LEARNING WITH AUXILIARY ACTIVATION FOR MEMORY-EFFICIENT TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

While deep learning has achieved great success in various fields, a large amount of memory is necessary to train deep neural networks, which hinders the development of massive state-of-the-art models. The reason is the conventional learning rule, backpropagation, should temporarily store input activations of all the layers in the network. To overcome this, recent studies suggested various memory-efficient implementations of backpropagation. However, those approaches incur computational overhead due to the recomputation of activations, slowing down neural network training. In this work, we propose a new learning rule which significantly reduces memory requirements while closely matching the performance of backpropagation. The algorithm combines auxiliary activation with output activation during forward propagation, while only auxiliary activation is used during backward propagation instead of actual input activation to reduce the amount of data to be temporarily stored. We mathematically show that our learning rule can reliably train the networks whose loss landscape is convex if the auxiliary activation satisfies certain conditions. Based on this observation, we suggest candidates of auxiliary activation that satisfy those conditions. Experimental results confirm that the proposed learning rule achieves competitive performance compared to backpropagation in various models such as ResNet, Transformer, BERT, ViT, and MLP-Mixer.

1 INTRODUCTION

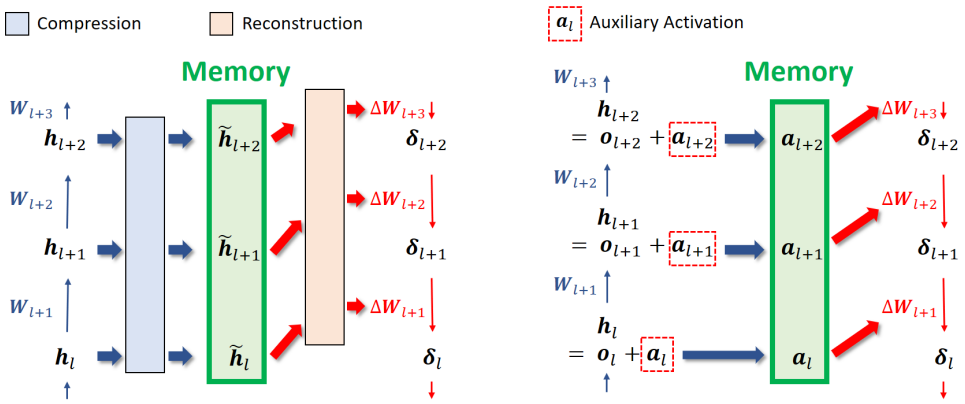
Backpropagation (1) is an essential learning rule to train deep neural networks and has proven its outstanding performance in diverse models and tasks. In general, the wider and deeper the deep learning model, the better the training performance (2). However, increasing model size unavoidably requires larger memory in training hardware such as GPU (3; 4). This is because backpropagation needs to temporarily store input activations of all the layers in the network generated in forward propagation as they are later used to update weights in backward propagation. Consequently, state-of-the-art deep learning models require substantial memory resources due to the large amount of input activation to store. In order to train very deep models with limited hardware resources, the batch size may be reduced (3; 4) or many GPUs can be used in parallel (5; 6; 7; 8; 9; 10; 11). However, reducing the batch size causes a long training time, and the advantage of batch normalization (12) disappears. Also, training huge models such as GPT-3 (13) still requires expensive GPU clusters with thousands of GPUs and incurs high I/O costs even with parallelism.

Recently, a wide range of algorithms have been proposed to alleviate this memory requirement. For instance, gradient checkpointing (14; 15; 16; 17; 18) does not store all input activations, but only some of them during forward propagation. Then, it restores the unsaved input activations through recomputation in backward propagation. In-place activated batch normalization (19) merges a batch normalization layer and a leaky ReLU layer and stores the output activation of the merged layer in forward propagation. In backward propagation, the layer input can be reconstructed for training because the leaky ReLU function is reversible. Similarly, RevNet (20) and Reformer (21) employ reversible neural network architectures, which allow for calculating input activation from output activation in backward propagation. Gradient checkpointing and reversible network structures reduce training memory space because they partially store input activations (e.g., input activations of selected layers). However, these methods incur additional computational overhead because the unstored input activations must be recomputed during backward propagation. Alternatively, algo-

rithms to approximate activation have been suggested (22; 23; 24; 25; 26; 27; 28), but they suffer from performance degradation or slow down training due to additional computations to quantize and dequantize activations. TinyTL (29) entirely avoids saving activations by updating only bias parameters while fixing weight parameters. However, it is only applicable to fine-tuning of a pre-trained model.

In this study, we propose a new learning rule, Auxiliary Activation Learning, which can significantly reduce memory requirements for training deep neural networks without sacrificing training speed. We first introduce the concept of auxiliary activation in the training process. Auxiliary activations are combined with output activations and become the input activation of the next layer when processing forward propagation, but only the auxiliary activations are temporarily stored instead of actual input activation for updating weights in backward propagation. To justify our algorithm, we prove that an alternate type of input activation could reliably train neural networks whose landscapes are convex if auxiliary activation satisfies certain conditions. Then, we propose multiple candidates of auxiliary activations which meet this criterion. Experimental results demonstrate that the proposed algorithm not only succeeds in training ResNet models (30) on ImageNet (31) with similar performance to backpropagation, but is also suitable for training other neural network architectures such as Transformer (32), BERT (33), ViT (34), and MLP-Mixer (35).

2 Auxiliary Activation Learning



(a) Conventional memory-saving algorithms (b) Auxiliary Activation Learning

Figure 1: Conventional memory-efficient training algorithms and Auxiliary Activation Learning.

2.1 Memory requirements of backpropagation

To understand why backpropagation uses a large memory space for training, we describe how it trains a deep neural network. The training process of backpropagation can be expressed by the equations below.

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

$$\boldsymbol{\delta}_l = \boldsymbol{W}_{l+1}^T \boldsymbol{\delta}_{l+1} \odot \phi^{'}(\boldsymbol{y}_l) \tag{2}$$

$$\Delta \boldsymbol{W}_{l+1} = -\eta \boldsymbol{\delta}_{l+1}\, \boldsymbol{h}_l^T \tag{3}$$

Here $\boldsymbol{h}_l$, $\boldsymbol{W}_{l+1}$, and $\boldsymbol{b}_{l+1}$ denote the input activation, weight, and bias of the hidden layer $l+1$, respectively. ϕ, $\boldsymbol{\delta}$, and η represent the nonlinear function, backpropagated error, and learning rate, respectively. $\boldsymbol{y}_l$ is the output of a linear or convolutional layer (i.e., $\boldsymbol{y}_l = \boldsymbol{W}_l\, \boldsymbol{h}_{l-1} + \boldsymbol{b}_l$), which becomes the input of the nonlinear function ϕ. In forward propagation, the input propagates through the network following equation (1). During this process, the generated input activations are stored in memory. In backward propagation, the error is backpropagated through the network based on equation (2). The weights are updated using the backpropagated errors and the activations stored

in memory (equation (3)). Due to the time gap between forward and backward computations, all activations obtained in forward propagation must be temporarily stored in memory as they are later used for backward propagation. Therefore, as the model size increases, the amount of activation to be stored in memory increases accordingly. To alleviate this issue, gradient checkpointing (14; 15; 16; 17; 18) stores only some of the activations in memory during forward propagation, and the rest of the activations are restored during backpropagation by recomputing equation (1). Similarly, activation compression methods (22; 23; 24; 26; 27; 28) quantize activations before saving them in memory to reduce memory requirements, and they are dequantized during backward propagation. However, these methods incur computational overhead and increase training time due to additional processes during forward and backward propagation as depicted in Fig. 1a.

2.2 Auxiliary Activation Learning: store auxiliary activation instead of exact activation

For memory-efficient training with minimal overhead, here we propose a new learning rule, *Auxiliary Activation Learning*. First, we introduce the concept of auxiliary activation in the training process. Then, the algorithm trains the model following the equations below.

$$\boldsymbol{h}_l = \boldsymbol{o}_l + \boldsymbol{a}_l \tag{4}$$

$$\boldsymbol{o}_{l+1} = \phi(\boldsymbol{W}_{l+1}\,\boldsymbol{h}_l + \boldsymbol{b}_{l+1}) \tag{5}$$

$$\boldsymbol{\delta}_l = \boldsymbol{W}_{l+1}^T \boldsymbol{\delta}_{l+1} \odot \phi^{'}(\boldsymbol{y}_l) \tag{6}$$

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

Here $\boldsymbol{a}_l$ represents the newly introduced auxiliary activation of layer l. In forward propagation, each layer takes the sum of auxiliary activation $\boldsymbol{a}_l$ and the previous layer's output activation $\boldsymbol{o}_l$ as input activation $\boldsymbol{h}_l$ (equation (4)). The generated input activation propagates through the layer following equation (5). In backward propagation, the error $\boldsymbol{\delta}_{l+1}$ is backpropagated by equation (6). Finally, the weights are updated only using auxiliary activation $\boldsymbol{a}_l$ instead of the actual input activation $\boldsymbol{h}_l$ following equation (7). The learning process of our algorithm is depicted in Fig. 1b. Now the memory needs to store the auxiliary activations $\boldsymbol{a}_l$, not real activations $\boldsymbol{h}_l$. Consequently, if the auxiliary activation is reused (e.g., $\boldsymbol{a}_l = \boldsymbol{a}_{l+1}$) or the amount by auxiliary activation itself is small, the required memory space can be reduced accordingly. Furthermore, it does not require additional computations, avoiding training time increases.

