Activation Sharing with Asymmetric Paths Solves Weight Transport Problem without Bidirectional Connection

Anonymous Author(s)
Affiliation
Address
email

Abstract

One of the reasons why it is difficult for the brain to perform backpropagation (BP) is the weight transport problem, which argues forward and feedback neurons cannot share the same synaptic weights during learning in biological neural networks. Recently proposed algorithms address the weight transport problem while providing good performance similar to BP in large-scale networks. However, they require bidirectional connections between the forward and feedback neurons to train their weights, which is observed to be rare in general neural network structures in the biological brain. In this work, we propose an Activation Sharing algorithm that removes the need for bidirectional connections between the two types of neurons. In this algorithm, hidden layer outputs (activations) are shared across multiple layers during weight update. By applying the proposed learning rule to both forward and feedback networks, we solve the weight transport problem without the constraint of bidirectional connections, also achieving good performance even on deep convolutional neural networks, unlike other biologically plausible algorithms. In addition, our algorithm could significantly reduce memory access overhead when implemented in hardware.

1 Introdution

Backpropagation (BP) [1] is the representative approach to training various deep neural networks. While BP exhibits excellent training performance, similar to or even better than that of humans, it has been long argued that the structure of biological neural networks does not support the backpropagation of errors [2, 3, 4, 5]. To resolve this issue, a wide range of studies have been conducted to develop an algorithm that is feasible in biological neural networks. One important reason behind the biological implausibility of BP is the *weight transport problem*. BP requires identical forward and feedback paths for reliable training; i.e., the two paths must have the same synaptic weights. While biological neural networks may also implement two separate processing paths (forward and feedback paths), it is impossible to make their weights identical without explicitly passing the weights between the paths, which is called the weight transport problem [6].

The Feedback Alignment (FA) algorithm [7] solves the weight transport problem by propagating errors through the feedback path with random fixed weights. It was shown that the feedback weights could be aligned with the forward weights in the course of training, and consequently, the network is trained in a similar way to BP. Similarly, Direct Feedback Alignment (DFA) [8] directly propagates errors from the top layer to lower layers using random fixed weights, and Direct Random Target Projection algorithm [9] locally creates errors using targets rather than using backpropagated errors. Although these algorithms demonstrate good training performance on simple networks, they exhibit

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

large performance degradation when applied to complex networks, especially deep convolutional neural networks [10, 11].

More recent works suggest training the feedback weights as well as the forward weights instead of using random fixed weights in the feedback path. Training feedback weights through Reinforcement Learning [12] or Spiking Neural Network [13] provides better accuracy in classification tasks than the algorithms using random fixed weights. However, those approaches have been only evaluated in shallow neural networks, and it is questionable whether they would be capable of training deeper neural networks. The Sign-Symmetry algorithm [14] achieves good training performance in deep neural networks by transporting only the signs of the forward weights to the feedback path during training. However, since it still needs to transport the signs of the weights, it is difficult to say that the weight transport problem has been solved entirely.

Recently, Weight Mirrors and modified Kolen-Pollack algorithms [15, 16], which align the feedback weights with the forward weights without explicit weight transport, succeeded in training deep convolutional neural networks on large datasets with outstanding performance close to BP. However, these algorithms require a bidirectional connection between a forward neuron and its corresponding neuron in the feedback path as a means of sharing information between forward and feedback neurons. While this type of connection could be found in some biological neurons [17, 18, 19], it is difficult to generalize them to a broader range of biological neural networks that typically only have unidirectional paths between neurons, where feedback must route through multiple neurons in order to reach their destination [20].

In this paper, we propose an Activation Sharing algorithm, which performs weight update of a layer using the output (*activation*) of a lower layer, not the output of the immediately previous layer. Applying this learning method to both forward and feedback paths solves the weight transport problem without using bidirectional paths, and also successfully trains deep convolutional networks such as ResNet-34. Furthermore, since activation is shared across multiple layers during training, the algorithm could significantly reduce memory access and is suitable for efficient hardware implementation.

The remainder of this paper is organized as follows. In section 2, related works that address the weight transport problem are introduced. Then, we propose the Activation Sharing algorithm and explain how this algorithm solves the weight transport problem without bidirectional connections in section 3. Section 4 shows experimental results of training deep convolutional networks using the proposed algorithm on various datasets. In section 5, we discuss the significance and limitations of the algorithm. Proof of the proposed Activation Sharing algorithm and specific hyperparameters used in the experiments are provided in Appendix.

