$$\begin{aligned}
\ddot{M} =& \frac{d}{dt}[-tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) - 2tr(\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2)] \\
=& -2tr(\dot{\boldsymbol{E}}\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) - 2tr(\boldsymbol{E}\boldsymbol{C}^T\dot{\boldsymbol{W}}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) \\
& -4tr(\dot{\boldsymbol{W}}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2) - 4tr(\boldsymbol{W}_2^T\dot{\boldsymbol{E}}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2) \\
=& -2tr((-\boldsymbol{W}_2\dot{\boldsymbol{W}}_1 - \dot{\boldsymbol{W}}_2\boldsymbol{W}_1)\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}\boldsymbol{C}\boldsymbol{E}^T) - 2tr(\boldsymbol{E}\boldsymbol{C}^T\dot{\boldsymbol{W}}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) \\
& -4tr(\dot{\boldsymbol{W}}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2) - 4tr(\boldsymbol{W}_2^T(-\boldsymbol{W}_2\dot{\boldsymbol{W}}_1 - \dot{\boldsymbol{W}}_2\boldsymbol{W}_1)\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2) \\
=& 2tr(\boldsymbol{W}_2\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) + 2tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) \\
& -2tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2\boldsymbol{C}\boldsymbol{E}^T) - 4tr(\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2) \\
& +4tr(\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2) + 4tr(\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{W}_2)
\end{aligned}$$

Here, $\ddot{M}$ consists of $\boldsymbol{W}_2$, $\boldsymbol{C}$, $\boldsymbol{E}$, and $\boldsymbol{W}_1\boldsymbol{C}^T$. If all components of $\ddot{M}$ are bounded, we can prove that $\ddot{M}$ is bounded. First, $\boldsymbol{W}_2$ is bounded by Condition 3.1. Secondly, random matrix $\boldsymbol{C}$ is fixed in Forward Alignment during training. Third, $\boldsymbol{E}$ is also bounded because $M = tr(\boldsymbol{E}\boldsymbol{C}\boldsymbol{C}^T\boldsymbol{E})$ by equation (3.20) and M is bounded. The reason why M is bounded is that $M \geq 0$ by Condition 1 while $\dot{M} \leq 0$ by Condition 2. Finally, $\boldsymbol{W}_1\boldsymbol{C}^T$ is bounded because $\boldsymbol{W}_1\boldsymbol{C}^T = \frac{1}{2}\boldsymbol{W}_2^T\boldsymbol{W}_2$ by equation (3.19) and $\boldsymbol{W}_2$ is also bounded. Therefore, we can conclude that $\ddot{M}$ is finite. □

The M satisfies all conditions of Barbalat's lemma, and therefore $\dot{M} \rightarrow 0$ as $t \rightarrow \infty$. By using the equation (3.22) in proof of the Condition 2, we know that

$$\begin{aligned}\dot{\boldsymbol{M}} =& -\|\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T\|^2 - 2\|\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T\|^2 \\ =& \quad 0\end{aligned}$$

which means $\boldsymbol{E}\boldsymbol{C}^T\boldsymbol{W}_2^T = 0$ and $\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T = 0$. The equation $\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T = 0$ can be expressed as $\boldsymbol{W}_2^T\boldsymbol{E} = 0$ because we assumed that left pseudoinverse matrix of $\boldsymbol{C}$ exists. Therefore, $\dot{\boldsymbol{W}}_1 = \boldsymbol{W}_2^T\boldsymbol{E} = 0$, and therefore $\boldsymbol{W}_1$ is constant. Consequently, $\boldsymbol{W}_2^T\boldsymbol{W}_2$ is also zero by using $\boldsymbol{W}_1\boldsymbol{C}^T = \frac{1}{2}\boldsymbol{W}^T\boldsymbol{W}$ in equation (3.19). Moreover,

$$\boldsymbol{W}_2^T\boldsymbol{E}\boldsymbol{C}^T = \boldsymbol{W}_2^T(\boldsymbol{T} - \boldsymbol{W}_2\boldsymbol{W}_1)\boldsymbol{C}^T = 0 \tag{3.23}$$

The equation (3.19) could be expressed as $\boldsymbol{W}_2^T\boldsymbol{T}\boldsymbol{C}^T = \boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{C}^T$. In this case, the second term $\boldsymbol{W}_2^T\boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{C}^T$ is constant because the components of second term ($\boldsymbol{W}_2^T\boldsymbol{W}_2$, $\boldsymbol{W}_1$, and $\boldsymbol{C}^T$) are constant. Therefore, the second terms would become zero when differentiating this equation as below:

$$\dot{\boldsymbol{W}}_2^T\boldsymbol{T}\boldsymbol{C}^T = 0 \tag{3.24}$$

By substituting $\dot{\boldsymbol{W}}_2 = \boldsymbol{E}\,\boldsymbol{C}^T$ to above equation, we can get $\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{T}\boldsymbol{C}^T = 0$. If we apply left pseudoinverse of $\boldsymbol{C}$ to this equations, the equation is expressed as

$$0 = \boldsymbol{C}\boldsymbol{E}^T\boldsymbol{T}\boldsymbol{C}^T = \boldsymbol{C}^+\boldsymbol{C}\boldsymbol{E}^T\boldsymbol{T}\boldsymbol{C}^T(\boldsymbol{C}^+)^T = \boldsymbol{E}^T\boldsymbol{T} = \boldsymbol{T}^T\boldsymbol{E} \tag{3.25}$$

By applying above equation (3.25) and (3.11) to the loss $\|\boldsymbol{E}\|$, we can conclude

$$\begin{aligned}\|\boldsymbol{E}\| =& tr(\boldsymbol{E}^T\boldsymbol{E}) \\ =& tr(\boldsymbol{T}^T - \boldsymbol{W}_1^T\boldsymbol{W}_2^T)\boldsymbol{E}) \\ =& tr(\boldsymbol{T}^T\boldsymbol{E} - \boldsymbol{W}_1^T\boldsymbol{W}_2^T\boldsymbol{E}) \\ =& 0\end{aligned}$$

because $\boldsymbol{T}^T\boldsymbol{E} = 0$ and $\boldsymbol{W}_2^T\boldsymbol{E} = 0$. In conclusion, we successfully prove that the Forward Alignment can train the simple linear network in some conditions by converging loss $\boldsymbol{E}$ of the network to zero.

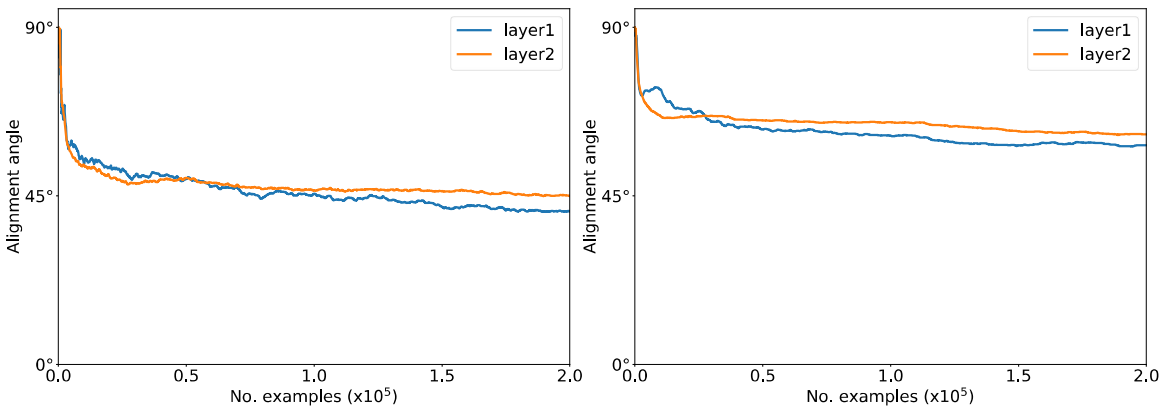
3.3 Experiments for Forward Alignment

We proved in the previous section that the Forward Alignment algorithm can train neural networks when some certain conditions are satisfied. In this section, we demonstrate the algorithm's ability to train neural network experimentally. We first confirm whether we can get a similar precision to BP by Forward Alignment, and whether the learning process of Forward Alignment is similar to that of backpropagation by checking that the forward weights and random fixed weights are aligned. We utilize two target networks with a structure of 10-10-10-10. First, we trained a linear network which consists of only fully-connected layer without batch normalization [30] and the nonlinear ReLU function [23]. Afterwards, a nonlinear network with batch normalization and ReLU is trained. We trained both networks for 200 epochs on the MNIST dataset [31]. The learning late and the batch size is set to 1e-4 and 50, respectively. The details about experiments like hyperparameters are provided in Appendix.

Table 3.1: Test Accuracy in Linear and Non-Linear Network[1]

Network	Linear		Non-Linear	
Learning Algorithm	FoA	BP	FoA	BP
Accuracy	90.93	92.39	93.03	95.01

The Forward Alignment achieves 90.93% and 93.03% when training linear network and nonlinear network, respectively. This results are close to that of backpropagation which attained 92.39% on linear network and 95.01% on nonlinear network. Furthermore, as shown in Fig. 3.2, the forward weights of the network are aligned with random fixed weights during training with Forward Alignment even if the network is nonlinear, just as Feedback Alignment aligns forward weights to random fixed feedback weights. When comparing Fig. 3.2a and Fig. 3.2b, it is confirmed that the alignment was worse in the nonlinear network than linear network. This observation is consistent with the result that the performance of Forward Alignment is more close to that of backpropagation in linear network than nonlinear network. In other words, learning process of Forward Alignment is similar to that of backpropagation when the forward weight is more aligned with a random fixed forward weight.



(a) linear network (b) nonlinear network

Fig. 3.2: Alignment Angle in Forward Alignment[2]

[1]FoA means Forward Alignment.