The intuition behind only using auxiliary activation for weight updates is that the direction of a weight update obtained using auxiliary activation could be similar to that of backpropagation if we carefully choose auxiliary activation. More specifically, we use the sum of the auxiliary activation $\boldsymbol{a}_l$ and the previous layer's output activation $\boldsymbol{o}_l$ as the input activation $\boldsymbol{h}_l$ (equation (4)). If we take an existing value (e.g., activation from a different layer) as the auxiliary activation, this can be seen as making an additional connection to the layer. If we update weights using backpropagation as in equation (3), the input activation $\boldsymbol{h}_l$ generated in forward propagation will be used for updating weights: $\Delta \boldsymbol{W}_{l+1}^{BP} = -\eta \boldsymbol{\delta}_{l+1}\,\boldsymbol{h}_l^T = -\eta \boldsymbol{\delta}_{l+1}\,(\boldsymbol{o}_l + \boldsymbol{a}_l)^T = -\eta \boldsymbol{\delta}_{l+1} \boldsymbol{o}_l^T - \eta \boldsymbol{\delta}_{l+1} \boldsymbol{a}_l^T$. Consequently, the weight gradient obtained by using our algorithm can be expressed by $\Delta\,\boldsymbol{W}_{l+1}^{AAL} = \Delta\,\boldsymbol{W}_{l+1}^{BP} + \eta \boldsymbol{\delta}_{l+1} \boldsymbol{o}_l^T$. Therefore, it is expected that the weight update direction of our algorithm can be similar to that of backpropagation if the effect of the second term is ignorable, which will result in proper training of the neural network. In the next section, we prove that this assumption can be realized if we choose the auxiliary activation appropriately.

3 Analysis of Auxiliary Activation Learning

3.1 How and when using alternative activation can train deep neural networks

This section shows how learning with an alternative form of activation can make the loss of nonlinear networks converge to a minimum value if a specific condition is satisfied. For mathematical analysis, we consider a loss function $f(x)$ which is convex, differentiable, and Lipschitz continuous. $\boldsymbol{h}$ and $\boldsymbol{r}$ are column vectors representing real activations and random alternative activations, respectively. $\boldsymbol{\delta}$ is a column vector of backpropagated errors, and $\boldsymbol{W}$ is a flattened vector of weights. When weights

are updated by backpropagation using exact activations $\boldsymbol{h}$,

$$\Delta \boldsymbol{W}^t = \nabla f(\boldsymbol{W}^t) = \mathrm{Vec}(\boldsymbol{\delta}_1 \boldsymbol{h}_0^T, \ldots, \boldsymbol{\delta}_L \boldsymbol{h}_{L-1}^T) \tag{8}$$

where t denotes the current epoch. Contrarily, now we assume that the weights are updated using alternative activations $\boldsymbol{r}_l$ as below, which represents a generalization of our algorithm.

$$\Delta \boldsymbol{W}^t = \mathrm{Vec}(\boldsymbol{\delta}_1 \boldsymbol{r}_0^T, \ldots, \boldsymbol{\delta}_L \boldsymbol{r}_{L-1}^T) \neq \nabla f(\boldsymbol{W}^t) \tag{9}$$

$$\boldsymbol{W}^{t+1} = \boldsymbol{W}^t - \eta \Delta \boldsymbol{W}^t \tag{10}$$

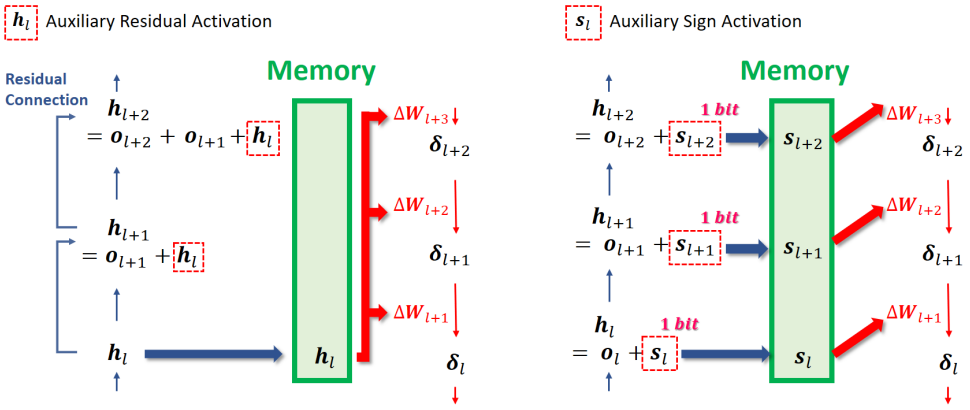
Theorem 1. *If $\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l) > 0$ in all layers and the step size η satisfies $0 \leq \eta \leq \frac{1}{L}$ where $L = \nabla^2 f$, then the loss function $f(\boldsymbol{W})$ converges to a minimum value.*

Theorem 1 can be proven by applying equations (9)-(10) to inequality for a convex function with an L-Lipschitz continuous gradient. A detailed proof is provided in Appendix A. Because condition $0 \leq \eta \leq \frac{1}{L}$ can be easily satisfied if we use a small step size, $\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l) \geq 0$ is the most important condition for successful learning that alternative activations should satisfy. We name this condition *Learning Criterion*. Furthermore, the larger the value of $\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l)$, the better the network converges because the amount of loss reduction in each epoch (K^n in Appendix A) increases. One interesting observation is that $\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l) = 2\boldsymbol{r}_l \cdot \boldsymbol{h}_l - \|\boldsymbol{r}_l\|^2 = -\|\boldsymbol{h}_l - \boldsymbol{r}_l\|^2 + \|\boldsymbol{h}_l\|^2$ reaches its maximum when $\boldsymbol{r}_l = \boldsymbol{h}_l$, which translates to a conventional backpropagation algorithm. For convenience, we normalize the learning criterion inequality with respect to its maximum value $\|\boldsymbol{h}_l\|^2$ and define a learning indicator as below.

$$\textit{Learning Indicator} \equiv \frac{\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l)}{\|\boldsymbol{h}_l\|^2} \tag{11}$$

In summary, alternative activation can train the network if the learning indicator in equation (11) is positive. Moreover, the larger the learning indicator, the better the training performance. Finally, the training performance is maximized when the learning indicator is 1 ($\boldsymbol{r}_l = \boldsymbol{h}_l$, backpropagation).

3.2 Constructing Auxiliary Activations



(a) Auxiliary Residual Activation (b) Auxiliary Sign Activation

Figure 2: Two candidates of auxiliary activation.

Theorem 1 suggests that using an alternate type of activation in calculating weight gradient could still reliably train the neural network. However, in reality, it is cumbersome to meet the learning criterion if we use an arbitrary value as the alternate activation $\boldsymbol{r}_l$ (see Appendix B). To address this, we modify the forward propagation path by adding auxiliary activation to the input activation as in equation (4) in the proposed Auxiliary Activation Learning. Then, the input activation $\boldsymbol{h}_l = \boldsymbol{o}_l + \boldsymbol{a}_l$ and the alternate activation $\boldsymbol{r}_l = \boldsymbol{a}_l$ share a common term $\boldsymbol{a}_l$, making it easier to meet the learning criterion. Also, if we employ some values already existing in the network as auxiliary activation $\boldsymbol{a}_l$,

this modification can be regarded as introducing an additional connection in the layer. The learning indicator of Auxiliary Activation Learning is expressed by

$$Learning\ Indicator = \frac{\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l)}{\|\boldsymbol{h}_l\|^2} = \frac{\boldsymbol{a}_l^T(2\boldsymbol{o}_l + \boldsymbol{a}_l)}{\|\boldsymbol{h}_l\|^2} \tag{12}$$

by applying equation (4) to equation (11). For successful learning, the value of the learning indicator in equation (12) should be positive. To satisfy this learning criterion, we suggest two candidates of auxiliary activation as below.