2 Related works

2.1 Backpropagation and weight transport problem

We consider typical training process of a deep neural network. The forward path propagates input data through multiple layers, generating output activation of each layer:

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

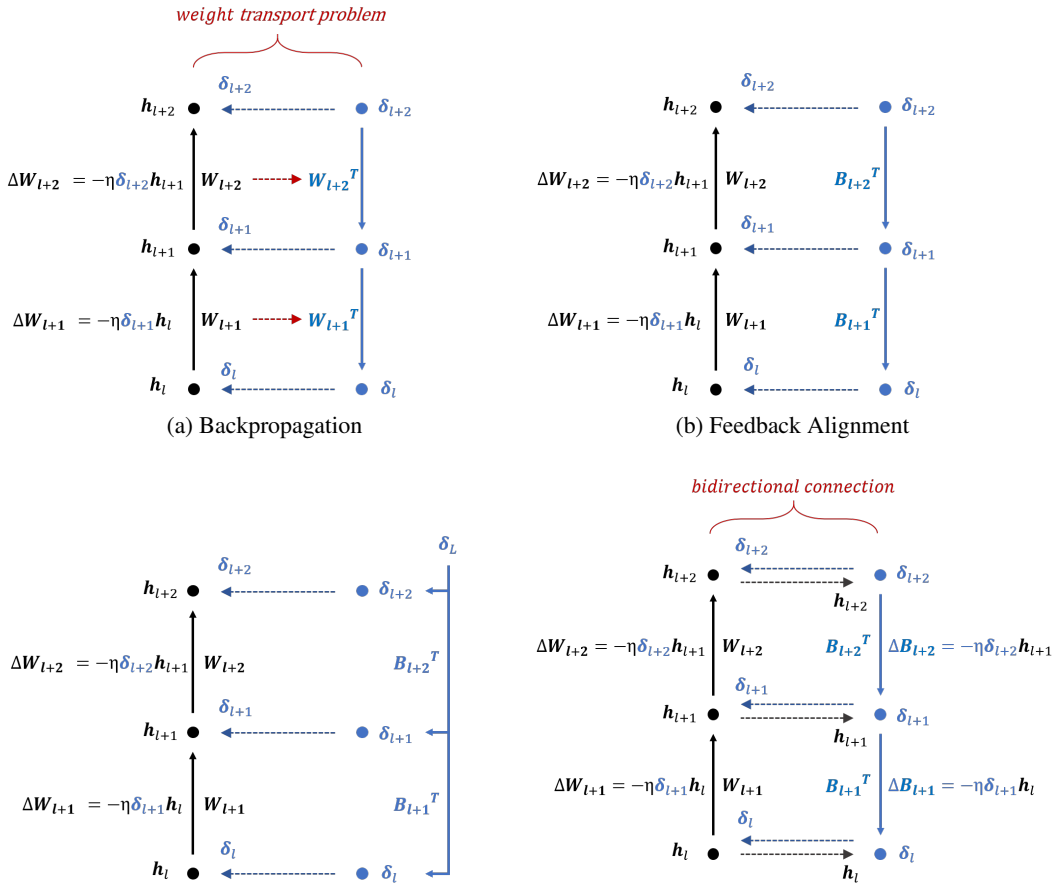
where $\boldsymbol{h}_l$ is the output activation of the hidden layer l, $\boldsymbol{W}_l$ is the weight matrix, and $\boldsymbol{b}_l$ is the bias vector of layer l in the forward path. In the brain, $\boldsymbol{h}$ could be interpreted as neural firing rates, $\boldsymbol{W}$ as synaptic weights between neurons, $\boldsymbol{b}$ as bias current, and ϕ as nonlinearities in neurons [15]. To train the model, synaptic weight changes $\Delta\boldsymbol{W}$ are obtained by

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

where $\boldsymbol{\delta}$ and $\boldsymbol{\eta}$ denote the error signal that flows through the feedback path and the learning rate, respectively. For calculating synaptic weight change $\Delta\boldsymbol{W}$, the error $\boldsymbol{\delta}$ should propagate from the output layer to lower layers in the reverse direction. In backpropagation [1], errors that propagate through the feedback path are represented as

$$\boldsymbol{\delta}_l = \phi^{'}(\boldsymbol{h}_l)\,\boldsymbol{W}_{l+1}^T\,\boldsymbol{\delta}_{l+1} \tag{3}$$

where $\phi^{'}$ denotes the derivative of the activation function ϕ in equation (1). For this BP to occur in the brain, the error must be propagated in the reverse direction of the forward path, which is not



(a) Backpropagation (b) Feedback Alignment

(c) Direct Feedback Alignment (d) Modified Kolen-Pollack

Figure 1: Backpropagation and biologically plausible algorithms.

feasible in biological neural networks that only have unidirectional paths. Even if there is a separate feedback path, as shown in Fig. 1a, the synaptic weights $\boldsymbol{W}$ in equation (1) should be also used to propagate the error in equation (3). In other words, the forward and feedback paths must have identical synaptic weights. Neurons need to exchange their synaptic weights to realize this, but there is no proof that this weight transport actually occurs in the brain (*weight transport problem*).

2.2 Training with random fixed feedback weights

The Feedback Alignment algorithm [7] was developed based on the idea that errors could be propagated using a random fixed weights $\boldsymbol{B}$ rather than the forward weights $\boldsymbol{W}$ in equation (1). As shown in Fig. 1b, errors propagate through the feedback path following the equation below.

$$\boldsymbol{\delta}_l = \phi^{'}(\boldsymbol{h}_l)\, \boldsymbol{B}_{l+1}^T\, \boldsymbol{\delta}_{l+1} \tag{4}$$

Moreover, the Direct Feedback Alignment algorithm [8] showed that the neural network could be trained if the error propagates directly from the output layer to each layer instead of flowing through all hidden layers sequentially (Fig. 1c):

$$\boldsymbol{\delta}_l = \phi^{'}(\boldsymbol{h}_l)\, \boldsymbol{B}_{l+1}^T\, \boldsymbol{\delta}_L \tag{5}$$

where L denotes the output layer of a deep neural network. These methods perfectly solve the weight transport problem by using random fixed weights in the feedback path, and demonstrate good training performance in simple networks. However, they fail to train the model or incur significant performance drop when applied to deep convolutional neural networks [10, 11, 14, 21].