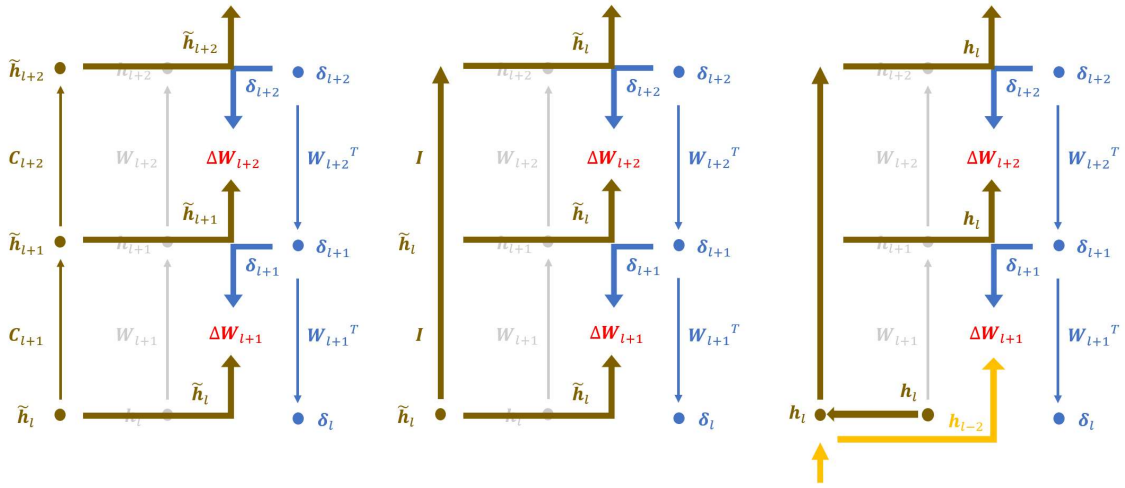
[2]Note that layer 1 and layer 2 represents the first 10×10 layer, and the second 10×10 layer, respectively.

Chapter 4. Activation Sharing for biological plausibility

In Chapter 3., we propose Forward Alignment which approximates forward path using random fixed weights. This learning algorithm achieves good performance in simple fully-connected networks. However, the Forward Alignment need additional computation compared to backpropagation and there is large performance degradation in deep convolutional neural networks like Feedback Alignment. In this section, we first upgrade Forward Alignment by using identity matrix as random fixed forward weights and dividing forward path into several blocks: Activation Sharing. Secondly, we solve the weight transport problem without bidirectional connections by applying Activation Sharing with Asymmetric Paths (ASAP). And then, we analysis the ASAP in terms of biological plausibility. Finally, we propose Activation Sharing in Feedback Paths (ASFP) which would be more biological plausible.

4.1 Activation Sharing: Extending Forward Alignment

In Forward Alignment, approximate forward path have to be calculated by equation (4.2) while it is unnecessary in backpropagation. To alleviate this problem, we use the identity matrix $\boldsymbol{I}$ as random fixed forward weights C as shown in Fig. 4.1b. In other words, the equation (3.4), which describes approximate forward path, is transformed to $\tilde{\boldsymbol{h}}_{l+1} = \tilde{\boldsymbol{h}}_l$. Therefore, additional computation for approximate forward path would be removed. Furthermore, we dived approximate forward path into several blocks as shown in Fig. 4.1c to improve performance. We assumed the reason why a performance is degraded in Forward Alignment is that the deviation between actual activation and approximate forward activation increases when input propagates through a lot of random fixed forward weights. In detail, the actual activation, which is used



(a) Forward Alignment (b) Using identity matrix (c) Dividing into blocks

Fig. 4.1: Advancement of Forward Alignment

in backpropagation, is $\boldsymbol{h}_l = \boldsymbol{W}_l \boldsymbol{h}_{l-1} = \cdots = \prod \boldsymbol{W}_k \, \boldsymbol{h}_0$ while the approximate activation is $\tilde{\boldsymbol{h}}_l = \boldsymbol{C}_l \, \tilde{\boldsymbol{h}}_{l-1} = \cdots = \prod \boldsymbol{C}_k \, \boldsymbol{h}_0$. Hence, the difference between $\boldsymbol{h}_l$ and $\tilde{\boldsymbol{h}}_l$ increases when l increase, and therefore the difference between learning direction of Forward Alignment and that of backpropagation also increase. To solve this problem, we constrained the input to propagate only a few layers within the block, rather than propagating all layers in the approximate forward path. For example, if there is two layer inside a block, the approximate activation is expressed as $\tilde{\boldsymbol{h}}_l = \boldsymbol{C}_l \, \boldsymbol{C}_{l-1} \, \boldsymbol{h}_{l-2}$ while the actual activation is $\boldsymbol{h}_l = \boldsymbol{W}_l \, \boldsymbol{W}_{l-1} \, \boldsymbol{h}_{l-2}$ in backpropagation. Consequently, the deviation of activation between Forward Alignment and backpropagation would decrease, and therefore the learning direction of Forward Alignment would be more close to that of backpropagation. We propose *Activation Sharing* based on these two intuitions as below:

$$\tilde{\boldsymbol{h}}_{l+1} = \boldsymbol{C}_{l+1} \, \tilde{\boldsymbol{h}}_l = \boldsymbol{h}_l \tag{4.1}$$

$$\tilde{\boldsymbol{h}}_{l+2} = \boldsymbol{C}_{l+2} \, \tilde{\boldsymbol{h}}_{l+1} = \boldsymbol{C}_{l+2} \, \boldsymbol{C}_{l+1} \, \tilde{\boldsymbol{h}}_l = \boldsymbol{h}_l \tag{4.2}$$

$$\Delta \boldsymbol{W}_{l+2} = -\eta \, \boldsymbol{\delta}_{l+2} \, \tilde{\boldsymbol{h}}_{l+1}^T = -\eta \, \boldsymbol{\delta}_{l+2} \, \boldsymbol{h}_l^T \tag{4.3}$$

$$\Delta \boldsymbol{W}_{l+3} = -\eta\, \boldsymbol{\delta}_{l+3}\, \tilde{\boldsymbol{h}}_{l+2}^{T} = -\eta\, \boldsymbol{\delta}_{l+3}\, \boldsymbol{h}_{l}^{T} \tag{4.4}$$

where there are two layers in a block as shown in Fig. 4.2. The weights $\boldsymbol{W}_{l+2}$ and $\boldsymbol{W}_{l+3}$ in a block are updated by activation $\boldsymbol{h}_l$ as shown in equation (4.3) and equation (4.4). We named the activation $\boldsymbol{h}_l$, which is used to update the weights in the block, as *shared activation*. In other words, if shared activations are decided, it is repeatedly used for updating weights in the blocks. The shared activation is determined as output activation of previous block. In Activation Sharing, the number of layers in a block could be changed. We named this number as block size k.

Activation Sharing can reduce computational cost and would train the network in a direction more similar to backpropagation. However, just applying Activation Sharing to forward path would incur weight transport problem as depicted in Fig. 4.2. In next section, we will deal with this problem.

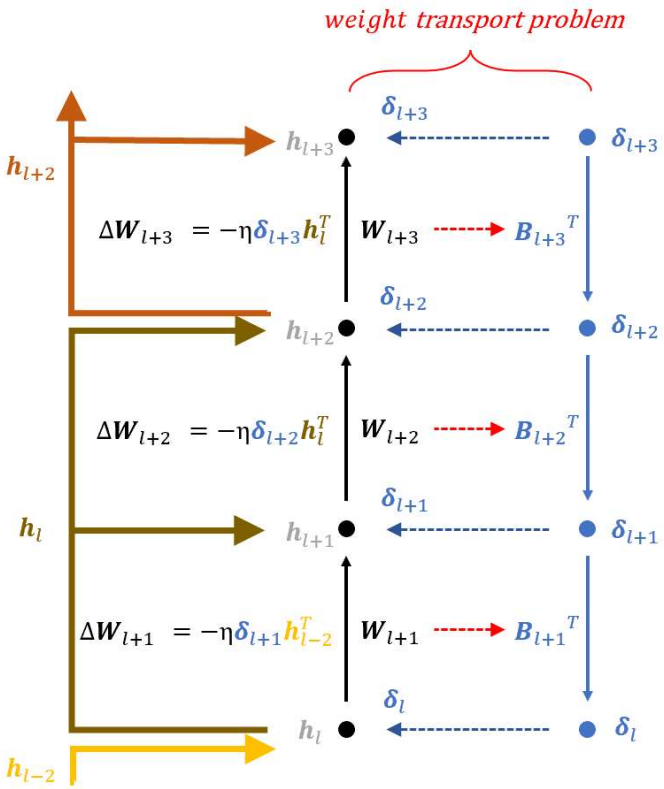


Fig. 4.2: Activation Sharing (k=2)

4.2 Activation Sharing with Asymmetric Paths: ASAP

4.2.1 ASAP can solve weight transport problem without bidirectional connections

To solve weight transport problem in Fig. 4.2, we used the theorem about weight changes proposed by Kolen and Pollack. They consider the network which has forward and asymmetric feedback path. In this network, the initial value of forward weight $\boldsymbol{W}$ is different from that of feedback weight $\boldsymbol{B}$. Both paths are updated by gradient descent with weight decay [32].