Auxiliary Residual Activation First, we can use the previous layer's input activations as auxiliary activations and combine them with output activations, which is identical to having a residual connection. Then, the learning process is expressed by

$$\boldsymbol{o}_{l+1} = \phi(\boldsymbol{W}_{l+1}(\boldsymbol{o}_{l+1} + \boldsymbol{h}_l) + \boldsymbol{b}_{l+1}) \tag{13}$$

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

Using auxiliary residual activations can save training memory if we only store $\boldsymbol{h}_l$ and use it for updating $\boldsymbol{W}_{l+1}$ using backpropagation as well as for updating $\boldsymbol{W}_{l+2}$ and $\boldsymbol{W}_{l+3}$ using Auxiliary Activation Learning as depicted in Fig. 2a and equation (14). Contrarily, backpropagation needs to store $\boldsymbol{h}_l$, $\boldsymbol{h}_{l+1}$, and $\boldsymbol{h}_{l+2}$ for updating these three layers.

Auxiliary Sign Activations Another option is using the sign of the output activation as auxiliary activations. Then, the weight update process is represented by the equations below.

$$\boldsymbol{o}_{l+1} = \phi(\boldsymbol{W}_{l+1}(\boldsymbol{o}_l + \epsilon\text{sign}(\boldsymbol{o}_l)) + \boldsymbol{b}_{l+1}) \tag{15}$$

$$\Delta\boldsymbol{W}_{l+1} = -\eta\boldsymbol{\delta}_{l+1}\,\epsilon\text{sign}(\boldsymbol{o}_l)^T \tag{16}$$

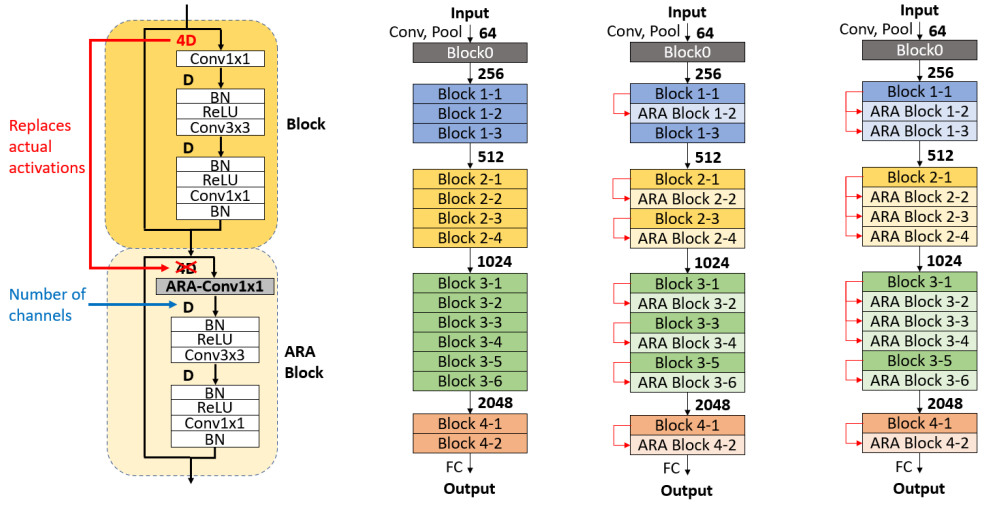
The auxiliary sign activation ($\boldsymbol{s}_l = \epsilon\text{sign}(\boldsymbol{o}_l)$) can be guaranteed to satisfy the learning criterion. Because $2\boldsymbol{a}_l^T\boldsymbol{o}_l = 2\epsilon\text{sign}(\boldsymbol{o}_l)^T\boldsymbol{o}_l \geq 0$, the learning indicator in equation (12) is always positive. Using auxiliary sign activation can enable memory-efficient training by storing only a 1-bit sign of activation instead of actual input activation in high precision as shown in Fig. 2b. In equations (15) and (16), we multiply the auxiliary activation by a hyperparameter ϵ to make the two types of activation comparable in magnitude since the output of the sign function is -1 or 1.

4 Experimental Results

In experiments, we trained various deep neural networks using the proposed learning rule and compared its training performance to that of backpropagation. We observed if the selected auxiliary activations satisfied the learning criterion during training and also measured the amount of memory savings and training time. Our algorithm was used to train ResNet (30) as well as transformer and its variants (32; 33; 34; 35), which are widely used architectures in vision and NLP-related applications.

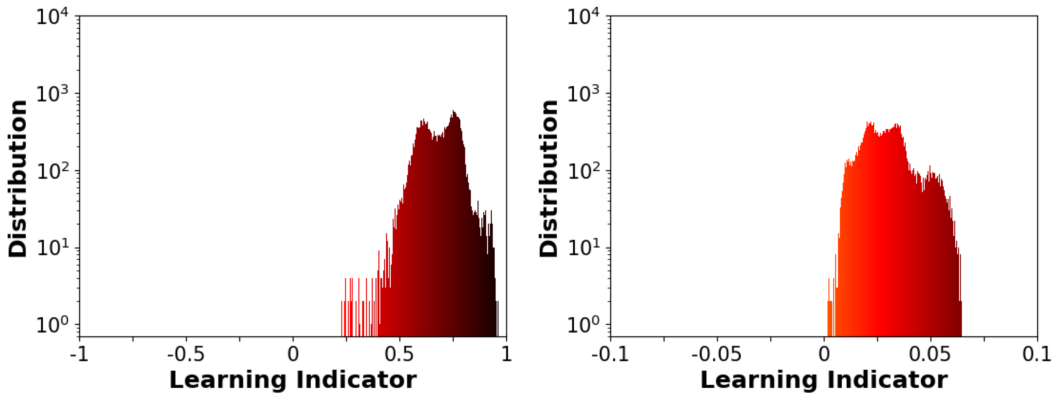
4.1 Training ResNet

We applied the proposed Auxiliary Activation Learning to ResNet training using two types of auxiliary activation as discussed above: Auxiliary Residual Activation (ARA) and Auxiliary Sign Activation (ASA). For ARA, we replaced the first convolutional layer, which receives a residual connection from the previous block, with an ARA-Conv layer which performs weight updates using auxiliary residual activations instead of actual activations (Fig. 3a). Hence, memory overhead can be reduced for these ARA-Conv layers. The blocks in ResNet (30) can be categorized into different sets based on their dimension. For example, ResNet-50 consists of blocks with an input channel dimension of 256, 512, 1024, or 2048 as displayed in Fig. 3b. In experiments with Auxiliary Activation Learning, the blocks are replaced with ARA blocks with different strides ("sharing stride") in each set of blocks as shown in Fig. 3c and 3d. For instance, ARA(3,4,4,2) has a sharing stride of 3, 4, 4, and 2 in the sets with a dimension of 256, 512, 1024, and 2048, respectively (Fig. 3d). Therefore, the larger the sharing stride, the more blocks that are replaced by ARA blocks and the larger the amount of memory savings. Similarly, for experiments with auxiliary sign activation, the first convolutional layer was replaced by an ASA-Conv layer, which employs auxiliary sign activation



(a) ARA Block (b) BP (c) ARA(2,2,2,2) (d) ARA(3,4,4,2)

Figure 3: Training ResNet-50 using Auxiliary Residual Activation (ARA).



(a) ARA(2,2,2,2) (b) ASA(2,2,2,2)

Figure 4: Distributions of learning indicator values in ResNet-18 trained on CIFAR-10.

for weight updates. Other details of the experiments including hyperparameters can be found in Appendix C.

We first trained ResNet-18 on CIFAR-10, CIFAR-100 (36), and Tiny-ImageNet (37) by using BP, ARA, and ASA to confirm the effectiveness of Auxiliary Activation Learning and the correlation between training performance and the value of the learning indicator. Fig. 4a shows that ARA not only satisfies the learning criterion (i.e., keeps learning indicator values positive), but also results in relatively large learning indicator values close to 1. This is consistent with the training performance displayed in Table 1; ARA achieves high accuracy and even outperforms backpropagation in some cases. Fig. 4b shows that ASA satisfies the learning criterion, but the values of the indicator are much lower than that of ARA. Accordingly, ASA achieves slightly lower accuracy than BP and ARA. In addition, the experiments in Appendix B show that the performance is significantly degraded when the learning criterion is unsatisfied due to using modifications of auxiliary activation. These results suggest that the proposed learning indicator has a strong correlation with training performance.