2.3 Training with feedback weight update

Since using random fixed weights for error propagation causes poor performance for training deep and complex neural networks, various methods that train the feedback weights simultaneously have been recently proposed [12, 13, 22, 23]. For example, the Weight Mirror (WM) and modified Kolen-Pollack (KP) algorithms attain very competitive training performance on deep convolutional networks [15]. In WM, the feedback weights are updated by

$$\Delta \boldsymbol{B}_{l+1} = -\eta \, \boldsymbol{h}_{l+1} \, \boldsymbol{h}_l^T \tag{6}$$

using output activations of layers l and $l+1$ while the error propagates in the same way as FA according to equation (4). This update rule makes $\Delta \boldsymbol{B}_{l+1}$ proportional to W_{l+1}, which aligns the feedback weights with the forward weights during training as discussed in [15]. On the other hand, the KP algorithm updates the feedback weights as below:

$$\Delta \boldsymbol{B}_{l+1} = \Delta \boldsymbol{W}_{l+1} = -\eta \, \boldsymbol{\delta}_{l+1} \, \boldsymbol{h}_l^T \tag{7}$$

This method supports updating forward and feedback weights in the same direction, and those two weights could be also aligned during training. These algorithms succeed in training deep convolutaional neural networks with similar performance to BP. Both WM and KP solve the weight transport problem by avoiding explicitly exchanging information related to synaptic weights such as weight changes $\Delta \boldsymbol{W}$ and $\Delta \boldsymbol{B}$ between the two paths; instead, a forward neuron and its corresponding neuron in the feedback path exchange activation $\boldsymbol{h}$ and error $\boldsymbol{\delta}$. For example, $\boldsymbol{\delta}_{l+1}$ of the feedback path is needed to update the forward weights $\boldsymbol{W}_{l+1}$ in equation (2), while $\boldsymbol{h}_{l+1}$ of the forward path is required to update the feedback weights $\boldsymbol{B}_{l+1}$ in equation (6) in WM. KP also needs $\boldsymbol{h}_{l+1}$ to update the feedback weights $\boldsymbol{B}_{l+2}$ in equation (7). However, exchanging information between two neurons constitutes a bidirectional signal path, which is not supported in general neural network structures consisting of only unidirectional connections.

3 Activation Sharing to solve weight transport problem without bidirectional connections

Until now, researches to find biologically plausible algorithms have mainly focused on the propagation of errors or targets in different directions without changing the forward path for training. However, the demand for correct activations to update weights inevitably incurs bidirectional connections when training feedback weights as depicted in Fig. 1d. Our study, on the other hand, focuses on the learning algorithm without accurate activations. In this section, we first formulate the weight update process and define the required information. Based on this, we show that an additional forward path and the original forward path can be aligned, and we present an Activation Sharing algorithm. Finally, we show that the algorithm solves the weight transport problem without bidirectional connections in the network.

3.1 Information required for weight updates

For the sake of simplicity, here we consider a linear network (i.e., we assume $\boldsymbol{h}_{l+1} = \boldsymbol{W}_{l+1} \, \boldsymbol{h}_l$ neglecting ϕ). As shown in equation (2), we need the error $\boldsymbol{\delta}$ and the activation $\boldsymbol{h}$ to update the weights in the network. For backpropagation, we could calculate these values very accurately. We use the activations obtained when the input propagates the layers, and the errors that back-propagated through the same layers in the reverse order. Since we assume a linear network, equation (3) is simplified to $\boldsymbol{\delta}_{l+1} = \boldsymbol{W}_{l+2}^T \, \boldsymbol{\delta}_{l+2}$. However, recent studies suggest that approximate values could also be used for weight update. For instance, FA and DFA use the errors obtained through random fixed weights instead of the backpropagated errors.

To express different types of learning rules, here we denote the activations from the forward operation and the backpropagated errors by $\boldsymbol{h}$ and $\boldsymbol{\delta}$, respectively, whereas the approximate activations and errors actually used for weight update are denoted by $\dot{\boldsymbol{h}}$ and $\dot{\boldsymbol{\delta}}$. Then, the weight of layer $l+2$ is updated by

$$\Delta \boldsymbol{W}_{l+2} = -\eta \, \dot{\boldsymbol{\delta}}_{l+2} \, \dot{\boldsymbol{h}}_{l+1}^T \tag{8}$$

The weight update process in backpropagation (Fig. 2a) follows the equations below.

