Accumulated Decoupled Learning with Gradient Staleness Mitigation for Convolutional Neural Networks

Huiping Zhuang [1] **Zhenyu Weng** [1] **Fulin Luo** [1] **Kar-Ann Toh** [2] **Haizhou Li** [3] **Zhiping Lin** [1]

Abstract

Gradient staleness is a major side effect in decoupled learning when training convolutional neural networks asynchronously. Existing methods that ignore this effect might result in reduced generalization and even divergence. In this paper, we propose an accumulated decoupled learning (ADL), which includes a module-wise gradient accumulation in order to mitigate the gradient staleness. Unlike prior arts ignoring the gradient staleness, we quantify the staleness in such a way that its mitigation can be quantitatively visualized. As a new learning scheme, the proposed ADL is theoretically shown to converge to critical points in spite of its asynchronism. Extensive experiments on CIFAR-10 and ImageNet datasets are conducted, demonstrating that ADL gives promising generalization results while the state-of-the-art methods experience reduced generalization and divergence. In addition, our ADL is shown to have the fastest training speed among the compared methods. The code will be ready soon in https://github.com/ZHUANGHP/Accumulated-Decoupled-Learning.git.

1. Introduction

Convolutional neural networks (CNNs) (LeCun et al., 1998) are normally trained by backpropagation (BP) which runs a forward pass followed by a backward one synchronously through a network before parameter update. The synchronism of BP is mainly characterized by three lockings (Jaderberg et al., 2016), i.e., the forward, the backward and the update lockings (see details in Section 3.1), which render the majority of a network idle during training. To improve the efficiency, the *decoupled learning* (Jaderberg et al., 2016) emerges by addressing one or more of these lockings.

The decoupled learning partitions a network depth-wise into several modules with each module containing a stack of layers, and facilitates a parallel training among the partitioned modules. A large number of partitioned modules would encourage a high degree of efficiency being improved. For convenience, the partition size (PS) is taken as the number of modules being partitioned from a network. Methods of decoupled learning seek to deliver comparable generalization performance to its BP counterpart under various PS. We divide these prior arts into two groups: the delayed gradient (DG) based methods and the local error learning (LEL) based methods.

The DG-based methods (Huo et al., 2018a;b; Zhuang et al., 2021) adopt DGs to avoid the synchronism of BP. They deliver comparable generalization performance relative to the BP baseline, but introduce the gradient staleness (or the stale gradient effect). Such a staleness has limited the DG-based methods towards a small PS. That is, most DG-based methods only perform well for PS ≤ 4 (Huo et al., 2018a;b). This is because the gradient staleness grows as PS increases, causing a reduced generalization or even divergence.

The staleness issue does not appear in the LEL-based methods as they sever the gradient flow between adjacent modules by building auxiliary networks to generate local error gradients. This allows the modules to avoid the global BP thereby addressing all the three lockings. Methods in this group is distinguished by the design of auxiliary networks. The LEL-based methods do not experience divergence as PS increases, but usually encounter generalization loss instead.

In summary, the DG-based methods can give better generalization results than LEL-methods but they are constrained by a limited PS due to the staleness issue. Dwelling on the DG-based methods, we aim to scale the learning to a large PS with comparable generalization performance (see Table 1). To this end, the key is to reduce the gradient staleness. Here, we propose an accumulated decoupled learning (ADL), which effectively incorporates a module-wise gradient accumulation (mGA) technique in the partitioned modules to mitigate the gradient staleness. This work include the following contributions:

[1]School of Electrical and Electronic Engineering, Nanyang Technological University. [2]Department of Electrical and Electronic Engineering, Yonsei University. [3]School of Electrical and Computer Engineering, National Univeristy of Singapore. Correspondence to: Zhiping Lin <ezplin@ntu.edu.sg>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

• Proposal of a new decoupled learning technique by addressing all the three locking problems in BP.

• Incorporation of an mGA technique into the decoupled learning, which has been shown to reduce the gradient staleness, theoretically and empirically.

• Convergence analysis which shows that our method can converge to critical points.

• Experimental validation on CIFAR-10 and ImageNet datasets. Particularly, we show that the ADL in general outperforms the state-of-the-arts especially under the scenario of a large PS, and has the fastest training speed among the compared methods.

2. Related Works

2.1. Local Error Learning Based Methods

The key feature of LEL-based methods is the design of auxiliary networks. They originate from the decoupled neural interface (DNI) (Jaderberg et al., 2016) by generating local synthetic gradients. This approach is followed up by (Mostafa et al., 2018) using a local classifier. The decoupled greedy learning (DGL) (Belilovsky et al., 2020) designs a light-weight auxiliary network for the purpose of making a trade-off between the generalization performance and the computation workload. Although the pred-sim method in (Nøkland & Eidnes, 2019) with combined losses manages to obtain a comparable performance with the BP baseline, it has only been verified in relatively shallow networks ($\leq$13 layers).