In the next experiment, we applied ARA to training ResNet-50 and ResNet-152 on ImageNet with various sharing stride values. Experimental results are summarized in Table 2. ARA achieved competitive performance close to BP in all cases, and ARA(3,4,2,2) even achieved slightly higher

Table 1: Test accuracy of ResNet-18 trained on CIFAR-10, CIFAR-100 and Tiny ImageNet.

	Dataset	BP	ARA(2,2,2,2)	ASA(2,2,2,2)
Accuracy (%)	CIFAR-10	94.77	94.81	94.76
	CIFAR-100	75.81	75.49	75.31
	Tiny ImageNet	58.43	58.46	57.01

performance than BP in ResNet-152. Since the mechanism of our algorithm is entirely different from other memory-saving techniques, our algorithm can be employed in addition to those conventional approaches for even larger memory savings. For experiments, we trained the same networks using gradient checkpointing (GCP) (16) and ActNN (25) which is per-group activation compression training. Then, we additionally applied our ARA to these learning rules. More specifically, GCP was applied to the layers that did not employ ARA, whereas ActNN was applied to all layers including ARA_Conv layers (see Appendices C.1 and C.2 for details). Experimental results clearly show that our algorithm allows for larger memory savings with a very small increase in training time. This is in contrast with GCP and ActNN, which noticeably slows down training due to additional computations for reconstructing activations during backward propagation. For instance, applying ActNN alone to ResNet-50 reduces memory space by 11.8x while increasing training time from 18m to 49m compared to BP. Contrarily, additionally applying ARA(3,4,6,2) increases the compression rate to 15.01x without impacting training speed.

Table 2: Test Accuracy, compression rate (bracketed), and training time (italic) of one epoch in ResNet training on ImageNet with 512 batch size and six RTX-3090 GPUs.

Models	Baseline	-	ARA (2,2,2,2)	ARA (3,4,2,2)	ARA (3,4,4,2)	ARA (3,4,6,2)
ResNet-50	BP	**75.22** - *17m 35s*	74.96 (1.12x) *17m 59s*	74.92 (1.2x) *17m 54s*	74.85 (1.21x) *17m 55s*	74.5 (1.22x) *18m 3s*
	GCP	**75.22** (2.2x) *36m 42s*	74.96 (2.88x) *37m 2s*	74.92 (3.44x) *37m 9s*	74.85 (3.54x) *36m 58s*	74.52 (3.66x) *37m 14s*
	ActNN	75.03 (11.8x) *48m 48s*	74.93 (14x) *48m 48s*	74.87 (14.6x) *48m 10s*	74.65 (14.8x) *47m 58s*	74.5 (**15.01**x) *47m 08s*
ResNet-152	BP	77.38 - *35m 37s*	77.14 (1.16x) *36m 17s*	**77.41** (1.21x) *36m 24s*	76.84 (1.27x) *36m 11s*	76.64 (1.29x) *36m 30s*
	GCP	77.38 (2.1x) *1h 18m*	77.14 (2.92x) *1h 19m*	**77.41** (3.26x) *1h 19m*	76.84 (3.75x) *1h 19m*	76.64 (**3.95**x) *1h 20m*

4.2 Training Transformer, ViT, and MLP-Mixer

Transformer (32) achieved state-of-the-art performance in NLP, and the models inspired by Transformer such as BERT (33), ViT (34), and MLP-Mixer (35) are producing promising results in computer vision. The structure of these models is quite distinct from ResNet models (30; 38). Hence, we applied our Auxiliary Activation Learning to Transformer and its variants to verify the scalability of our algorithm. We trained Transformer on IWSLT (39) and BERT-Large on MNLI (40) datasets. For ViT-Large and MLP-Mixer-Large, we used CIFAR-100 (36). Note that ARA was not used in this experiment since residual connections are connected to layer normalization layers (41) rather than to fully-connected layers, whereas ARA requires residual connections to directly connect to fully-connected layers to realize equation (13). Instead, we applied ASA to linear layers in multi-head attention and feedforward network. Fig. 5 depicts how ASA is applied to those layers. Since ASA-Linear layers store only 1-bit auxiliary sign activation instead of exact activation, memory overhead can be mitigated. We trained networks with BP and Mesa (27) which is activation compression training for Transformer-like models. Then, we additionally applied ASA to these algorithms (see

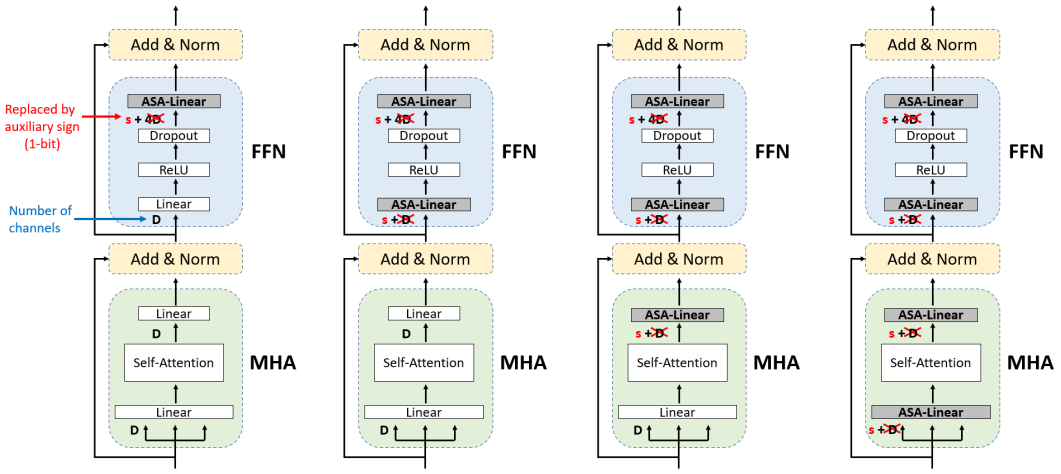


Figure 5: Transformer-like networks trained using Auxiliary Sign Activations (ASA). MHA and FFN represent multihead attention and feedforward network, respectively.

Appendix C.2 for details). Mesa stores 8-bit quantized activation for linear layers; hence, applying ASA to these linear layers can further reduce memory space as ASA-Linear layers only store 1-bit sign activations. Because Mesa is designed for the networks with self-attention, it is not applied to MLP-Mixer which does not employ self-attention layers.

Experimental results are displayed in Table 3. When we apply ASA only, it closely matches or even outperforms BP while using smaller memory space. Since ASA needs to extract 1-bit sign from actual activation to construct auxiliary activation, training time is slightly increased. However,

Table 3: Test/validation scores, compression rate (bracketed), and training time (italic) of one epoch of Transformer, BERT, ViT, and MLP-Mixer training with 4096, 32, 512, and 256 batch sizes, respectively.

Models	Dataset	Baseline	-	ASA1	ASA2	ASA3	ASA4
Trans-former	IWSLT	BP	35.23 (-) *3m 15s*	35.44 (1.1x) *3m 21s*	34.87 (1.1x) *3m 23s*	34.9 (1.2x) *3m 26s*	35.02 (1.3x) *3m 38s*
		Mesa	**35.45** (1.6x) *5m 52s*	34.74 (1.7x) *5m 47s*	35.19 (1.7x) *5m 44s*	35.11 (1.7x) *5m 36s*	34.84 (**1.8**x) *5m 8s*
Bert-Large	MNLI	BP	86.52 (-) *2h 37m*	**86.65** (1.1x) *2h 40m*	86.49 (1.2x) *2h 40m*	86.42 (1.3x) *2h 41m*	86.39 (1.3x) *2h 41m*
		Mesa	86.32 (2.1x) *3h 51m*	86.37 (2.3x) *3h 39m*	86.29 (2.4x) *3h 35m*	86.54 (2.4x) *3h 32m*	86.17 (**2.5**x) *3h 24m*
ViT-Large	CIFAR-100	BP	92.93 (-) *2m 46s*	92.81 (1.3x) *2m 48s*	92.84 (1.5x) *2m 50s*	**92.97** (1.5x) *2m 51s*	92.65 (1.6x) *2m 52s*
		Mesa	92.89 (3.0x) *4m 46s*	92.75 (3.5x) *4m 15s*	92.72 (3.6x) *3m 57s*	92.95 (3.7x) *3m 49s*	92.84 (**4.3**x) *3m 33s*
Mixer-Large	CIFAR-100	BP	91.62 (-) *3m 39s*	91.45 (1.3x) *3m 47s*	91.72 (1.4x) *3m 50s*	91.66 (1.7x) *3m 57s*	**91.91** (**2.0**x) *4m 1s*

it is negligible compared to Mesa which requires quantization and dequantization during training. For example, when training BERT-Large on MNLI, BP and ASA consumed 2h 37m and 2h 41m, respectively, while Mesa takes 3h 51m. We can maximize memory savings by applying Mesa and our algorithm (ASA) simultaneously. If we train ViT-Large on CIFAR-100 using Mesa only, we can obtain a 3.0x compression rate. On the other hand, applying both Mesa and ASA4 simultaneously results in the compression rate of 4.3x. Furthermore, the training time is reduced by 1m 13s because linear layers are replaced with ASA-Linear layers, avoiding costly quantization and dequantization processes for these layers.