$$\dot{\boldsymbol{h}}_{l+1} = \boldsymbol{h}_{l+1} = \boldsymbol{W}_{l+1}\,\boldsymbol{h}_l \tag{9}$$

$$\dot{\boldsymbol{\delta}}_{l+2} = \boldsymbol{\delta}_{l+2} = \boldsymbol{W}_{l+3}^T\,\boldsymbol{\delta}_{l+3} \tag{10}$$

$$\Delta\boldsymbol{W}_{l+2} = -\eta\,\dot{\boldsymbol{\delta}}_{l+2}\,\dot{\boldsymbol{h}}_{l+1}^T = -\eta\,\boldsymbol{\delta}_{l+2}\,\boldsymbol{h}_{l+1}^T \tag{11}$$

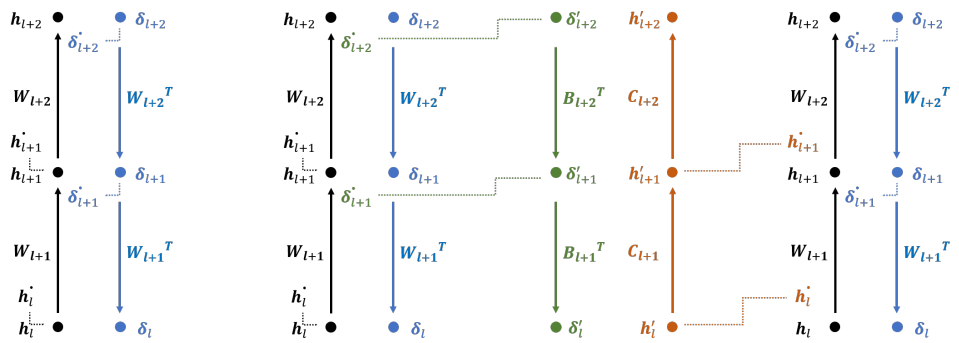
The FA algorithm uses the same activations obtained by propagating the input through the forward path. However, the algorithm uses errors that backpropagated through random fixed weights to update weights. This update process is described in Fig. 2b and is expressed by

$$\dot{\boldsymbol{h}}_{l+1} = \boldsymbol{h}_{l+1} = \boldsymbol{W}_{l+1}\,\boldsymbol{h}_l \tag{12}$$

$$\dot{\boldsymbol{\delta}}_{l+2} = \boldsymbol{B}_{l+3}^T\,\dot{\boldsymbol{\delta}}_{l+3} \neq \boldsymbol{\delta}_{l+2} \tag{13}$$

$$\Delta\boldsymbol{W}_{l+2} = -\eta\,\dot{\boldsymbol{\delta}}_{l+2}\,\dot{\boldsymbol{h}}_{l+1}^T = -\eta\,\dot{\boldsymbol{\delta}}_{l+2}\,\boldsymbol{h}_{l+1}^T \tag{14}$$

where $\dot{\boldsymbol{\delta}}_L = \boldsymbol{\delta}_L$ since the error at the output layer due to the loss function is the same for BP and FA.



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

Figure 2: Overview of weight update process in different algorithms.

3.2 Forward Alignment & Activation Sharing

The Feedback Alignment algorithm uses random fixed weights rather than the transposed forward weights to obtain errors. This approach successfully trains a neural network by aligning the forward weights $\boldsymbol{W}$ with the random fixed weights $\boldsymbol{B}$. Similarly, we hypothesize that the network could be trained even if we use approximate activations obtained by propagating the input through random fixed weights $\boldsymbol{C}$ instead of the forward path, which we dub *Forward Alignment*. The flow of the Forward Alignment is described in Fig. 2c and the equations below.

$$\dot{\boldsymbol{h}}_{l+1} = \boldsymbol{C}_{l+1}\,\dot{\boldsymbol{h}}_l \neq \boldsymbol{h}_{l+1} \tag{15}$$

$$\dot{\boldsymbol{\delta}}_{l+2} = \boldsymbol{\delta}_{l+2} = \boldsymbol{W}_{l+3}^T\,\boldsymbol{\delta}_{l+3} \tag{16}$$

$$\Delta\boldsymbol{W}_{l+2} = -\eta\,\dot{\boldsymbol{\delta}}_{l+2}\,\dot{\boldsymbol{h}}_{l+1}^T = -\eta\,\boldsymbol{\delta}_{l+2}\,\boldsymbol{h}_{l+1}^T \tag{17}$$

where $\dot{\boldsymbol{h}}_0 = \boldsymbol{h}_0$ since the network input between the forward path and the additional forward path having random fixed weights are equal. The Feedback Alignment algorithm was proven to work in simple linear networks through Barbalat's lemma that the loss of the network converges to zero if certain conditions are satisfied [24]. Likewise, it can be shown that the loss converges to zero by Forward Alignment in simple linear networks through Barbalat's lemma (see Appendix for proof). In experiments, we confirmed that the Forward Alignment algorithm aligns the forward weights with the random fixed weights to achieve learning in deep neural networks.