Theorem. *If the weight changes of both forward path and feedback path except for weight decay term are equal during training,*

$$\Delta \boldsymbol{W}(t) = \boldsymbol{R}(t) - \lambda \boldsymbol{W}(t)$$

$$\Delta \boldsymbol{B}(t) = \boldsymbol{R}(t) - \lambda \boldsymbol{B}(t)$$

then the network converges to symmetric networks (i.e., $\boldsymbol{W} = \boldsymbol{B}$*).*

Proof.

$$\begin{aligned} & \boldsymbol{W}(t+1) - \boldsymbol{B}(t+1) \\ = \; & (\boldsymbol{W}(t) + \Delta \boldsymbol{W}(t)) - (\boldsymbol{B}(t) + \Delta \boldsymbol{B}(t)) \\ = \; & (1-\lambda)(\boldsymbol{W}(t) - \boldsymbol{B}(t)) \\ = \; & (1-\lambda)^{t+1}(\boldsymbol{W}(0) - \boldsymbol{B}(0)) \end{aligned}$$

Therefore, the $\boldsymbol{W}(t+1) - \boldsymbol{B}(t+1)$ would converge to zero as the number of training t is increased. In other words, the asymmetric network converges to symmetric network ($\boldsymbol{W} = \boldsymbol{B}$) even though initial values of both paths are different. □

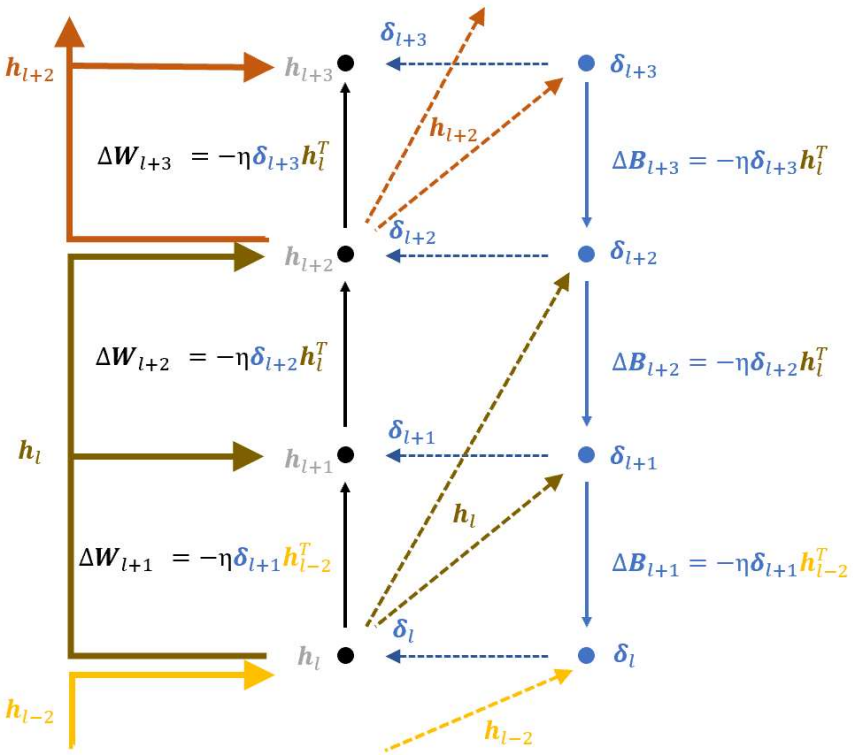


Fig. 4.3: Activation Sharing with Asymmetric Paths (k=2)

By using above theorem, the Activation Sharing in Fig. 4.2 with symmetric path can be transformed the network which has asymmetric paths ($\boldsymbol{W} \neq \boldsymbol{B}$) with same weight changes ($\Delta \boldsymbol{W} = \Delta \boldsymbol{B}$). The detail training process are as follow:

$$\Delta \boldsymbol{W}_{l+2} = \Delta \boldsymbol{B}_{l+2} = -\eta\, \boldsymbol{\delta}_{l+2}\, \boldsymbol{h}_l^T \tag{4.5}$$

$$\Delta \boldsymbol{W}_{l+3} = \Delta \boldsymbol{B}_{l+3} = -\eta\, \boldsymbol{\delta}_{l+3}\, \boldsymbol{h}_l^T \tag{4.6}$$

We named this algorithm as *Activation Sharing with asymmetric paths* (ASAP) because the shared activation $\boldsymbol{h}_l$ is shared to both forward path and asymmetric feedback path for updating weights as shown in Fig. 4.3. The ASAP not only takes the advantages about computational cost and backpropagation-like learning as mentioned in Section 4.1, but also solves weight transport problem without bidirectional connec-

tions as shown in Fig. 4.3.

In this section, we proposed Activation Sharing with Asymmetric Paths which apply Activation Sharing to forward and feedback paths. This algorithm solves weight transport problem without bidirectional connections successfully. However, some issues remain to ensure biological plausibility of the algorithm. In next section, we will discuss this issues.

4.2.2 Structural constraints relaxation by ASAP

The ASAP removes bidirectional connections while other algorithms like Weight Mirror and Modified Kolen-Pollack algorithm [24] need them. As explained in Section. 2.4, this bidirectional connections incur strict structural constraint of biological neural networks [18, 21]. Therefore, our ASAP can relax structural constraints of biological systems by removing bidirectional connections, and therefore support general and diverse biological neural networks. Although the ASAP relaxes structural constraints in biological neural networks, some constraints for constructing ASAP exist: the multi-step dependency between forward and feedback path like $\boldsymbol{h}_l \rightarrow \boldsymbol{\delta}_{l+1} \rightarrow \boldsymbol{\delta}_l \rightarrow \boldsymbol{h}_l$ in Fig. 4.3. However, this multi-step dependencies through intra and inter laminar routes are frequently found in biological systems while bidirectional connection is hardly found [29]. For example, the feedforward neurons of layer 4 neurons are connected to layer 3A via layer 6 indirectly in the visual cortex; can't directly be connected with layer 3A directly [29].

The ASAP could form spatially asymmetric connections as depicted in Fig. 4.3 because strict structural constraint which require symmetric one-to-one paring is alleviated. Therefore, it can support the spatially asymmetric biological neural networks, which are frequently found in biological organism. For instance, terminal arborization of axon by feedforward connections is concentrated while that of axon by feedback connections is diffused asymmetrically [29, 33, 34] in visual cortices. Furthermore, neurons of areas 35 and 36 project to other cortical regions while they do not receive

neuronal connections from these perirhinal regions [34, 35]. This asymmetry in biological systems can be supported by ASAP, whereas previous algorithms with bidirectional connections cannot support this.

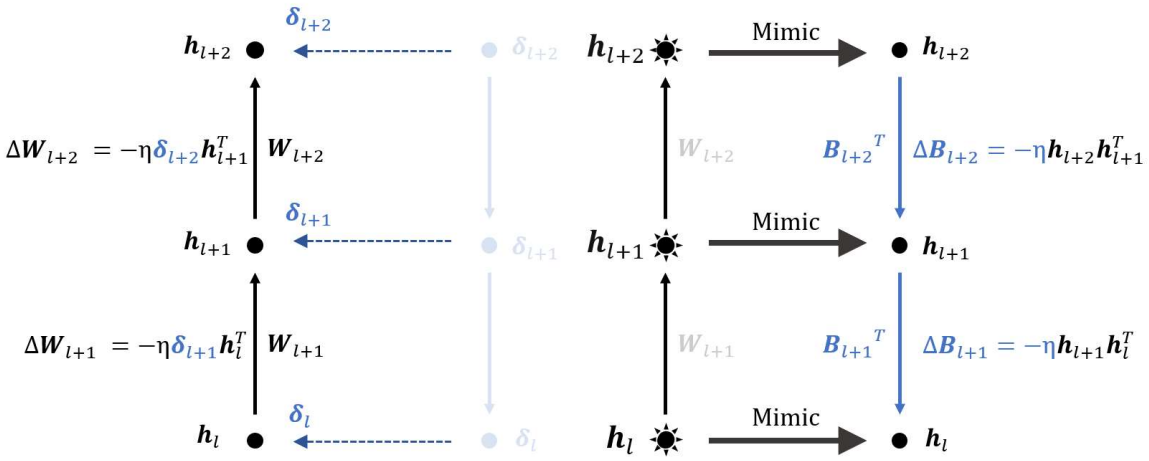
In summary, by eliminating the necessity for bidirectional connections, our ASAP alleviates the structural limitations of stringent two-step interdependence between forward and feedback neurons. Therefore, it can support biological neural networks rather than other algorithms with bidirectional connections, by using one-way skip connections which appears frequently in biological organisms [36–39].

4.2.3 Neuron-specific signals in ASAP

In the general learning rules like Hebbian learning [40], neuron-specific signals, which are signals of pre- or post-synaptic neurons, are used for training weights. However, ASAP do not uses these information. Instead, it utilize synapse-specific signals, which are signals transferred to synapse not from pre- or post-synaptic neurons, but from other neurons. In Fig. 4.3, for example, pre-synaptic neuron $\boldsymbol{h}_{l+1}$ and post-synaptic neurons $\boldsymbol{h}_{l+2}$ are not used for updating weights $\boldsymbol{W}_{l+1}$, but $\boldsymbol{h}_l$ and $\boldsymbol{\delta}_{l+2}$ are transferred to synapse $\boldsymbol{W}_{l+2}$ and they are used for training the synapse. However, this synapse-specific learning requires a mount of connections from other neurons in order to learn just a one connection of synapse because, there is no spatial information about synapse in the neurons which are not pre- or post-synaptic neurons. On the other hands, just two neurons connected by a synapse are required to learn this synapse in neuron-specific learning. Therefore, neuron-specific learning is more reasonable than synapse-specific learning.

To deal with this issue, we apply mirror mode to our algorithm, which is proposed by Akrout et al [24]. In Weight Mirror Algorithm, feedback weights $\boldsymbol{B}_{l+1}$ are updated by using equation (2.6) with $\boldsymbol{h}_l$ and $\boldsymbol{h}_{l+1}$ as shown in Fig.2.4. In this case, it does not utilize neuron-specific signals (pre-synaptic neuron $\boldsymbol{\delta}_l$ and post-synaptic neuron $\boldsymbol{\delta}_{l+1}$), but utilizes synapse-specific signals ($\boldsymbol{h}_l$ and $\boldsymbol{h}_{l+1}$). However, after learning forward

weights $\boldsymbol{W}_{l+1}$ in engage mode as depicted in Fig.4.4a, pre- and post-synaptic neurons of feedback weights $\boldsymbol{B}_{l+1}$ mimic forward neurons ($\boldsymbol{\delta}_{l+1} \rightarrow \boldsymbol{h}_{l+1}, \boldsymbol{\delta}_{l} \rightarrow \boldsymbol{h}_{l}$) in mirror mode as shown in Fig. 4.4b, and therefore neuron-specific learning is possible.