4.3 Usage case of Auxiliary Activation Learning

The proposed Auxiliary Activation Learning algorithm allows for training deep neural networks on the systems with limited hardware resources by alleviating memory overheads with minimal computational cost. For example, one may use a larger batch size using the same number of GPUs, which enables faster training. For experiments, we trained various networks using only three RTX-3090 GPUs to confirm the effectiveness of our algorithm. The experimental results for ResNet-152 and ViT-Large are displayed in Tables 4 and 5. Using memory-saving techniques allow for using a larger batch size for a given GPU memory space. Interestingly, although conventional algorithms such as GCP, ActNN, and Mesa enables a larger batch size, it does not translate to faster training speed due to additional computations. In contrast, our ARA and ASA algorithms take full advantage of larger batch size and improve training speed.

Table 4: Maximum batch size and training time of ResNet-152 on ImageNet. The sharing stride of ARA is (3,4,6,2).

	BP	GCP	ActNN	ARA	GCP + ARA	ActNN + ARA
Max batch size	350	700	1950	450	1250	**2200**
Training time	49m	1h 4m	56m	**47m**	57m	52m

Table 5: Maximum batch size and training time of ViT-Large on CIFAR-100.

	BP	Mesa	ASA4	Mesa + ASA4
Max batch size	600	1600	900	**2100**
Training time	1m 53s	2m 9s	**1m 49s**	1m 59s

5 Discussion

In this work, we proposed a new learning rule, Auxiliary Activation Learning, which effectively alleviates memory overhead in deep neural network training. The algorithm employs an alternate form of activations for weight updates, and hence reduces the amount of data to be stored in memory without additional computations. We proved that the algorithm can train neural networks whose landscape is convex if an alternative form of activations satisfies a certain condition and also provided a performance indicator that can anticipate the training performance of the selected auxiliary activation type. Experimental results confirmed that the proposed algorithm successfully trains various deep neural networks for vision and NLP tasks with smaller memory without training speed reduction. Furthermore, since our algorithm is orthogonal to existing memory saving algorithms, it is possible to increase the compression rate as much as possible without reduction in training speed by applying existing algorithms and our algorithm simultaneously.

Since our study aims to save training memory in linear and convolutional layers, it currently cannot be applied to layers that perform other types of operations such as normalization and activation functions. In future work, we will try to address those issues to realize a more memory-efficient learning algorithm and expand its applicability.

REPRODUCIBILITY

We presented the rationale for our algorithm in Section 3. And it was proven through the experimental results in Section 4. We have detailed various points for experimental reproduction in 4 and Appendix C. Furthermore, we support code to reproduce our results as follow: https://github.com/viqpldem/Auxiliary_Activation_Learning.

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] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[3] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. High-performance semantic segmentation using very deep fully convolutional networks. *CoRR*, abs/1604.04339, 2016.

[4] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE*, pages 2242–2251, 2017.

[5] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alexander J. Smola, Alexander L. Strehl, and Vishy Vishwanathan. Hash kernels. In *AISTATS*, volume 5 of *JMLR Proceedings*, pages 496–503, 2009.

[6] Gideon Mann, Ryan T. McDonald, Mehryar Mohri, Nathan Silberman, and Dan Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *NIPS*, pages 1231–1239, 2009.

[7] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS*, pages 1232–1240, 2012.

[8] Ryan T. McDonald, Keith B. Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pages 456–464, 2010.

[9] Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. Parallelized stochastic gradient descent. In *NIPS*, pages 2595–2603, 2010.

[10] Alekh Agarwal and John C. Duchi. Distributed delayed stochastic optimization. In *NIPS*, pages 873–881, 2011.

[11] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *J. Mach. Learn. Res.*, 15(1):1111–1133, 2014.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[13] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Neurips*, 2020.

[14] Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, 26(1):19–45, 2000.

[15] James Martens and Ilya Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 479–535. 2012.

[16] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.

[17] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In *NIPS*, pages 4125–4133, 2016.

[18] Jianwei Feng and Dong Huang. Optimal gradient checkpoint search for arbitrary computation graphs. In *IEEE*, pages 11433–11442, 2021.

[19] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *CVPR*, pages 5639–5647, 2018.

[20] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. In *NIPS*, pages 2214–2224, 2017.

[21] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

[22] Ayan Chakrabarti and Benjamin Moseley. Backprop with approximate activations for memory-efficient network training. In *NIPS*, pages 2426–2435, 2019.

[23] R. David Evans, Lufei Liu, and Tor M. Aamodt. JPEG-ACT: accelerating deep learning via transform-based lossy compression. In *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Valencia, Spain, May 30 - June 3, 2020*, pages 860–873. IEEE, 2020.

[24] Fangcheng Fu, Yuzheng Hu, Yihan He, Jiawei Jiang, Yingxia Shao, Ce Zhang, and Bin Cui. Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3304–3314. PMLR, 2020.

[25] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael W. Mahoney, and Joseph Gonzalez. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1803–1813. PMLR, 2021.

[26] R. David Evans and Tor M. Aamodt. AC-GC: lossy activation compression with guaranteed convergence. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27434–27448, 2021.

[27] Zizheng Pan, Peng Chen, Haoyu He, Jing Liu, Jianfei Cai, and Bohan Zhuang. Mesa: A memory-saving training framework for transformers. *CoRR*, abs/2111.11124, 2021.

[28] Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2021.

[29] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE*, pages 770–778, 2016.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[34] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[35] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. In *Neurips*, pages 24261–24272, 2021.

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

[37] Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015.

[38] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[39] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, Roldano Cattoni, and Marcello Federico. The IWSLT 2016 evaluation campaign. In *IWSLT*, 2016.

[40] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.

[41] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[42] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[43] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICLR*, 2017.

[44] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[45] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[46] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? In *NeurIPS*, pages 4696–4705, 2019.

[47] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[48] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):1–10, 2016.

[49] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1037–1045, 2016.

[50] Qianli Liao, Joel Z. Leibo, and Tomaso A. Poggio. How important is weight symmetry in backpropagation? In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 1837–1844. AAAI Press, 2016.

APPENDICES

A PROOF OF THEOREMS

Theorem 1. *If $\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l) > 0$ in all layers and the step size η satisfies $0 \leq \eta \leq \frac{1}{L}$ where $L = \nabla^2 f$, then the loss function $f(\boldsymbol{W})$ converges to a minimum value.*

Proof. Since $f(x)$ is convex, differentiable, and Lipschitz continuous, it satisfies $f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}L\|\boldsymbol{x} - \boldsymbol{y}\|^2$. By substituting $\boldsymbol{W}^t$ for x and $\boldsymbol{W}^{t+1}$ for y, we obtain

$$\begin{aligned}
f(\boldsymbol{W}^{t+1}) &\leq f(\boldsymbol{W}^t) + \nabla f(\boldsymbol{W}^t)^T(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t) + \frac{1}{2}L\|\boldsymbol{W}^{t+1} - \boldsymbol{W}^t\|^2 \\
&= f(\boldsymbol{W}^t) + \nabla f(\boldsymbol{W}^t)^T(-\eta\Delta\boldsymbol{W}^t) + \frac{1}{2}L\|-\eta\Delta\boldsymbol{W}^t\|^2 \\
&= f(\boldsymbol{W}^t) - \eta[\nabla f(\boldsymbol{W}^t)^T\Delta\boldsymbol{W}^t - \frac{1}{2}\eta L\|\Delta\boldsymbol{W}^t\|^2] \\
&= f(\boldsymbol{W}^t) - \eta K^t
\end{aligned}$$

where

$$\begin{aligned}
K^t &= \nabla f(\boldsymbol{W}^t)^T\Delta\boldsymbol{W}^t - \frac{1}{2}\eta L\|\Delta\boldsymbol{W}^t\|^2 \\
&\geq \nabla f(\boldsymbol{W}^t)^T\Delta\boldsymbol{W}^t - \frac{1}{2}\|\Delta\boldsymbol{W}^t\|^2
\end{aligned}$$

because we assumed that the step size is $0 \leq \eta \leq \frac{1}{L}$. By applying equations (8) and (9), we obtain