2.2. Delayed Gradient Based Methods

The DG-based methods attain decoupled learning by updating the network modules with DGs. The decoupled parallel BP with DGs (DDG) (Huo et al., 2018b) breaks the backward locking while having a comparable performance to BP on the ResNet (RN) (He et al., 2016) under a small PS (≤ 4). The future replay (FR) (Huo et al., 2018a) that follows up employs a recomputation unit with inconsistent weights and gradients during the forward and the backward passes. The FR also unlocks the backward pass and this gives as equally good performance as the BP baseline for small PS values. The fully decoupled method with DGs (Zhuang et al., 2021) further addresses the forward and the update lockings, and this leads to a lock-free decoupled learning. Apart from these, a technique called DSP (Xu et al., 2020) has also attained a lock-free decoupled learning. However, these prior arts using DGs inevitably suffer from the gradient staleness, which becomes apparent as the PS grows.

Table 1. Characteristics of decoupled learning methods and the proposed ADL.

Methods	DG-based methods	LEL-based methods	ADL (ours)
Generalization v.s BP	comparable	worse	comparable
Partition size	small	large	large

2.3. Other Works Related to Decoupled Learning

Another set of methods, named asynchronous stochastic gradient descent (ASGD) (Dean et al., 2012; Lian et al., 2015; Zheng et al., 2017), involves utilization of DGs to perform asynchronous parallelization. Essentially, the ASGD methods differ from the decoupled learning in terms of parallelization paradigm. They belong to the data parallelism category as each work handles a complete network replica. In addition, the pipeline BP shares certain similarity with the decoupled learning. For instance, the GPipe proposed in (Huang et al., 2019) splits a network into several modules and breaks each mini-batch into several micro-batches to conduct pipeline model parallelization. The micro-batching shares some similarities with the mGA technique in our proposed ADL. However, unlike decoupled learning, the GPipe is a form of synchronous model parallelism, which does not involve DGs.

3. Preliminaries

Here, we revisit the necessary knowledge for training a feedforward neural network, including the generic gradient accumulation (gGA) used in BP learning. Along with this revisit, the BP lockings (Jaderberg et al., 2016) as well as the gradient staleness are also explained.

3.1. Backpropagation and Lockings

Assume that we need to train an $\mathcal{L}$-layer network. The l^{th} ($1 \leq l \leq \mathcal{L}$) layer produces an activation $\boldsymbol{z}_l = F_l(\boldsymbol{z}_{l-1}; \boldsymbol{\theta}_l)$ by taking $\boldsymbol{z}_{l-1}$ as its input, where F_l is an activation function and $\boldsymbol{\theta}_l \in \mathbb{R}^{n_l}$ is weight vector at layer l. The sequential generation of the activations results in a *forward locking* since $\boldsymbol{z}_l$ depends on the activations from its previous layers. Let $\boldsymbol{\theta} = [\boldsymbol{\theta}_1^T, \boldsymbol{\theta}_2^T, ..., \boldsymbol{\theta}_{\mathcal{L}}^T]^T \in \mathbb{R}^{\Sigma_{i=1}^{\mathcal{L}} n_i}$ denote the parameter vector of the entire network. Assume that f is a loss function. Training the feedforward network can then be formulated as

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad f_{\boldsymbol{x}}(\boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{x}$ represents the entire input-label information (or the entire dataset). In the rest of this paper, we shall use $f(\boldsymbol{\theta})$ to represent $f_{\boldsymbol{x}}(\boldsymbol{\theta})$ for convenience.

The gradient descent algorithm is often used to solve Eq. (1) by updating the parameter $\boldsymbol{\theta}$ iteratively as follows:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \gamma_t \bar{\boldsymbol{g}}_\theta^t \tag{2}$$

or equivalently,

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \bar{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t,\ l = 1, ..., \mathcal{L} \tag{3}$$

where γ_t is the learning rate. Index t here usually implies the *batch index*, with $\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t$ indicating the gradient obtained with respect to (w.r.t.) data batch t. Let $\bar{\boldsymbol{g}}_{\theta}^t = [(\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_1}^t)^T, (\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_2}^t)^T, ..., (\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_\mathcal{L}}^t)^T]^T \in \mathbb{R}^{\Sigma_{i=1}^{\mathcal{L}} n_i}$, which is obtained by

$$\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t = \frac{\partial f(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^t}. \tag{4}$$