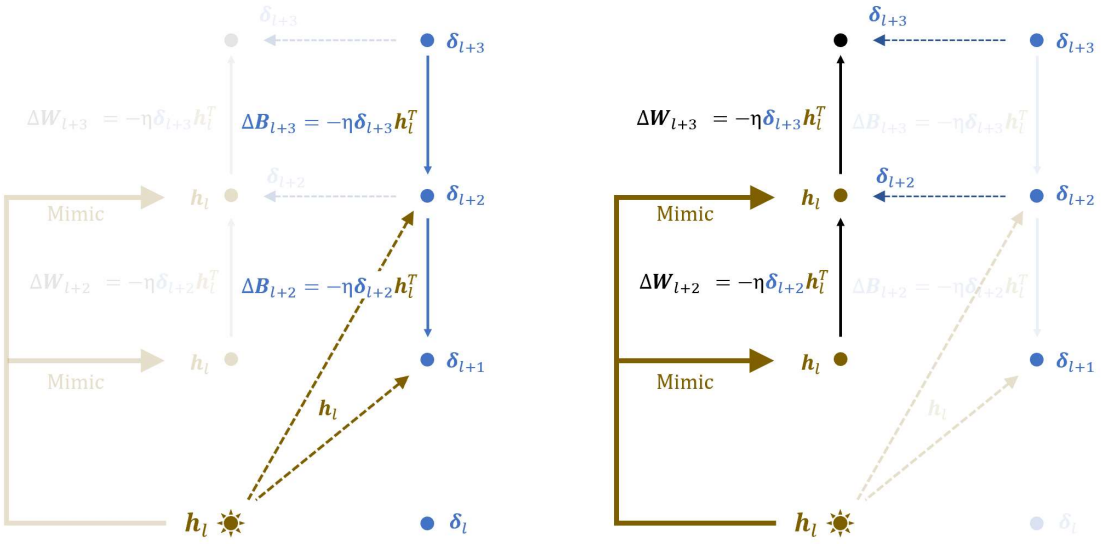
(a) engaged mode (b) mirror mode

Fig. 4.4: WM implementation using neuron-specific signals.

Similarly, the neuron-specific learning is also possible in ASAP by mimicking neurons. To update feedback weights ($\boldsymbol{B}_{l+3}$ and $\boldsymbol{B}_{l+2}$), pre-synaptic neurons ($\boldsymbol{\delta}_{l+3}$ and $\boldsymbol{\delta}_{l+2}$) and received signals ($\boldsymbol{h}_{l}$) from forward path are used in Fig. 4.5a. In forward path, mimicry occurs (i.e. $\boldsymbol{h}_{l+1} \rightarrow \boldsymbol{h}_{l}, \boldsymbol{h}_{l+2} \rightarrow \boldsymbol{h}_{l}$) and the forward weights are updated by using pre- and post-synaptic neurons ($\boldsymbol{h}_{l}$) and received signals ($\boldsymbol{\delta}_{l+2}$ and $\boldsymbol{\delta}_{l+3}$)from feedback path.

While Weight Mirror algorithm have to update feedback weight in mirror mode after weight update of feedback weight is finished in engaged mode, our ASAP can do forward and feedback weights concurrently. In Weight Mirror, the feedback neuron $\boldsymbol{\delta}_{l+1}$, which is used to update forward weights $\boldsymbol{W}_{l+1}$, mimics forward neuron $\boldsymbol{h}_{l+1}$ as shown in Fig. 4.4b (i.e., $\boldsymbol{\delta}_{l} \rightarrow \boldsymbol{h}_{l}$). Therefore, if forward weights and feedback path are updated concurrently, the forward weight uses received signal $\boldsymbol{h}_{l}$ which is mimicked, and therefore the learning process is confused. On the other hands, when ASAP is implemented, feedback updates do not require information of forward neurons ($\boldsymbol{h}_{l+1}$

and $\boldsymbol{h}_{l+2}$) which mimic shared activation ($\boldsymbol{h}_l$) as shown in Fig. 4.5b. Therefore, there is no confusion of neuronal information between forward and feedback path if forward and feedback weights are learned simultaneously.



(a) Updating feedback path (b) Updating forward path with mimicry

Fig. 4.5: ASAP implementation using neuron-specific signals.

4.3 Activation Sharing in Feedback Path: ASFP

In above section, we show that our ASAP can do neuron-specific learning by mimicry. However, this mimicry make our biological learning rule complex. Therefore, we propose *Activation Sharing in Feedback Path* (ASFP) which adopt activation sharing to only feedback path, not to forward path. The detail process of ASFP is as follow:

$$\Delta \boldsymbol{W}_{l+2} = -\eta \, \boldsymbol{\delta}_{l+2} \, \boldsymbol{h}_{l+1}^T \tag{4.7}$$

$$\Delta \boldsymbol{W}_{l+3} = -\eta \, \boldsymbol{\delta}_{l+3} \, \boldsymbol{h}_{l+2}^T \tag{4.8}$$

$$\Delta \boldsymbol{B}_{l+2} = -\eta \, \boldsymbol{\delta}_{l+2} \, \boldsymbol{h}_{l}^T \tag{4.9}$$

$$\Delta \boldsymbol{B}_{l+3} = -\eta\, \boldsymbol{\delta}_{l+3}\, \boldsymbol{h}_l^T \tag{4.10}$$

While ASAP uses shared activation $\boldsymbol{h}_l$ to update both forward weights ($\boldsymbol{W}_{l+2}$, $\boldsymbol{W}_{l+3}$ and feedback weights ($\boldsymbol{B}_{l+2}, \boldsymbol{B}_{l+3}$) and feedback as shown in equation (4.5-4.6), ASFP does not uses shared activation $\boldsymbol{h}_l$ to update ($\boldsymbol{W}_{l+2}$, $\boldsymbol{W}_{l+3}$). Instead, it uses pre-synaptic neurons ($\boldsymbol{h}_{l+1}, \boldsymbol{h}_{l+2}$) to update feedback path by equation (4.7-4.8) while the shared activation ($\boldsymbol{h}_l$) is still used for forward weight updates by equation (4.9-4.10). In this algorithm, both forward path and feedback path uses pre-synaptic neuronal information as shown in Fig. 4.6, and therefore additional mimicry is unnecessary.

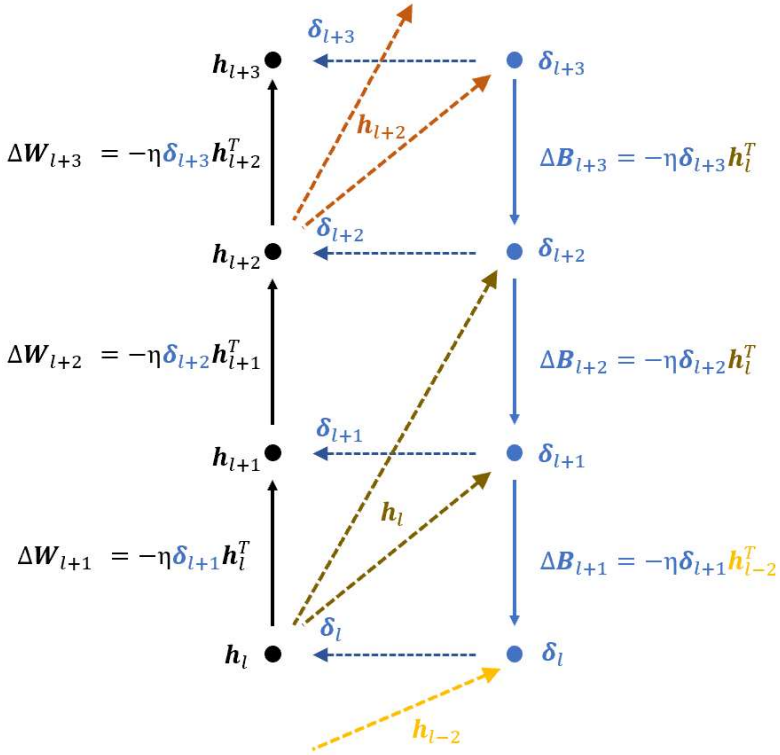


Fig. 4.6: Activation Sharing in Feedback Path

Chapter 5. Experiments

In this section, we verified the validity of the Activation Sharing by experiments. We would describe the detail experimental setting in Section 5.1. Afterwards, the results of experiments are discussed in Section 5.2

5.1 Experimental Details

5.1.1 Contents of experiments

Matching dimension for sharing activation When activation sharing is performed, shared activation replaces actual activation to update weights. It assumes that the dimensions of shared activation and actual activation are the same. However, the dimension of activation in neural networks changes sometimes; channel size would to be increased and feature map size would be decreased as the layer deepens in general deep convolutional networks. Therefore, we proposed three method to match dimension between shared activation and actual activation when it is different. First, multiple copies of the shared activation are concatenated to fit the channel size and maxpool function is adopted to fit the feature map size in method 1. Second, we match the channel size with concatenation as same as method 1 but average pool is adopted for matching the feature map size in method 2. Third, random fixed weights are applied to shared activaitons just as Forward Alignment. These three methods are experimented in AlexNet [41] on CIFAR-10 [42].