Extending this idea, we propose an Activation Sharing algorithm that updates the weights using the activations of a lower layer based on two intuitions. First, since the Forward Alignment algorithm works with any random fixed weights, the identity matrix $\boldsymbol{I}$ could also be used for generating approximate activations when the adjacent layers have the same input dimension. Second, propagating

the input $\boldsymbol{h}_0$ through more layers with random fixed weights to obtain approximate activations $\dot{\boldsymbol{h}}$ would incur more deviations from the forward activations $\boldsymbol{h}$ since $\dot{\boldsymbol{h}}_l = \boldsymbol{C}_{l-1}\,\dot{\boldsymbol{h}}_{l-1} = \prod \boldsymbol{C}_k\,\boldsymbol{h}_0$. On the other hand, if we limit the propagation of input or activation through random fixed weights for approximate activation generation within a *block*, which is defined as a set of multiple consecutive layers in the network, this deviation would be significantly reduced and the training process would more closely follow the backpropgation. For instance, if each block consists of two layers, $\dot{\boldsymbol{h}}_l = \boldsymbol{C}_{l-1}\,\boldsymbol{C}_{l-2}\,\boldsymbol{h}_{l-2}$, which would be closer to $\boldsymbol{h}_l$ than $\dot{\boldsymbol{h}}_l = \prod \boldsymbol{C}_k\,\boldsymbol{h}_0$. Combining these two intuitions, we propose the Activation Sharing algorithm with a block size of 2 as follows:

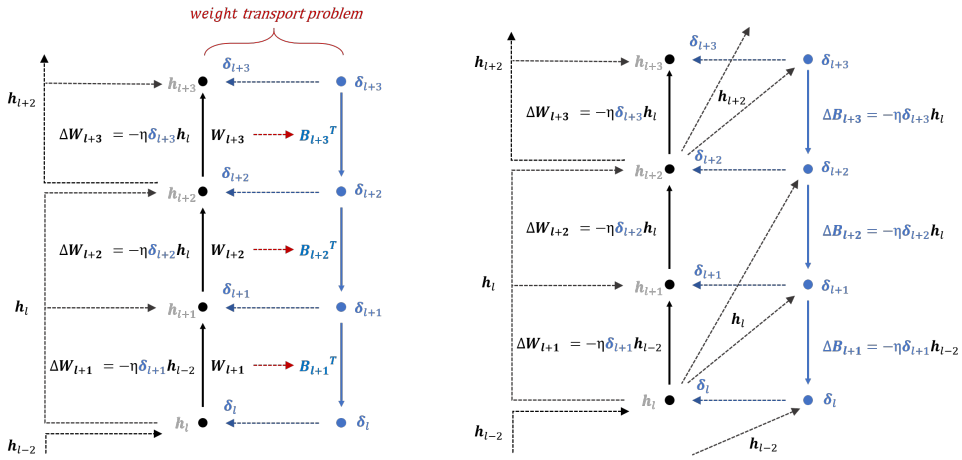
$$\dot{\boldsymbol{h}}_{l+1} = \boldsymbol{C}_{l+1}\,\dot{\boldsymbol{h}}_l = \boldsymbol{h}_l \tag{18}$$

$$\dot{\boldsymbol{h}}_{l+2} = \boldsymbol{C}_{l+2}\,\dot{\boldsymbol{h}}_{l+1} = \boldsymbol{C}_{l+2}\,\boldsymbol{C}_{l+1}\,\dot{\boldsymbol{h}}_l = \boldsymbol{h}_l \tag{19}$$

$$\Delta\boldsymbol{W}_{l+2} = -\eta\,\dot{\boldsymbol{\delta}}_{l+2}\,\dot{\boldsymbol{h}}_{l+1}^T = -\eta\,\dot{\boldsymbol{\delta}}_{l+2}\,\boldsymbol{h}_l^T \tag{20}$$

$$\Delta\boldsymbol{W}_{l+3} = -\eta\,\dot{\boldsymbol{\delta}}_{l+3}\,\dot{\boldsymbol{h}}_{l+2}^T = -\eta\,\dot{\boldsymbol{\delta}}_{l+3}\,\boldsymbol{h}_l^T \tag{21}$$

The algorithm updates the weights of all the layers inside a block using the same activations, which is the output activation of the previous block, through equations (18) and (19). Hence, all layers in the block update their weights using the same shared activation $\boldsymbol{h}_l$ (equations (20) and (21)). In other words, once the activations to be shared are determined, the weights are updated using the designated activations instead of their own activations for k layers when the block size is k. Fig. 3a depicts the Activation Sharing algorithm when $k = 2$. If $K = L - 1$, all layers except the first layer share the output activations of the first layer to update the weights (see the pseudocode in the Appendix for details).



(a) Activation Sharing (k=2)

(b) Activation Sharing with Asymmetric Paths (k=2)