$$\begin{aligned}
K^t &\geq \nabla f(\boldsymbol{W}^t)^T\Delta\boldsymbol{W}^t - \frac{1}{2}\|\Delta\boldsymbol{W}^t\|^2 \\
&= \nabla f(\boldsymbol{W}^t)^T\Delta\boldsymbol{W}^t - \frac{1}{2}(\Delta\boldsymbol{W}^t)^T\Delta\boldsymbol{W}^t \\
&= \frac{1}{2}[2f(\boldsymbol{W}^t) - (\Delta\boldsymbol{W}^t)]^T\Delta\boldsymbol{W}^t \\
&= \frac{1}{2}\mathrm{Vec}(2\boldsymbol{\delta}_1\boldsymbol{h}_0^T - \boldsymbol{\delta}_1\boldsymbol{r}_0^T, \ldots, 2\boldsymbol{\delta}_L\boldsymbol{h}_{L-1}^T - \boldsymbol{\delta}_L\boldsymbol{r}_{L-1}^T)^T\mathrm{Vec}(\boldsymbol{\delta}_1\boldsymbol{r}_0^T, \ldots, \boldsymbol{\delta}_L\boldsymbol{r}_{L-1}^T) \\
&= \frac{1}{2}\sum_{l=0}^{L-1}\mathrm{Vec}(\boldsymbol{\delta}_{l+1}(2\boldsymbol{h}_l - \boldsymbol{r}_l)^T)^T\mathrm{Vec}(\boldsymbol{\delta}_{l+1}\boldsymbol{r}_l^T)
\end{aligned}$$

We can show that $\mathrm{Vec}(AB^T)^T\mathrm{Vec}(AC^T) = \|A\|^2C^TB$ when A, B, and C are column vectors, and B and C have the same dimension as below.

$$\begin{aligned}
\mathrm{Vec}(AB^T)^T\mathrm{Vec}(AC^T) &= \mathrm{Vec}([a_ib_j]_{ij})^T\mathrm{Vec}([a_ic_j]_{ij}) \\
&= [a_1b_1, a_2b_1, \cdots, a_mb_n][a_1c_1, a_2c_1, \cdots, a_mc_n]^T \\
&= (a_1^2 + \cdots + a_m^2)b_1c_1 + (a_1^2 + \cdots + a_m^2)b_2c_2 + \cdots + (a_1^2 + \cdots + a_m^2)b_nc_n \\
&= (a_1^2 + \cdots + a_m^2)(b_1c_1 + \cdots + b_nc_n) \\
&= \|A\|^2C^TB
\end{aligned}$$

Consequently, by applying this equality to the inequality above,

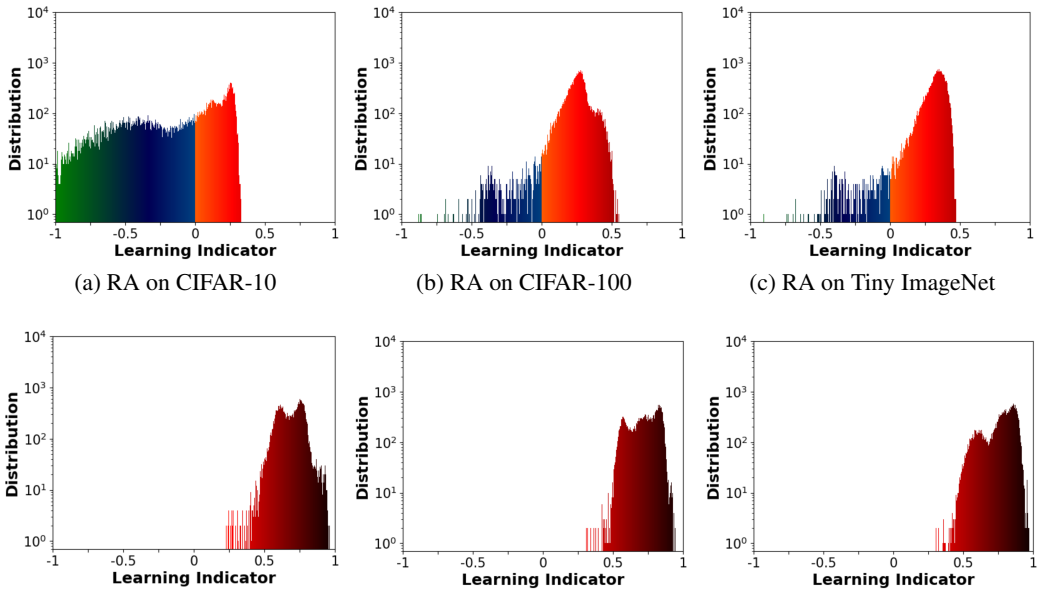
$$\begin{aligned}
K^t &= \frac{1}{2}\sum_{l=0}^{L-1}\mathrm{Vec}(\boldsymbol{\delta}_{l+1}(2\boldsymbol{h}_l - \boldsymbol{r}_l)^T)^T\mathrm{Vec}(\boldsymbol{\delta}_{l+1}\boldsymbol{r}_l^T) \\
&\geq \frac{1}{2}\sum_{l=0}^{L-1}\|\boldsymbol{\delta}_{l+1}\|^2\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l) \\
&\geq 0
\end{aligned}$$

because $\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l) > 0$ in all layers by the assumption above. Therefore, $0 \leq f(\boldsymbol{W}^{t+1}) \leq f(\boldsymbol{W}^t) - \eta K^t \leq f(\boldsymbol{W}^{t-1}) - \eta K^{t-1} - \eta K^t \leq f(\boldsymbol{W}^0) - \eta \sum_{n=0}^{t} K^n$, which suggests that the value of $f(\boldsymbol{W})$ becomes smaller as the number of epochs t increases because $K^n \geq 0$. Consequently, because $f(\boldsymbol{W})$ decreases and is bounded below by 0, $f(\boldsymbol{W})$ converges to zero during training even when using alternative activations for updating weights by the monotone convergence theorem. Furthermore, if the values of $\boldsymbol{r}_l^T(2\boldsymbol{h}_l - \boldsymbol{r}_l)$ increases, K^n, which is the amount of loss reduction in each epoch, also increases and therefore the network converges well. □

B DOES ADDING AUXILIARY ACTIVATION REALLY HELP?

Theorem 1 suggests that an alternative type of activation can replace actual input activation in backward propagation, as long as it satisfies the learning criterion. However, it is difficult to meet this condition using arbitrary values, so we proposed to add the alternate activation to real activation in forward propagation. Then, the input activation $\boldsymbol{h}_l = \boldsymbol{o}_l + \boldsymbol{a}_l$ in forward propagation and the alternate activation $\boldsymbol{a}_l$ in backward propagation share a common term, which is expected to help satisfy the learning criterion. In this section, we experimentally verify this claim by comparing ARA and ASA to baselines where the alternate activations are not added to real activation in forward propagation.

In baselines, we replace real activation with the previous layer's activation and 1-bit sign of original activation in backward propagation, while they are not added to real activation in forward propagation. They are denoted by Residual Activation (RA) and Sign Activation (SA), respectively. SA is identical to simply quantizing activation into 1 bit in conventional backpropagation.