Experiments on ASAP We trained deep convolutional networks with backpropagation (BP), modified Kolen-Pollack (KP), Feedback Alignment (FA), Direct Feedback Alignment (DFA), and proposed Activation Sharing with Asymmetric Paths (ASAP). We also compared our ASAP to diverse local learning algorithms [13, 43, 44], which

trained the network locally. The block size k of ASAP is set to 2. In addition, we increased block size k to 4 to confirm the effect of block size on ASAP algorithm. All experiments using ASAP do not apply ASAP to first layer (i.e., do not train this layer), because input of first layer has considerably smaller dimension than the subsequent layers, and therefore it can't be shared activation. Therefore, we apply activation sharing from second layer. Furthermore, we apply method 1 as mentioned above for matching dimension of shared activation, because method 1 performed well than other methods in above experiments (See Section 5.2.1)

Effect on weight decay on ASAP The theorem in Section 4.2.1 show that the network with asymmetric paths converges to the network with symmetric paths when weight changes of both forward and feedback paths are same except for weight decay term. In this proof, we assume that the weight decay factor λ would play a important role because it controls how fast $\boldsymbol{W}(t+1) - \boldsymbol{B}(t+1) = (1-\lambda)^{t+1}(\boldsymbol{W}(0) - \boldsymbol{B}(0))$ converges to zero. Therefore, we implement the experiments in diverse weight factors to verify this assumption.

Experiments on ASFP To justify our Activation Sharing in Feedback Paths (ASFP), we trained deep convolutional networks with ASFP. Furthermore, we also apply local classifier to ASFP for improving performance. We compared this learning rules with backpropagation, ASAP, and other local learning algorithms [13, 43, 44]

5.1.2 dataset

The MNIST [31] has 60,000 training sets and 10,000 test sets which are 28×28 gray images of digits. The SVHN [45] consists of colored digits images with 32×32 scale, which are divided to 73,257 training sets and 26,032 test sets. On the ohter hands, the CIFAR-10 dataset [42] has 32×32 colored images about 10 objects. It has 50,000 training sets while the number of test sets is 10,000. Likewise, CIFAR-100 [42] also consists of 50,000 colored images for training and 10,000 colored images for

test with 32×32 scale, but the number of objects is 100. The Tiny ImageNet [46] includes 100,000 training sets and 10,000 test sets which are 64×64 colored images for classifying 200 objects. All datasets in experiments are normalized, and we also adopt random crop and random horizontal flip to CIFAR-10, CIFAR-100, and Tiny ImageNet as data argumentation.

5.1.3 Models

We do experiments on AlexNet [41] and ResNet [47]. The AlexNet in our experiments is $64\,Conv3 \times 3 - 128\,Conv3 \times 3 - 256\,Conv3 \times 3 - 1024\,FC - 1024\,FC - 10\,FC$, where $64\,Conv3 \times 3$ denotes convolutional layers with 64 of out channel size and 3×3 of kernel size. The stride and padding of this convolutional layers is set to 1 and 2, respectively. The $1024\,FC$ denotes fully-connected layers with 1024 of output size. The output size of final fully-connected layer is determined by the number of classes in dataset (i.e., $FC\,100$ on CIFAR-100). We also trained ResNet-18 and ResNet-34 described by He et al [47]. Furthermore, we also trained ResNet-18 without shortcut, which is residual connection adding previous activation to current one, to verify how this shortcut affect our ASAP learning rule.

5.1.4 Hyperparameters

In all experiments, the stochastic gradient descent with momentum [48] and the weight decay [32] are adopted when updating weights. The learning rate is scheduled by cosine annealing [49]. We initialize all weights in the networks by Kaming initialization [50]. The batch size, momentum, weight decay factor, and training epoch are set to 128, 0.9, and 5e-4. The learning rates are chosen as follow:

Table 5.1: Learning rates used in experiments.

Dataset	Model	BP	KP	FA	DFA	ASAP
MNIST	AlexNet	3e-2	3e-2	1e-4	1e-4	1e-4
SVHN	AlexNet	3e-2	3e-2	1e-4	1e-4	1e-4
	ResNet-18 (nsc*)	3e-2	3e-2	3e-3	3e-3	3e-3
	ResNet-18	3e-2	3e-2	3e-3	3e-3	3e-3
CIFAR-10	AlexNet	3e-2	3e-2	1e-4	1e-4	1e-4
	ResNet-18 (nsc*)	3e-2	3e-2	3e-3	3e-3	3e-3
	ResNet-18	3e-2	3e-2	3e-3	3e-3	3e-2
	ResNet-34	3e-2	3e-2	3e-3	3e-3	3e-2
CIFAR-100	AlexNet	3e-2	3e-2	1e-4	1e-4	1e-4
	ResNet-18 (nsc*)	3e-2	3e-2	3e-3	3e-3	3e-3
	ResNet-18	3e-2	3e-2	3e-3	3e-3	3e-2
	ResNet-34	3e-2	3e-2	3e-3	3e-3	3e-2
Tiny ImageNet	ResNet-18	3e-2	3e-2	3e-3	3e-3	3e-2
	ResNet-34	3e-2	3e-2	3e-3	3e-3	3e-2

*no shortcut

5.2 Results

5.2.1 Matching dimension for sharing activation

As shown in Fig. 5.1, method 1, which uses concatenating and maxpool function, achieves the best accuracy in CIFAR-10 on AlexNet. We assumed that the reason why method 1 do better is that the difference between actual activation and shared activation made by dimension matching methods is the smallest in method 1. Nevertheless, the method 3, which uses random fixed weight to match dimension, would be more reasonable than method 1 in terms of biological plausibility. We know that the activation sharing is advanced Forward Alignment by using identity matrix as random fixed forward weights. Therefore, method 3 could be interpreted as Forward Alignment is implemented when dimension of activation is changed.

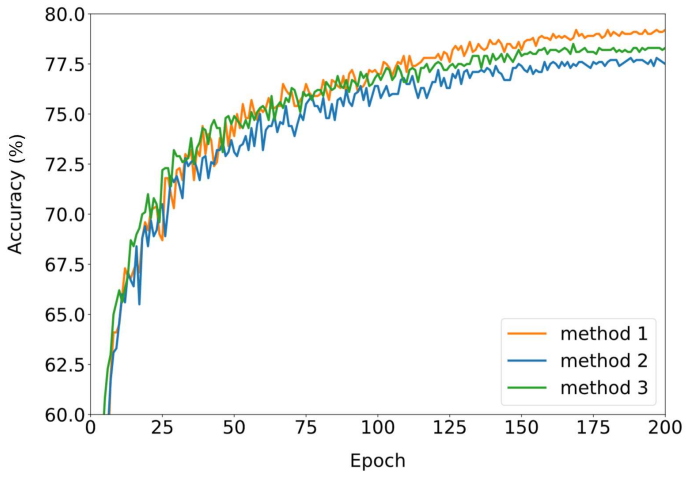


Fig. 5.1: Test accuracy comparisons of dimension matching methods.

5.2.2 Results of ASAP

Table 5.2: Test accuracy of BP, KP, FA, DFA, and ASAP (k=2) in classification task

Dataset	Model	BP	KP	FA	DFA	ASAP
MNIST	AlexNet	99.59	99.55	99.14	99.28	99.32
SVHN	AlexNet	94.64	93.3	82.21	87.42	88.04
	ResNet-18 (nsc*)	96.14	95.84	84.91	85.64	93.17
	ResNet-18	96.29	96.13	85.08	85.56	94.86
CIFAR-10	AlexNet	90.58	79.23	67.92	73.85	78.25
	ResNet-18 (nsc*)	94.70	94.65	71.38	75.46	83.44
	ResNet-18	94.93	94.76	72.47	76.0	92.19
	ResNet-34	95.18	94.54	66.99	73.02	93.97
CIFAR-100	AlexNet	63.61	47.43	33.32	35.38	46.99
	ResNet-18 (nsc*)	77.34	74.98	37.11	37.48	51.72
	ResNet-18	77.74	74.51	38.15	38.51	68.86
	ResNet-34	78.41	75.0	33.01	35.6	72.81
Tiny ImageNet	ResNet-18	60.13	58.14	20.54	24.07	48.46
	ResNet-34	62.63	59.45	16.86	21.1	52.25

*no shortcut

Good performance on deep convolutional networks The test accuracy of training algorithms on various datasets and models is described in the Table 5.2. In all experiments, ASAP consistently outperforms FA and DFA, wchich are the algorithms solving weight transport problem. Although there is performance drop compared with KP, the ASAP removes biologically implausible bidirectional connections while KP

requires them. In other words, Our ASAP achieved the highest performance among biologically plausible algorithms that solved the weight transport problem without bidirectional connection.

The ASAP also achieves comparable performance to BP even in deep convolutioanl networks even on complex datasets. While FA and DFA fail to train ResNet-34 on CIFAR-100 and Tiny-ImageNet, our ASAP achieves 72.81% and 52.25% on CIFAR-100 and Tiny-ImageNet, respectively, which are much more similar performance to BP than FA and DFA. This is because ASAP can take advantage of the benefits of deeper networks while FA and DFA cannot achieve this advantage. We know that the deeper the network, the better the performance. However, FA and DFA do not increase test accuracy when the network is deepen from AlexNet to ResNet-34 as shown in Table 5.2. On the other hands, ASAP increase performance when the network is deepen as BP do.

One interesting thing is that the performance of ASAP is sharply increased when shortcut exist. He et al. [47] introduce shortcut of ResNet increases performance as BP increases test accuracy by 0.23% and 0.5% on CIFAR-10 and CIFAR-100, respectively. In ASAP, however, the performance are increased by 8.75% and 17.14% on CIFAR-10 and CIFAR-100, respectively. We would explain this phenomenon as below.