Figure 3: Activation Sharing & Activation Sharing with Asymmetric Paths

3.3 How does Activation Sharing solve weight transport problem without bidirectional connections?

Kolen and Pollack [16] showed that when $\Delta\boldsymbol{W} = \Delta\boldsymbol{B}$ where $\boldsymbol{W}$ and $\boldsymbol{B}$ are the forward and feedback weights, the network is trained similarly to a typical artificial neural network with symmetric forward and feedback paths ($\boldsymbol{W} = \boldsymbol{B}$). For proof, the weight changes with weight decay [25] are written as

$$\Delta\boldsymbol{W}(t) = \boldsymbol{R}(t) - \lambda\,\boldsymbol{W}(t) \tag{22}$$

$$\Delta\boldsymbol{B}(t) = \boldsymbol{R}(t) - \lambda\,\boldsymbol{B}(t) \tag{23}$$

where $\boldsymbol{R}(t)$ denotes the weight change obtained by the learning rule in equation (2) and λ denotes a weight decay factor. Then, $\boldsymbol{W}$ and $\boldsymbol{B}$ converge to identical values because $\boldsymbol{W}(t+1) - \boldsymbol{B}(t+1) = (\boldsymbol{W}(t) + \Delta\boldsymbol{W}(t)) - (\boldsymbol{B}(t) + \Delta\boldsymbol{B}(t)) = (1-\lambda)(\boldsymbol{W}(t) - \boldsymbol{B}(t)) = (1-\lambda)^{t+1}(\boldsymbol{W}(0) - \boldsymbol{B}(0))$ goes to 0 when $0 < \lambda < 1$ [15, 16].

This logic also applies to the Activation Sharing algorithm. As shown in Fig. 3a, the Activation Sharing algorithm itself does not solve the weight transport problem because it still assumes symmetric forward and feedback paths ($\boldsymbol{W} = \boldsymbol{B}$). However, the algorithm could be further modified to avoid the weight transport problem by relaxing this constraint and only enforcing $\Delta\boldsymbol{W} = \Delta\boldsymbol{B}$ in training. Specifically, we assume asymmetric forward and feedback paths (i.e., $\boldsymbol{W} \neq \boldsymbol{B}$) and apply the Activation Sharing algorithm to both paths using the same shared activations as shown in Fig. 3b. Since the forward and feedback neurons share the same activations and errors, we can guarantee the following:

$$\Delta\boldsymbol{W}_{l+2} = \Delta\boldsymbol{B}_{l+2} = -\eta\, \dot{\boldsymbol{\delta}}_{l+2}\, \boldsymbol{h}_l^T \tag{24}$$

$$\Delta\boldsymbol{W}_{l+3} = \Delta\boldsymbol{B}_{l+3} = -\eta\, \dot{\boldsymbol{\delta}}_{l+3}\, \boldsymbol{h}_l^T \tag{25}$$

Therefore, the forward and feedback weights converge to the same values as discussed above, which enables training of deep neural networks while solving the weight transport problem. Moreover, by relaxing the constraints that each layer must use its own input activations to update the weights, bidirectional connections between forward and feedback neurons are not required any more. We named this algorithm Activation Sharing with Asymmetric Paths (ASAP) to distinguish it from the Activation Sharing algorithm using symmetric paths discussed in Section 3.2.

4 Experiments

To verify the effectiveness of the proposed ASAP algorithm, we measured its training performance for classification tasks on different datasets. We also implemented the backpropagation and other biologically plausible algorithms without bidirectional connections for comparisons. Since we aim at training deep convolutional networks using biologically plausible algorithms, we choose Alexnet [26], ResNet-18, and ResNet-34 [27] for experiments. The target models are trained on MNIST [28], SVHN [29], CIFAR-10, and CIFAR-100 [30] datasets for image classification. In training, we used the stochastic gradient descent (SGD) with momentum method [25]. In addition, the weight decay method [31] was adopted and the learning rate was adjusted through cosine annealing [32].

In all experiments, the block size k was set to 2, but we also experimented with $k = 4$ for ResNet-34. Since fully-connected layers can be trained sufficiently well with the Feedback Alignment, ASAP was applied only in convolutional layers while fully-connected layers are trained using Feedback Alignment. In ASAP, we update the weights of the layers in a block using the shared activations. However, when the channel size increases as it passes through a convolutional layer or the feature map size decreases after a maxpool or convolutional layer, the shared activations cannot be used for weight changes in later layers because of the difference in dimensions. In this case, we apply maxpool to the shared activation when activation map sizes differ, and concatenated along the channel dimension when channel sizes differed, to match its dimension with that of the corresponding layer. The input activation of the first layer, which is the network's input, has much lower number of channels than the following layers, so the input activation cannot be shared with later layers for training. Therefore, we use random fixed weights in the first layer and apply ASAP to the remaining convolutional layers. More details on the experiments are provided in the Appendix.

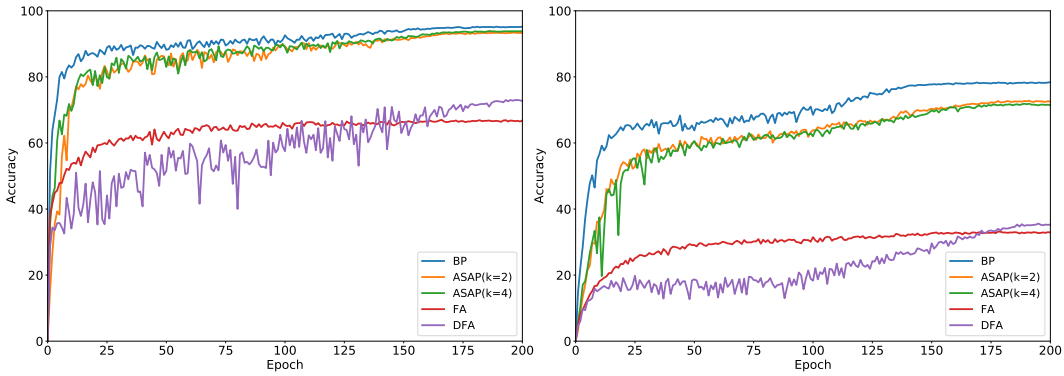
Table 1: Test accuracy of BP, FA, DFA, and ASAP in classification task