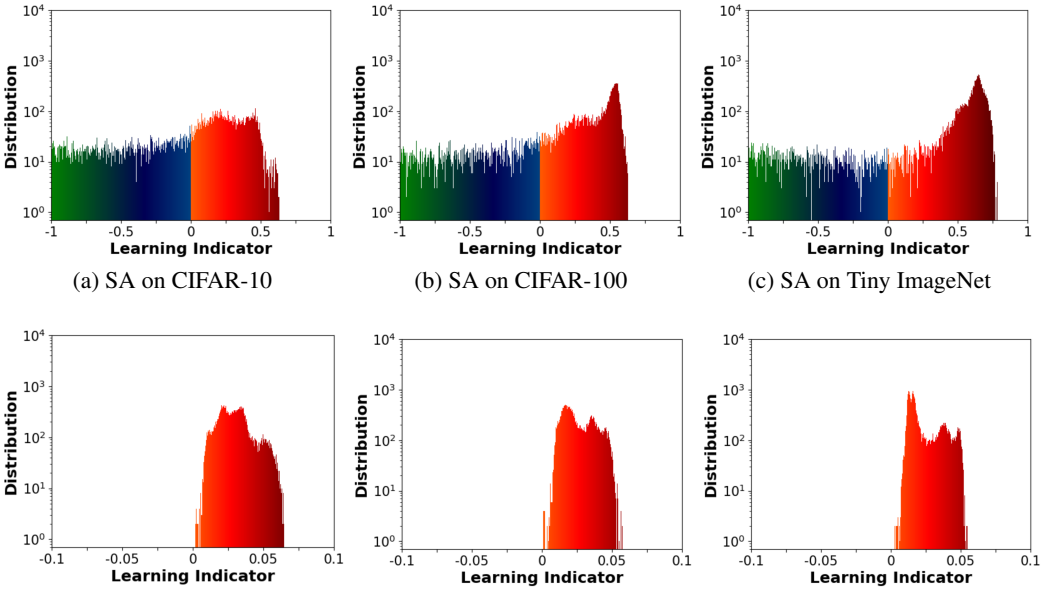


(a) RA on CIFAR-10 (b) RA on CIFAR-100 (c) RA on Tiny ImageNet

(d) ARA on CIFAR-10 (e) ARA on CIFAR-100 (f) ARA on Tiny ImageNet

Figure 6: Distributions of learning indicator values of RA and ARA in ResNet-18

Figs. 6 and 7 display learning indicator values of ResNet-18 trained on CIFAR-10, CIFAR-100, and Tiny ImageNet. These results show that adding auxiliary activation to real activation in forward propagation significantly improves the distributions of learning indicator values by positively skewing them toward 1. This is consistent with training performance comparisons provided in Table 6. ARA and ASA noticeably outperform RA and SA in all experiments, and the performance improvement is more significant for larger datasets. Therefore, we can conclude that adding auxiliary activation to real activation in forward propagation does help neural network training.



(a) SA on CIFAR-10 (b) SA on CIFAR-100 (c) SA on Tiny ImageNet

(d) ASA on CIFAR-10 (e) ASA on CIFAR-100 (f) ASA on Tiny ImageNet

Figure 7: Distributions of learning indicator values of SA and ASA in ResNet-18

Table 6: Test accuracy of ResNet-18 trained on CIFAR-10, CIFAR-100, and Tiny ImageNet.

	Learning Rule	CIFAR-10	CIFAR-100	Tiny ImageNet
Accuracy (%)	BP	94.77	75.81	58.43
	RA (2, 2, 2, 2)	94.1	72.23	53.91
	ARA (2, 2, 2, 2)	94.81	75.49	58.46
	SA (2, 2, 2, 2)	94.36	74.58	55.76
	ASA (2, 2, 2, 2)	94.76	75.31	57.01

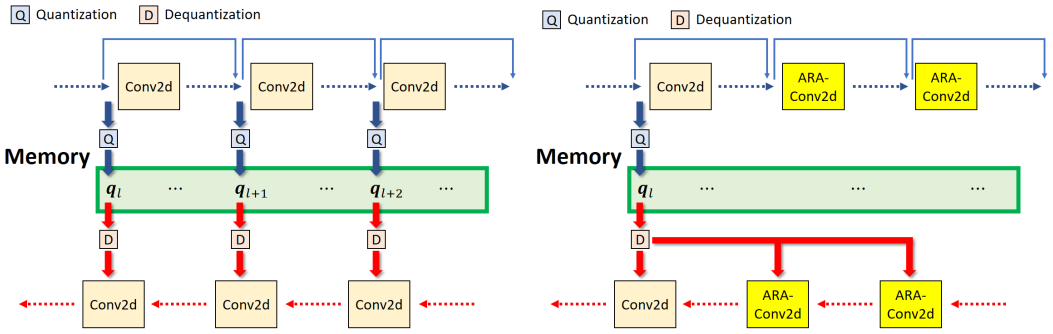
C Experimental Details

C.1 How we apply gradient checkpointing in each experiment

Chen et al.(16) suggested that when there is a BN-ReLU-Conv layer in the network, applying gradient checkpointing to those layers can reduce memory requirements without incurring significant computational overhead. More specifically, only the input activation of the BN layer is stored during forward propagation, and the rest is recalculated in backward propagation. Therefore, while the BN and ReLU layers have to be recomputed during backward propagation to generate the input activation of the Conv layer, there is no need to recompute the Conv layer itself which may lead to large computational overhead. We applied this method to BN-ReLU-Conv3x3 layers in Fig. 3a to reduce memory overhead without a significant amount of recomputation. In addition, we apply gradient checkpointing to BN-ReLU-Conv1x1-BN layers in Fig. 3a. Although we have to recompute Conv1x1, its computational overhead is relatively small compared to Conv3x3 layers. We also applied gradient checkpointing to the first Conv-BN-ReLU-MaxPool layers in ResNet. Here only the input of the first Conv layer is stored during forward propagation, and the input activation of the other layers is recomputed in backward propagation. Although the first Conv layer has to be recomputed, a large amount of training memory can be saved because the amount of the input activation in the first Conv, BN, and MaxPool layers is large.

C.2 How we apply activation compression training in each experiment

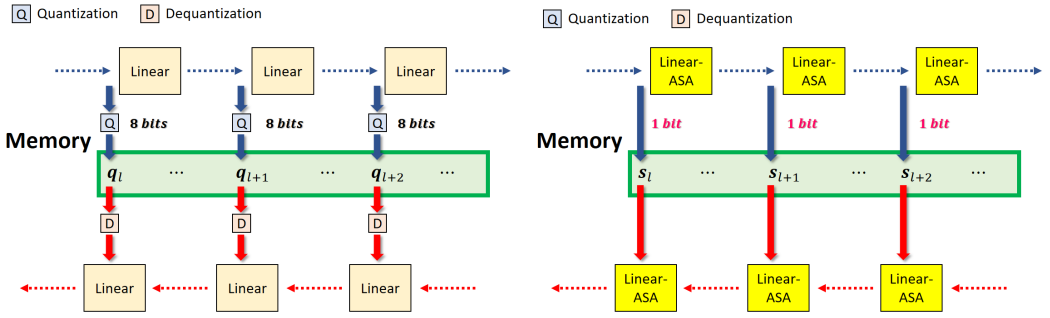
ActNN (25) stores 2-bit compressed activations instead of actual activations for all layers except for ReLU and pooling which only needs 1-bit sign activations. We applied ActNN (L3) to all layers,



(a) ActNN (b) ActNN with ARA

Figure 8: Training process of ActNN with ARA

which performs per-group quantization with fine-grained mixed precision. The per-group quantization partitions the activation into several groups, and then quantization is performed by groups. The fine-grained mixed precision includes two processes: per-sample allocation and per-layer allocation. In per-sample allocation, the sensitivity, which is related to gradient variance, is calculated for each group and then an optimal number of bits for each group is determined using this sensitivity. Then, per-layer allocation calculates the sensitivity of each layer and then an optimal number of bits for each layer is obtained. When ARA and ActNN are applied simultaneously, we perform 2-bit group-wise quantization and per-sample allocation on auxiliary residual activation. However, per-layer allocation is not applied to auxiliary residual activation because it is not straightforward to calculate the sensitivity of ARA layers. While ActNN needs to quantize and store activations of all layers, we can skip this process for ARA_Conv layers when using ActNN with ARA (Fig. 8).



(a) Mesa (b) Mesa with ASA

Figure 9: Training process of Mesa with ASA.

Pan et al. (27) pointed out that conventional activation compression algorithms do not consider heterogeneous activation distributions in a multi-head attention layer and therefore they are not applicable to Transformer-like models. Therefore, they proposed Mesa which performs head-wise quantization instead of per-tensor quantization. When we combine Mesa and ASA, we replace linear layers with ASA-Linear layers as shown in Fig. 9.

C.3 Datasets

CIFAR-10 and CIFAR-100 The CIFAR-10 dataset (36) consists of 50,000 training images and 10,000 test images which are 32x32 RGB images for 10-class image classification. Likewise, the CIFAR-100 dataset (36) includes 50,000 training images and 10,000 test images with the same resolution for 100-class image classification.

Tiny ImageNet The Tiny-ImageNet dataset (37) consists of images of 200 classes and each class has 500 images for training. It also contains 10,000 test images. All images included in Tiny ImageNet are selected from ImageNet and downsized to 64x64.

ImageNet The ImageNet dataset (31) is a representative large-scale dataset for image classification. It consists of 1,281,167 training images, 50,000 validation images, and 100,000 test images which are all RGB images. The average image size on ImageNet is 469x387, but they are randomly cropped to 224x224 for image classification.

IWSLT 2016 The IWSLT 2016 dataset (39) is constructed for spoken language translation task. We choose IWSLT2016 De-En dataset which is used for German-to-English translation. It has 1,611 talks, 197K sentences, and 3.96M tokens as a training set while the validation set consists of 13 talks, 1.1K sentences, and 21K tokens.

MNLI The Multi-Genre Natural Language Inference (MultiNLI) dataset (40) consists of 433k sentence pairs for ten genres (Face-to-Face, Telephone, etc.). The dataset is divided into 392702 train sets, 9815 validation_matched sets which are subsets of train sets, and 9832 validation_mismatched sets which are included in train sets.