Effect of shortcut We describe the structure of ResNet block as Fig. 5.2a. When we trained this residual block by backpropagation and ASAP, the equation is as follow:

$$\boldsymbol{h}_{l+2} = \phi(\boldsymbol{W}_{l+2}\boldsymbol{h}_{l+1} + \boldsymbol{b}_{l+2}) + \boldsymbol{h}_l \tag{5.1}$$

$$\Delta\boldsymbol{W}_{l+3} = \boldsymbol{\delta}_{l+3}\boldsymbol{h}_{l+2} = \boldsymbol{\delta}_{l+3}\,\phi(\boldsymbol{W}_{l+2}\boldsymbol{h}_{l+1} + \boldsymbol{b}_{l+2}) + \boldsymbol{\delta}_{l+3}\boldsymbol{h}_l \tag{5.2}$$

$$\Delta\tilde{\boldsymbol{W}}_{l+3} = \boldsymbol{\delta}_{l+3}\boldsymbol{h}_l \tag{5.3}$$

where $\Delta\tilde{\boldsymbol{W}}$ denotes weight updates by ASAP. However, when there is no shortcut, the equation is modified as below:

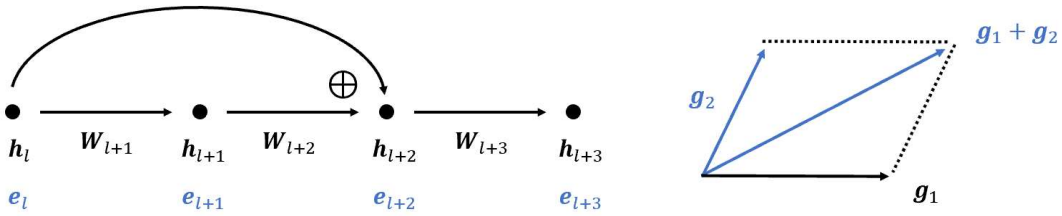
$$\boldsymbol{h}_{l+2} = \phi(\boldsymbol{W}_{l+2}\boldsymbol{h}_{l+1} + \boldsymbol{b}_{l+2}) \tag{5.4}$$

$$\Delta\boldsymbol{W}_{l+3} = \boldsymbol{\delta}_{l+3}\boldsymbol{h}_{l+2} = \boldsymbol{\delta}_{l+3}\,\phi(\boldsymbol{W}_{l+2}\boldsymbol{h}_{l+1} + \boldsymbol{b}_{l+2}) \tag{5.5}$$

$$\Delta\tilde{\boldsymbol{W}}_{l+3} = \boldsymbol{\delta}_{l+3}\boldsymbol{h}_{l} \tag{5.6}$$

For sake of simplicity, we replace $\boldsymbol{\delta}_{l+3}\,\phi(\boldsymbol{W}_{l+2}\boldsymbol{h}_{l+1} + \boldsymbol{b}_{l+2})$ and $\boldsymbol{\delta}_{l+3}\boldsymbol{h}_{l}$ to $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$, respectively. When there is shortcut, the equations (5.2 and 5.3) are changed to $\Delta\boldsymbol{W}_{l+3} = \boldsymbol{g}_1 + \boldsymbol{g}_2$ and $\Delta\tilde{\boldsymbol{W}}_{l+3} = \boldsymbol{g}_2$, respectively. In other words, the same term $\boldsymbol{g}_2$ between $\Delta\boldsymbol{W}_{l+3}$ and $\Delta\tilde{\boldsymbol{W}}_{l+3}$ are made by adding shared activation $\boldsymbol{h}_l$ through shortcut in equation (5.1). We assumed that this same terms make the learning direction of ASAP is similar to that of backpropagation as shown in Fig.4.3, and therefore the high performance is achieved similar to backpropagation.

On the other hands, if there is no shortcut, the equations (5.5 and 5.6) are changed to $\Delta\boldsymbol{W}_{l+3} = \boldsymbol{g}_1$ and $\Delta\tilde{\boldsymbol{W}}_{l+3} = \boldsymbol{g}_2$, respectively. Therefore, there is no same term between $\Delta\boldsymbol{W}_{l+3}$ and $\Delta\tilde{\boldsymbol{W}}_{l+3}$. Consequently, ASAP do not train the network in the similar direction to backpropagation compared to when there is a shortcut as shown in Fig.4.3.



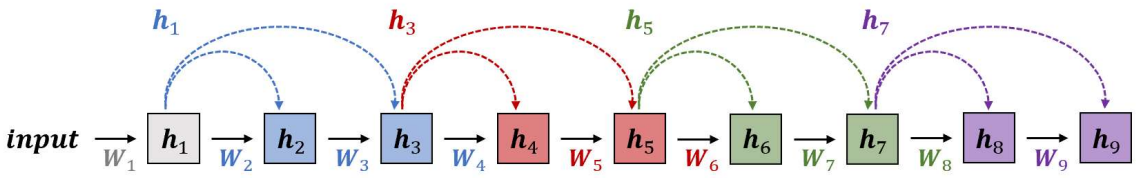
(a) Shortcut in ResNet (b) Better alignment due to shortcut

Fig. 5.2: Overview of shortcut effect.

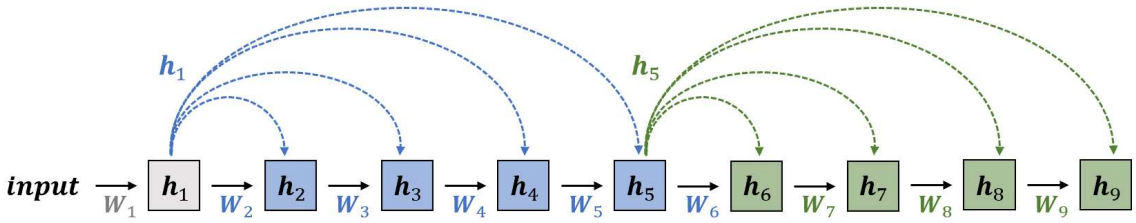
In summary, our ASAP has the strength to train the networks with residual connec-

tion, which are state-of-art architecture on classification task. We intuitively assume the reason is because shared activation, which is added to output activation through shortcut, makes the learning direction of ASAP close to that of backpropagation. In next study, we will prove this assumption more elaborately.

Effect of block size The block size itself has little bearing on biological plausibility since the act of sharing activation through several layers itself destroys the bidirectional connection as shown in Fig. 4.3. However, we can reduce memory overhead by increasing block size k. If block size k is increased, the number of layers, which do not store actual activation and just use shared activation for weight updates, increases. For example, $\boldsymbol{h}_1$, $\boldsymbol{h}_3$, $\boldsymbol{h}_5$, and $\boldsymbol{h}_7$ have to stored when block size k is 2 as shown in Fig. 5.3a while just $\boldsymbol{h}_1$ and $\boldsymbol{h}_5$ are required when block size k is 4 in Fig. 5.3b (The detail analysis of memory overhead is described in Section 6).



(a) ASAP (k=2)

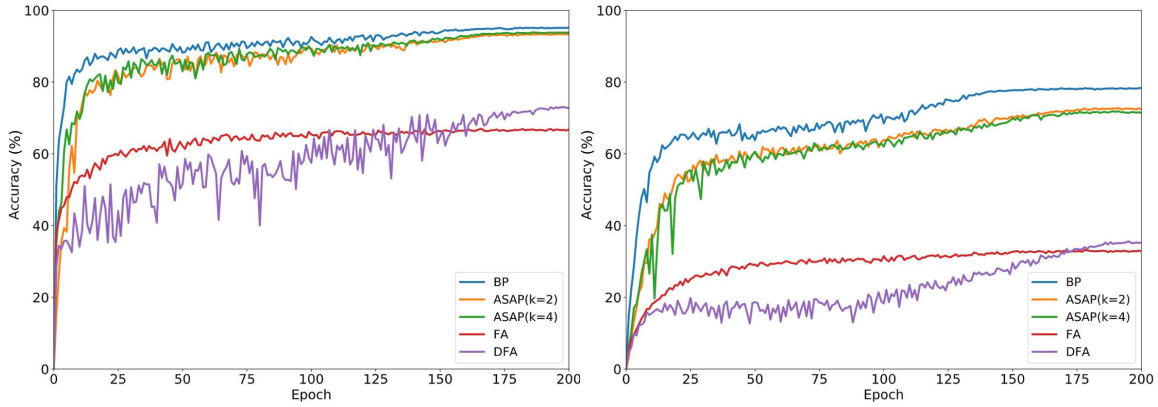


(b) ASAP (k=4)

Fig. 5.3: Overview of ASAP algorithm.

Therefore, we trained ResNet-34 by ASAP with $k = 4$ and compared this with other algorithms as shown in Fig.5.4. Interestingly, the ASAP with $k = 4$ achieves high performance similar to ASAP with $k = 2$. On CIFAR-10, the performance of

ASAP is 93.9% when $k = 4$, which is comparable to 93.97% when $k = 2$. There are little performance drop between 72.81% of $k = 2$ and 71.93% of $k = 4$ on CIFAR-100, however it still it still maintains much better performance than FA and DFA. In other words, we can reduce memory overhead of ASAP while maintaining competitive performance compared with other biologically plausible algorithm by reducing block size.



(a) Test accuracy on CIFAR-10 (b) Test accuracy on CIFAR-100

Fig. 5.4: Training performance comparisons for ResNet-34

Compared with local learning In backpropagation, input propagates through forward path by equation (2.1), and then global error, which is made by total output of network, is backward-propagated by equation (2.2). Finally, the weights are updated by using forward activations and backward errors by equation (2.3). On the other hands, the local learning algorithms [13, 43, 44] train the network locally. In other words, when the input propagates a layer, the output activation of the layer immediately generates local errors, and then weight of the layer is updated by this local errors. Mostafa et al. [51] generates local errors by using local classifier, which uses random fixed weights to make local outputs. The local errors are calculated by local outputs and targets vector. Nokland and Eidnes [43] uses both fully-connected layer and convolutional layer as local layer to make local error. Pogodin et al. [44] uses Hilbert-Schmidt Independence Criterion (HSIC) [52] to make local error. These local learning algo-

rithms are more biologically plausible than backpropagation because it is speculated that biological neural networks train its synapse locally.