Dataset	Model	BP	FA	DFA	ASAP
MNIST	AlexNet	99.59	99.14	99.28	99.32
SVHN	AlexNet	94.64	82.21	87.42	88.04
	ResNet-18(nsc*)	96.14	84.91	85.64	93.17
	ResNet-18	96.29	85.08	85.56	94.86
CIFAR-10	AlexNet	90.58	67.92	73.85	78.25
	ResNet-18(nsc*)	94.70	58.39	75.46	83.44
	ResNet-18	94.93	72.47	76.0	92.19
	ResNet-34	95.18	66.99	73.02	93.97
CIFAR-100	AlexNet	63.61	33.32	35.38	46.99
	ResNet-18(nsc*)	77.34	37.11	37.48	51.72
	ResNet-18	77.74	38.15	38.51	68.86
	ResNet-34	78.41	33.01	35.6	72.81

*no shortcut

Good performance on deep convolutional networks Table 1 summarizes the performance of training algorithms on different datasets and target models. In summary, ASAP outperforms other biologically plausible algorithms (FA and DFA) in all cases, and shows very competitive performance especially on deep convolutional networks. When training the relatively simple AlexNet on the smallest MNIST dataset, FA, DFA, and ASAP all achieve good training performances and closely match BP. However, for more complex datasets such as CIFAR-10 and CIFAR-100, ASAP shows significantly better test accuracy than other biologically plausible algorithms. When compared to BP, ASAP shows some performance degradation, but interestingly, the performance gap between ASAP and BP is considerably smaller for deeper convolutional networks such as ResNet-18 and ResNet-34. ASAP only shows 1.43% accuracy drop for ResNet-18 trained on SVHN, while FA and DFA show more than 10% performance degradation. This trend is confirmed more clearly with complex datasets: for CIFAR-10, ResNet-18 and ResNet-34 trained with ASAP demonstrate over 90% test accuracy with less than 3% performance drop compared to BP, whereas FA and DFA only show up to 76% accuracy. For CIFAR-100, ASAP achieves 68.86% and 72.81% test accuracy for ResNet-18 and ResNet-34, respectively, which translate to less than 9% performance degradation over BP. Contrarily, FA and DFA exhibit a much lower test accuracy in the range of 33-39%. Generally, as a convolutional network get deeper, its performance gradually improves [33]. Nevertheless, the test accuracy of FA and DFA rarely improved by using more convolutional layers in the model. On the other hand, ASAP successfully exploits deep network structures and continues to improve test accuracy when having more layers in the model. Through these experiments, we can confirm that ASAP can train deep convolutional networks on complex datasets such as CIFAR-100 where other biologically plausible algorithm without bidirectional connections exhibit poor performance.

Effect of shortcut ResNet models employ shortcut connections to prevent vanishing and exploding gradients, enabling reliable training on deep convolutional networks [27]. For this reason, when training ResNet-18 through BP and DFA, having shortcut connections slightly increases test accuracy in SVHN, CIFAR-10, CIFAR-100. In the case of FA, shortcut connections increases the performance of ResNet-18 by 17.07% in CIFAR-10, but this effect is not noticeable in SVHN and CIFAR-100. On the other hand, the performance gap appears large in the case of ASAP, depending on the presence or absence of shortcuts in all datasets. This is because ASAP uses the activation of a lower layer to update the weights. In the absence of shortcut, the activation of a lower layer is only used to update the weights of ASAP and is not used at all in BP. However, if there is a shortcut, the activation of a lower layer is added to the output activation of the previous layer for weight update in BP. Therefore, the direction of weight changes of ASAP and BP becomes more similar when there is a shortcut. See Appendix for more detail.



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

Figure 4: Training performance comparisons for ResNet-34

Effect of block size ASAP removes bidirectional connections by sharing activations across the layers in a block, so the block size itself does not determine biological plausibility. As the block size increases, the number of layers updated using the same shared activations also increases. In other words, the amount of activations required to perform weight update in training decreases due to data reuse. Therefore, if we perform Activation Sharing with a larger block size k, it is possible to reduce

memory access when implemented in hardware. We compare the test accuracy of ResNet-34 for $k = 2$ and $k = 4$ in Fig. 4. On CIFAR-10, there was no performance degradation when using a larger block size ($k = 4$). The test accuracy is 93.97% when $k = 2$ and 93.9% when $k = 4$, while BP shows 95.18% test accuracy. On CIFAR-100, the test accuracy is 71.93% when $k = 4$ and 72.81% when $k = 2$, which translates to 0.88% degradation due to increasing block size from 2 to 4. However, using the block size of 4 considerably reduces memory access because only about 25% of activations are used compared to BP and 50% compared to $k = 2$ are used for weights update.