C.4 HYPERPARAMETES

ResNet For training ResNet-18 from scratch on CIFAR-10, CIFAR-100, and Tiny ImageNet, we set the batch size and the total number of epochs to 512 and 200, respectively. We applied stochastic gradient descent with momentum (42) along with weight decaying (36) to our experiments. The momentum and weight decay rate was set to 0.9 and 1e-4, respectively. The learning rate of the layers except for ASA layers was scheduled by cosine annealing (43) with a 0.1 initial learning rate during 200 epochs. In comparison, we used a 100x higher learning rate for ASA layers to make the magnitude of weight updates comparable to those of other layers. We also set ϵ in equations (15) and (16) to 0.01. We used a nearly identical set of hyperparameters for ImageNet experiments. However, we set the batch size and the number of epochs to 512 and 90, respectively, and the learning rate was decreased by 10x every 30 epochs instead of cosine annealing.

Transformer In Transformer, the batch size and the number of epochs are set to 4096 and 20, respectively. We trained Transformer from scratch on IWSLT and used the Adam optimizer (44) by setting β_1 to 0.9, β_2 to 0.98, and ϵ to 1e-9. In addition, the learning rate was updated by the warm-up learning rate scheduler (43), and the number of warm-up steps was set to 4,000. We applied a 100x higher learning rate to ASA layers as in ResNet experiments. ϵ in equations (15) and (16) for ASA was set to 0.01. The model dimension was 512, the hidden layer dimension was 2048, the number of heads was 8, and the dropout rate (45) was 0.2. Furthermore, we applied label smoothing with a 0.1 label smoothing rate (46) to all experiments in Transformer.

BERT-Large We used Bert-Large (uncased) pretrained on BookCorpus (47) and English Wikipedia for fine-tuning on MNLI. The batch size was set to 32, and we trained BERT-Large for three epochs. The Adam optimizer (44) was used with 0.9 β_1, 0.999 β_2, 1e-6 ϵ, and 0.01 weight decay rate. We also employed warm-up in the learning rate up to 10% of the total step, and cosine annealing is used for the remaining epochs (43). The learning rate of cosine annealing was set to 1.5e-5. These learning rate was increased by 100x for ASA layers in ViT experiments. ϵ in equations (15) and (16) was set to 0.01. The other hyperparameters were selected according to the original BERT paper (33).

ViT-Large We fine-tuned a ViT-Large model pretrained on ImageNet-21k. The model was fine-tuned using 32 batch size on CIFAR-100. We set the training batch size to 512 and experimented with different learning rates of 3e-2, 1e-1, and 3e-1 to achieve the highest performance. For training, learning rate was adjusted by stochastic gradient descent with momentum (42) and a warm-up cosine scheduler (43) was applied. The model was trained for 1000 steps with 100 warm-up steps. For Linear-ASA layers, 100x larger learning rate was used. We also set ϵ in equations (15) and (16) to 0.01. The remaining hyperparameters were set according to the original ViT paper (34).

MLP-Mixer-Large The MLP-Mixer-Large model was fine-tuned from a pretrained model on ImageNet-21k with 16 patch size. We trained the model on CIFAR-10 for 2000 steps with 56 batch size. Like ViT, the SGD optimizer (42) was applied to these experiments. We set the weight decay rate and momentum to 0 and 0.9, respectively. During the first 200 steps, the learning rate warmed up and cosine annealing (43) was employed in the remaining epochs. We set the learning rate to 3e-2, 1e-1, and 3e-1. In the ASA layers, however, the learning rate was increased by 100x like Transformer and ViT-Large. ϵ of ASA layers in equations (15) and (16) was set to 0.01.The other hyperparameters followed the original MLP-Mixer paper (35).

C.5 Devices and memory measurements

In all experiments, we used Nvidia GeForce RTX 3090 GPUs. While one GPU was utilized to perform most of the experiments, we used six GPUs to train ResNet on ImageNet to reduce training time. We calculated the amount of training memory using *torch.cuda.memory_allocated* function in PyTorch instead of *nvidia-smi* because *nvidia-smi* includes unused memory space handled by the memory allocator in the report[1].

D Memory and compute complexity of Auxiliary Activation Learning

Table 7: The compute and memory complexity analysis.

	BP	GCP	AAL only	AAL with GCP
Compute complexity	$3N$	$4N$	$3N$	$4N$
Memory complexity	N	$2\sqrt{N}$	$N-A$	$2\sqrt{N-A}$

In this section, we compare the compute and memory complexity of different learning algorithms. For the sake of simplicity, we assume that the computational cost of forward propagation and backward propagation of a single layer is 1. The amount of activations of a single layer is also assumed to be 1. Then we calculate the computational cost and memory space required for training a neural network with N identical layers. BP would need N computations for forward propagation, N computations for error propagation, and N computations for calculating weight updates, which results in $3N$ computations in total. For memory space, BP has to store N activations generated during forward propagation. On the other hand, GCP requires N additional computations for the recomputation of activations. However, it can reduce the memory complexity to $2\sqrt{N}$ when the network is divided into $\sqrt{N}$ blocks as suggested in (16). In our Auxiliary Activation Learning (AAL), we have to store auxiliary activations instead of actual activations. We assume that our algorithm is applied to A layers in the network. Then, we have to store actual activation for the rest of the layers, which translates to a memory complexity of $N-A$. For the layers employing our algorithm, we only have to store auxiliary activations and its amount is negligible compared to real activations. For example, Auxiliary Residual Activation (ARA) uses the previous layer's activation as auxiliary activation, and hence there is no need to store this auxiliary activation since it is already stored in memory during forward propagation of the previous layer. Auxiliary Sign Activation (ASA) has to store a 1-bit sign of actual activations, respectively, but they are also significantly smaller than real activations. Therefore, our algorithm has $N-A$ memory complexity and $3N$ compute complexity. To further reduce memory complexity, we can apply gradient checkpointing to the layers where we do not apply our AAL algorithm. Thus, the memory complexity of those layers would decrease to $2\sqrt{N-A}$ and the compute complexity becomes $4N$ due to recomputation.

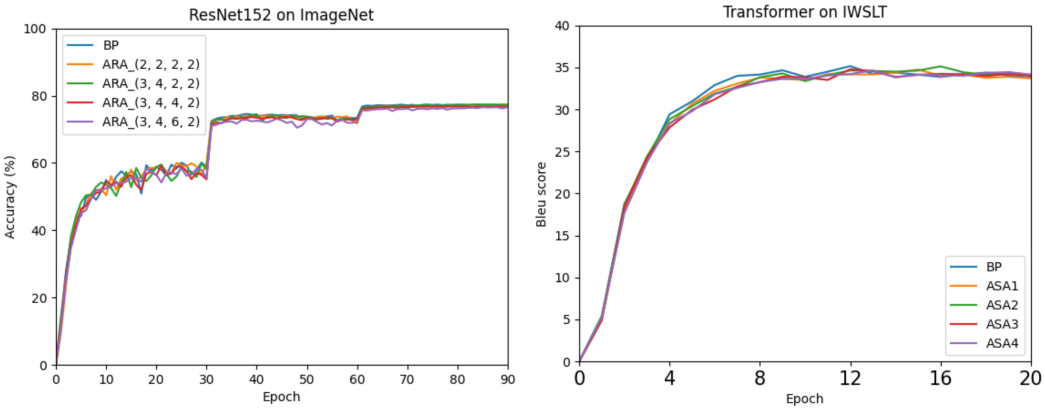
E Comparisons to algorithms to approximate backpropagation

Previous studies have suggested various algorithms to approximate backpropagation (48; 49; 50). Our algorithm may be considered in line with such approaches as it replaces actual input activation with auxiliary activation. Furthermore, Uniform Sign-Concordant Feedback (50), which uses the

[1] https://pytorch.org/docs/stable/notes/cuda.html?highlight=buffer#memory-management

sign of forward weights as feedback weights, looks similar to the concept of ASA which uses the sign of activation as auxiliary activation. However, the motivation is quite different. In prior works on approximate backpropagation, the authors aim to remove the need for symmetric feedforward and feedback weights ("weight transport problem") by approximating the feedback path for better bio-plausibility. Contrarily, our algorithm reduces memory requirements by using an alternate form of activations, but it does not necessarily improve bio-plausibility as we still need symmetric forward and backward paths.

F CONVERGENCE SPEED OF AUXILIARY ACTIVATION LEARNING



(a) ARA on ResNet-152

(b) ASA on Transformer

Figure 10: Learning curves of BP and Auxiliary Activation Learning.