Table 5.3: Test accuracy of ASAP and local learning algorithms on CIFAR-10

Architecture	BP	[51]	[43]	[44]	ASAP
ResNet-18	94.93	71.41	88.12	80.76	92.19
ResNet-34	95.18	69.8	87.57	77.2	93.97

We display the results of local learning algorithms mentioned above, and compared with our ASAP in Table 5.3. Despite achieving high performance on architecture like VGG, local learning algorithms degrade noticeably in ResNet, as was already reported in [43] while our ASAP achieves comparable performance to backpropagation. We assumed the reason is that the global errors can not back-propagated and just local errors are generated in local learning algorithms. Therefore, they do not take a advantage of residual connection proposed by He et al [50]. On the other hands, our ASAP not only takes a advantage of residual connection because it uses global errors, but also occurs effect of shortcut as described in Fig. 5.3b. Consequently, our ASAP can outperforms other local learning algorithms in ResNet.

5.2.3 Effect of weight decay on ASAP

Table 5.4: Test accuracy of diverse weight decay factor (λ) on CIFAR-10

Learning rules	λ = 0	λ = 5e-6	λ = 5e-5	λ = 5e-4	λ = 5e-3
BP	93.56	93.76	93.82	95.18	95.12
ASAP	89.77	89.85	91.63	93.97	91.54

We trained ResNet-34 on various weight decay factor as depicted in Table 5.4. The weight decay has an impact on performance in both BP and ASAP. However, this impact is grater in ASAP resulting larger performance variation than backpropagation. As explained in Section 4.2.1, large weight decay factor converges $\boldsymbol{W}(t+1) - \boldsymbol{B}(t+1) = (1-\lambda)^{t+1}(\boldsymbol{W}(0) - \boldsymbol{B}(0))$ to zero fast, and therefore the performance is increased

when weight decay factor is increased in ASAP. However, this trend is broken when weight decay factor is too large. When weight decay factor is 5e-3, the test accuracy is 91.54% which is lower than performance 93.9% when weight decay factor is 5e-4. This is because too large decay factor interrupt appropriate training as backpropagation do. Consequently, we have to choose appropriate weight decay factor to train network successfully by ASAP.

5.2.4 Results of ASFP with local learning

Table 5.5: Test accuracy of ASFP with local classifier on CIFAR-10

Architecture	BP	ASAP	ASFP	ASFP + LC*
ResNet-18	94.93	92.19	61.45	88.46
ResNet-34	95.18	93.97	31.7	89.28

*Local Classifier

We trained ResNet-18 and ResNet-34 on CIFAR-10 by ASFP which is described in Section 4.3. Table 5.5 shows that results of the experiments. We confirmed that our ASFP do not train deep convolutional networks well occurring performance drop by 33.48% and 64.48%, respectively. We assumed that this performance degradation occurs because the weight update of forward path ($\Delta \boldsymbol{W}$) is different from that of feedback path ($\Delta \boldsymbol{B}$) by equations (4.7-4.10).

To alleviate this issue, we adopt local classifier [51] to ASFP. We placed a local classifier at the end of the ASFP block. The local classifier consists of 2×2 average pool layer and fully-connected layer, and this fully-connected layer is trained by Feedback Alignment. By adopting this method, the performance of ASFP is increased to 88.46 and 89.28 in ResNet-18 and ResNet-34, respectively. Although the performance is decreased compared with BP and ASAP, ASFP with local classifier performed better than FA, DFA, and local learning algorithms as described in Table 5.2 and Table 5.3.

Chapter 6. Hardware implementation of ASAP

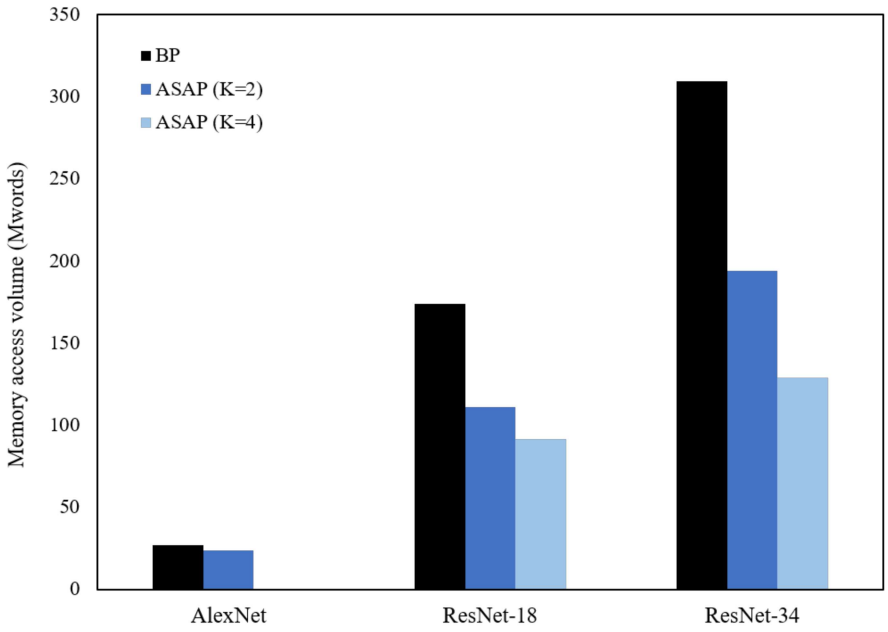


Fig. 6.1: External memory access comparisons.

When training deep neural network in hardware, a mount of calculation and memory overhead are the bottlenecks sze2017efficient. To compute this memory cost, Mostafa et al. [13] proposed the method to estimate external memory access when selected algorithm is implemented on hardware. They assumed that on-chip memory is sufficient for buffering input and output activations. When the forward path is implemented, the parameters like weights would be read from memory, and the intermediate output activations are calculated by compute device. This intermediate output activations, which are used in feedback path, would be written to the memory. The read and write accesses in forward path are expressed as below:

$$(n_{read},\, n_{write}) = (\boldsymbol{Param},\, \boldsymbol{Act}) \tag{6.1}$$

where $\boldsymbol{Param}$ and $\boldsymbol{Act}$ represents the number of parameters and intermediate

output activations, respectively. When the backward path is implemented, the parameters and activations would be read from the external memory in order to calculate back-propagated errors and update the weights. Afterwords, the computed weight updates are written back to the external memory for next epoch of training. Therefore, read and write accesses can be expressed as follow:

$$(n_{read}, n_{write}) = (\boldsymbol{Param} + \boldsymbol{Aaram}, \boldsymbol{Param}) \tag{6.2}$$

By equations (6.1-6.2), the number of external memory accesses is computed when deep convolutional networks are trained by backpropagation, ASAP with $k = 2$, and ASAP with $k = 4$ as shown in Fig. 6.1. We can confirm that our ASAP can reduce the number of external memory accesses compared to backpropagation. Furthermore, this memory cost could be more reduced by increasing block size. This trend is pronounced in deeper networks by reducing lots of memory accesses more than shallow networks. By this results, we expect that our ASAP could be adopted for memory-efficient hardware.

Chapter 7. Conclusion

In this study, we proved that the learning is possible by mathematically proof and experimental results even if approximate activations is used for weight updates instead of exact activations. Furthermore, we proposed activation sharing that advanced on idea above, and we solved weight transport problem without bidirectional connection successfully by applying activation sharing with asymmetric paths. This ASAP algorithm not only alleviated structural constraints on biological neural networks, but also achieved high performance close to backpropagation even in deep convolutional neural networks on complex datasets like Tiny-ImageNet. In addition, our algorithms sharply reduced memory overhead when implemented in hardware.

Chapter 8. Appendix

8.1 Gradient Free Approximation for biological plausibility

Gradient descent [25] is adopted to backpropagation for training deep neural networks in deep learning. As explained in Section 2.1, backpropagation based on gradient descent causes weight transport problem. Furthermore, it incurs separated feedback pathway as shown in Fig. 2.1 while there is no certainty that this pathway exists in biological neural networks. Moreover, it is still unknown whether gradients can be calculated in the brain.

Gradient Free Approximation [53–55], which does not calculate gradient for learning networks, can be a good solution to alleviate above issues. It does not require separated feedback path, which generates gradients, because it can train networks without gradients. Therefore, weight transport between two path is also not required. Furthermore, it is consistent with the observation that there is no evidence that gradients can be computed in biological neural networks.

Although gradient free approximation has possibility to develop biologically plausible algorithm, it suffer from two problems. First, it cannot be scaled to large-size tasks [56]. For instance, COBYLA [57], which is gradient free optimization solver, is only capable of handling problem with up to 2^{16} variables, where it is smaller than a single image of ImageNet. Second, gradient free optimization has a slow convergence rate than gradient descent because it does not use exact information of derivative function [54–56]. We would solve this problem in next study.

8.2 Pseudocode

Algorithm 1 Pseudocode of ASAP

procedure FORWARD PROPAGATION ▷ Determine shared activation
  $\tilde{\boldsymbol{h}}_1 = \boldsymbol{h}_1$
  $\boldsymbol{h}_{1,0} = \phi(\boldsymbol{W}_{1,0}\boldsymbol{h}_1 + \boldsymbol{b}_{1,0})$
  for $m = 1$ to M **do**
    for $k = 1$ to K **do**
      $\boldsymbol{h}_{m,k} = \phi(\boldsymbol{W}_{m,k}\boldsymbol{h}_{m,k-1} + \boldsymbol{b}_{m,k})$
    end for
    $\tilde{\boldsymbol{h}}_{m+1} = \boldsymbol{h}_{m,K-1}$
    $\boldsymbol{h}_{m+1,0} = \boldsymbol{h}_{m,K}$
  end for
end procedure

procedure BACKWARD PROPAGATION ▷ Propagate error through feedback weights
  $\boldsymbol{y}_{M,K} = softmax(\boldsymbol{h}_{M,K})$, $\boldsymbol{\delta}_{M,K} = \boldsymbol{y}_{M,K} - \boldsymbol{y}$
  for $m = M$ to 1 **do**
    for $k = K$ to 1 **do**
      $\boldsymbol{\delta}_{m,k-1} = \phi' \boldsymbol{B}_{m,k}^T \boldsymbol{\delta}_{m,k}$
    end for
  end for
end procedure

procedure WEIGHT UPDATE ▷ Use shared activation for weight changes
  for $m = 1$ to M **do**
    for $k = 1$ to K **do**
      $\Delta \boldsymbol{W}_{m,k} = \boldsymbol{\delta}_{m,k}\tilde{\boldsymbol{h}}_m^T - \lambda \boldsymbol{W}_{m,k}$
      $\Delta \boldsymbol{B}_{m,k} = \boldsymbol{\delta}_{m,k}\tilde{\boldsymbol{h}}_m^T - \lambda \boldsymbol{B}_{m,k}$
    end for
  end for
end procedure

Bibliography

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Neurips*, 2020.