5 Discussion

It has been shown that our algorithm solves the weight transport problem without bidirectional connections. Of course, our method is not completely biologically plausible. Unlike the brain, our algorithm does not use spikes to propagate signals. However, in this work we focused on solving the weight transport problem while removing the need for bidirectional connections in the network, and our ASAP algorithm could be a candidate for explaining the feedback mechanism in the absence of a direct one-to-one feedback circuit in neural structures. Although feedforward-feedback loops may consist of only two neurons (direct one-to-one connections), observations in visual cortices show that reciprocal connections are generally spatially asymmetrical, consisting of a one-way connection [34, 35]. From this observation, one may conclude that reciprocal connections are formed through chains of connections, making multiple routes before the feedback signal is sent to the feedforward synapses [20]. The structural relaxation that our ASAP algorithm provides is in line with these observations, as feedback do not need to be formed through bidirectional paths but rather through a chain of feedback paths. Moreover, we showed that sharing activations in multiple layers enables training of deep convolutional networks on complex datasets without using bidirectional connections. While the proposed ASAP algorithm still exhibits some performance degradation compared to BP, it overcomes the limitation of other biologically plausible algorithms that they fail to train deep convolutional networks.

When implementing deep learning in hardware, a large number of MAC operations and external memory access are the processing bottlenecks [36]. In backpropagation, activations calculated in the forward path are required in the backward path to update the weights. For large models, it is impossible to store all the activations on chip; hence, it is inevitable to store the activations externally and load them back to the chip when updating the weights. Contrarily, in our algorithm, not all the activations obtained in the forward path are required for weight updates. It only uses a smaller part of activations when updating weights since they are shared across multiple layers. Therefore, the ASAP algorithm would be a good candidate for efficient hardware implementation since it could greatly reduce memory access overheads.

References

[1] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[2] Yoshua Bengio, Dong-Hyun Lee, Jörg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *CoRR*, 2015.

[3] Alexander G. Ororbia II, Patrick Haffner, David Reitter, and C. Lee Giles. Learning to adapt by minimizing discrepancy. *CoRR*, 2017.

[4] Sergey Bartunov, Adam Santoro, Blake A. Richards, Luke Marris, Geoffrey E. Hinton, and Timothy P. Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, 2018.

[5] James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in cognitive sciences*, 2019.

[6] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 1987.

[7] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 2016.

[8] Arild Nø kland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[9] Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Direct random target projection as a feedback-alignment algorithm with layerwise feedforward training. *stat*, 2019.

[10] Theodore H. Moskovitz, Ashok Litwin-Kumar, and L. F. Abbott. Feedback alignment in deep convolutional networks. *CoRR*, 2018.

[11] Julien Launay, Iacopo Poli, François Boniface, and Florent Krzakala. Direct feedback alignment scales to modern deep learning tasks and architectures. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[12] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. In *International Conference on Learning Representations*, 2020.

[13] Jordan Guerguiev, Konrad Kording, and Blake Richards. Spike-based causal inference for weight alignment. In *International Conference on Learning Representations*, 2020.

[14] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. In *International Conference on Learning Representations*, 2019.

[15] Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[16] John F Kolen and Jordan B Pollack. Backpropagation without weight transport. In *International Conference on Neural Networks*, volume 3. IEEE, 1994.

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

[18] Marion Langen, Egemen Agi, Dylan J Altschuler, Lani F Wu, Steven J Altschuler, and Peter Robin Hiesinger. The developmental rules of neural superposition in drosophila. *Cell*, 2015.

[19] Sanford L Palay and Victoria Chan-Palay. *Cerebellar cortex: cytology and organization*. 2012.

[20] Kathleen S Rockland. Elements of cortical architecture. In *Extrastriate cortex in primates*. 1997.

[21] Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Direct random target projection as a feedback-alignment algorithm with layerwise feedforward training. *stat*, 2019.

[22] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, 2015.

[23] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems*, 2018.

[24] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[25] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 1999.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[29] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[30] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[31] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, 1992.

[32] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[34] Kathleen S Rockland and Agnes 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] Penelope C Murphy and Adam M Sillito. Functional morphology of the feedback pathway from area 17 of the cat visual cortex to the lateral geniculate nucleus. *Journal of Neuroscience*, 1996.

[36] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We present in the abstract and introduction that our algorithm can solve the weight transport problem without bidirectional connection.
   (b) Did you describe the limitations of your work? [Yes] We mentioned that our algorithm is not completely biologically plausible yet and that there is some degradation compared to BP in Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Since we discuss a training algorithm for general neural networks, we believe that there is no negative societal impact directly related to our work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We presented all assumptions in Section 3.
   (b) Did you include complete proofs of all theoretical results? [Yes] The intuition behind our algorithm is provided in Section 3 and the full proof is made in Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code used for experiments is included in the supplemental material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The important details are described in Section 4, and other details could be found in the supplemental material.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] In Fig. 4, since we trained the networks for 200 epochs, adding an error bar to all data points may degrade the clarity of the graph.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] This information is included in the supplemental material.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cited the creators in Section 4.

(b) Did you mention the license of the assets? [Yes] The license information of the datasets is provided in the code.

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]