[2] D. S. Modha, "Introducing a brain-inspired computer," *Published online at http://www. research. ibm. com/articles/brain-chip. shtml*, 2017.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[4] Y. Bengio, D. Lee, J. Bornschein, and Z. Lin, "Towards biologically plausible deep learning," *CoRR*, 2015.

[5] A. G. O. II, P. Haffner, D. Reitter, and C. L. Giles, "Learning to adapt by minimizing discrepancy," *CoRR*, 2017.

[6] S. Bartunov, A. Santoro, B. A. Richards, L. Marris, G. E. Hinton, and T. P. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," in *Advances in Neural Information Processing Systems*, 2018.

[7] J. C. Whittington and R. Bogacz, "Theories of error back-propagation in the brain," *Trends in cognitive sciences*, 2019.

[8] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive science*, 1987.

[9] F. Crick, "The recent excitement about neural networks.," *Nature*, vol. 337, no. 6203, pp. 129–132, 1989.

[10] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, 2016.

[11] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[12] C. Frenkel, M. Lefebvre, and D. Bol, "Learning without feedback: Direct random target projection as a feedback-alignment algorithm with layerwise feedforward training," *stat*, 2019.

[13] T. H. Moskovitz, A. Litwin-Kumar, and L. F. Abbott, "Feedback alignment in deep convolutional networks," *CoRR*, 2018.

[14] J. Launay, I. Poli, F. Boniface, and F. Krzakala, "Direct feedback alignment scales to modern deep learning tasks and architectures," in *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[15] B. J. Lansdell, P. R. Prakash, and K. P. Kording, "Learning to solve the credit assignment problem," in *International Conference on Learning Representations*, 2020.

[16] J. Guerguiev, K. Kording, and B. Richards, "Spike-based causal inference for weight alignment," in *International Conference on Learning Representations*, 2020.

[17] W. Xiao, H. Chen, Q. Liao, and T. Poggio, "Biologically-plausible learning algorithms can scale to large datasets," in *International Conference on Learning Representations*, 2019.

[18] A. Gushchin and A. Tang, "Total wiring length minimization of c. elegans neural network: a constrained optimization approach," *PloS one*, 2015.

[19] M. Langen, E. Agi, D. J. Altschuler, L. F. Wu, S. J. Altschuler, and P. R. Hiesinger, "The developmental rules of neural superposition in drosophila," *Cell*, 2015.

[20] S. L. Palay and V. Chan-Palay, *Cerebellar cortex: cytology and organization*. 2012.

[21] J. Sacramento, R. P. Costa, Y. Bengio, and W. Senn, "Dendritic cortical microcircuits approximate the backpropagation algorithm," in *Advances in Neural Information Processing Systems*, 2018.

[22] K. S. Rockland, "Elements of cortical architecture," in *Extrastriate cortex in primates*, 1997.

[23] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[24] M. Akrout, C. Wilson, P. Humphreys, T. Lillicrap, and D. B. Tweed, "Deep learning without weight transport," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[25] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[26] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random feedback weights support learning in deep neural networks," *arXiv preprint arXiv:1411.0247*, 2014.

[27] C. Frenkel, M. Lefebvre, and D. Bol, "Learning without feedback: Direct random target projection as a feedback-alignment algorithm with layerwise feedforward training," *stat*, 2019.

[28] D. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Machine Learning and Knowledge Discovery in Databases*, 2015.

[29] K. S. Rockland, J. H. Kaas, and A. Peters, *Cerebral Cortex: Volume 12: Extrastriate Cortex in Primates*, vol. 12. Springer Science & Business Media, 2013.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on Machine Learning*, PMLR, 2015.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[32] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, 1992.

[33] P. C. Murphy and A. M. Sillito, "Functional morphology of the feedback pathway from area 17 of the cat visual cortex to the lateral geniculate nucleus," *Journal of Neuroscience*, vol. 16, no. 3, pp. 1180–1192, 1996.

[34] K. S. Rockland and A. Virga, "Terminal arbors of individual "feedback" axons projecting from area v2 to v1 in the macaque monkey: a study using immunohistochemistry of anterogradely transported phaseolus vulgaris-leucoagglutinin," *Journal of Comparative Neurology*, 1989.

[35] K. S. Rockland and G. W. Drash, "Collateralized divergent feedback connections that target multiple cortical areas," *Journal of Comparative Neurology*, vol. 373, no. 4, pp. 529–548, 1996.

[36] G. Buzsáki and K. Mizuseki, "The log-dynamic brain: how skewed distributions affect network operations," *Nature Reviews Neuroscience*, vol. 15, no. 4, pp. 264–278, 2014.

[37] D. Fitzpatrick, "The functional organization of local circuits in visual cortex: insights from the study of tree shrew striate cortex," *Cerebral cortex*, vol. 6, no. 3, pp. 329–341, 1996.

[38] S. W. Oh, J. A. Harris, L. Ng, B. Winslow, N. Cain, S. Mihalas, Q. Wang, C. Lau, L. Kuan, A. M. Henry, *et al.*, "A mesoscale connectome of the mouse brain," *Nature*, vol. 508, no. 7495, pp. 207–214, 2014.

[39] A. M. Thomson, "Neocortical layer 6, a review," *Frontiers in neuroanatomy*, vol. 4, p. 13, 2010.

[40] D. Hebb, "The organization of behavior. emphnew york," 1949.

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

[42] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[43] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," in *International Conference on Machine Learning,*, vol. 97, pp. 4839–4850, PMLR, 2019.

[44] R. Pogodin and P. E. Latham, "Kernelized information bottleneck leads to biologically plausible 3-factor hebbian learning in deep networks," in *Advances in Neural Information Processing Systems*, 2020.

[45] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[46] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, 2015.

[47] 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.

[48] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, 1999.

[49] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *International Conference on Learning Representations*, 2017.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *IEEE international conference on computer vision*, 2015.

[51] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers in neuroscience*, vol. 12, 2018.

[52] A. Gretton, O. Bousquet, A. J. Smola, and B. Schölkopf, "Measuring statistical dependence with hilbert-schmidt norms," in *Algorithmic Learning Theory, 16th International Conference, ALT 2005, Singapore, October 8-11, 2005, Proceedings* (S. Jain, H. U. Simon, and E. Tomita, eds.), vol. 3734 of *Lecture Notes in Computer Science*, pp. 63–77, Springer, 2005.

[53] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

[54] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *Foundations of Computational Mathematics*, vol. 17, no. 2, pp. 527–566, 2017.

[55] A. K. Sahu, M. Zaheer, and S. Kar, "Towards gradient free and projection free stochastic optimization," in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 3468–3477, PMLR, 2019.

[56] S. Liu, P.-Y. Chen, B. Kailkhura, G. Zhang, A. O. Hero III, and P. K. Varshney, "A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications," *IEEE Signal Processing Magazine*, vol. 37, no. 5, pp. 43–54, 2020.

[57] M. J. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," in *Advances in optimization and numerical analysis*, pp. 51–67, Springer, 1994.

초 록
가중치 전송문제와 양방향성 연결 문제를 해결한 뉴로모픽 학습 알고리즘

우 승 현

지능정보융합학과

서울대학교 융합과학기술대학원

최근에는 역전파 (backpropagation) 학습방법이 다양한 분야에 적용되어 좋은 성과를 거두고 있다. 그러나 실제 뇌에서 역전파가 이루어지기에는 많은 한계가 있다. 그 이유 중 하나는 가중치 수송 문제 (weight transport problem)이다. 즉, 역전파 학습방법은 순방향 경로와 대칭적인 역방향 경로를 구성하는데, 이는 실제 뇌에서 생물학적으로 불가능하다. 최근 연구에서는 비대칭 역방향 경로를 구성하되 이를 순방향 경로와 동일하게 학습하여 가중치 전달 문제를 해결하며 역전파에 가까운 성능을 달성했다. 그러나 모든 경로를 훈련시키기 위해 생물학적으로 희귀한 양방향성 연결이 순방향 뉴런과 역방향 뉴런 사이에 발생한다는 문제가 여전히 존재한다.

본 연구에서는 가중치 전달 문제를 해결하면서 양방향 연결이 발생하지 않는 새로운 학습 알고리즘을 제안한다. 제안된 학습방법은 순방향 및 비대칭 역방향 경로를 훈련시키는 동안 여러 계층에서 활성화를 공유하여 양방향 연결을 제거한다. 이 알고리즘은 가중치 전달 문제를 해결한 다른 학습 알고리즘에 비해 성능이 크게 향상되었다. 나아가 기존 학습 알고리즘과 달리 정확한 활성화 정보 없이도 학습이 가능하다는 가능성을 제시했다. 결과적으로 학습 도중 모든 정확한 활성화를 저장할 필요가 없기 때문에 훈련 메모리를 줄일 수 있었다.

주요어: 가중치 수송 문제, 양방향성 연결, 활성화 공유

학번: 2020-23071