If the dataset is large, the stochastic gradient descent (SGD) is often used as an alternative:

$$\boldsymbol{g}_{\boldsymbol{\theta}_l}^t = \frac{\partial f_{\boldsymbol{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^t} \tag{5}$$

where $\boldsymbol{x}_t$ is the t^{th} mini-batch drawn from the dataset $\boldsymbol{x}$. We remove the bar "$^-$" on $\boldsymbol{g}$ to tell its difference from that in Eq. (4). Accordingly, the network weights can be updated through

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \boldsymbol{g}_{\boldsymbol{\theta}_l}^t,\ l = 1, ..., \mathcal{L}. \tag{6}$$

Assume that each sample is randomly drawn from a uniform distribution. Then the gradient is unbiased:

$$\mathbb{E}_{\boldsymbol{x}}\{\boldsymbol{g}_{\boldsymbol{\theta}_l}^t\} = \bar{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t \tag{7}$$

where the expectation $\mathbb{E}_{\boldsymbol{x}}$ is taken w.r.t. the random variable that draws $\boldsymbol{x}_t$ from the dataset.

To obtain the gradient vectors, the BP is used. We can calculate the gradient at layer l using the gradient back-propagated from layers j and i ($l < j < i$) as follows:

$$\boldsymbol{g}_{\boldsymbol{\theta}_l}^t = \frac{\partial f_{\boldsymbol{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^t} = \frac{\partial \boldsymbol{z}_j^t}{\partial \boldsymbol{\theta}_l^t} \frac{\partial f_{\boldsymbol{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{z}_j^t} = \frac{\partial \boldsymbol{z}_j^t}{\partial \boldsymbol{\theta}_l^t} \boldsymbol{g}_{\boldsymbol{z}_j}^t \tag{8}$$

where

$$\boldsymbol{g}_{\boldsymbol{z}_j}^t = \frac{\partial f_{\boldsymbol{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{z}_j^t} = \frac{\partial \boldsymbol{z}_i^t}{\partial \boldsymbol{z}_j^t} \frac{\partial f_{\boldsymbol{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{z}_i^t} = \frac{\partial \boldsymbol{z}_i^t}{\partial \boldsymbol{z}_j^t} \boldsymbol{g}_{\boldsymbol{z}_i}^t. \tag{9}$$

Here we introduce $\boldsymbol{g}_{\boldsymbol{z}_j}^t$—the gradient vector w.r.t. activation $\boldsymbol{z}_j$—because it travels through modules for communication in our ADL. Eq. (8) and Eq. (9) indicate that $\boldsymbol{g}_{\boldsymbol{\theta}_l}^t$ is obtained based on $\boldsymbol{g}_{\boldsymbol{z}_j}^t$ and $\boldsymbol{g}_{\boldsymbol{z}_i}^t$. That is, the gradient is not accessible before the forward pass has been completed and all the dependent gradients have been obtained, which is known as the *backward locking*. On the other hand, we cannot update the weights before every layer finishes its forward pass, which is recognized as the *update locking*.

3.2. Learning with Generic Gradient Accumulation

The gGA has frequently been adopted to increase the mini-batch size for training networks on devices with a relatively limited memory setting. The gradients obtained based on several mini-batches are accumulated before they are finally applied to update the network.

To describe the training development involving the gGA technique, we introduce an *update index* s, and a *wrapped batch index* U_s w.r.t. the original batch index t. We use the update index s to indicate the s^{th} parameter update of the network. It is connected to U_s in the way of $U_s = Ms$ given M gGA steps. The gGA renders the network parameters unchanged for M steps, i.e., $\boldsymbol{\theta}_l^{U_s} = \boldsymbol{\theta}_l^{U_s+1} = \cdots = \boldsymbol{\theta}_l^{U_s+M-1}$. Conversely, we can tell the update index from a batch index t by

$$s = \lfloor t/M \rfloor \tag{10}$$

where $\lfloor x \rfloor = \max\{n \in \mathbb{Z} | n \leq x\}$ is the *floor* operator. That is, when the network is processing the t^{th} mini-batch of data, the network has been updated for s times based on Eq. (10).

Assume that the gradients w.r.t. batch indexes $t = U_s, U_s + 1, \ldots, U_s + M - 1$ are accumulated. These gradients are obtained through

$$\boldsymbol{g}_{\boldsymbol{\theta}_l}^t = \frac{\partial f_{\boldsymbol{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^{U_s}} \tag{11}$$

where parameter $\boldsymbol{\theta}_l^{U_s} = \boldsymbol{\theta}_l^{U_{\lfloor t/M \rfloor}}$ is adopted instead of Eq. (5) in order to emphasize that the gradients w.r.t. to these data batches are being obtained based on the same parameter. Using the gGA technique, the weights are updated as follows:

$$\boldsymbol{\theta}_l^{U_{s+1}} = \boldsymbol{\theta}_l^{U_s} - \gamma_s (1/M) \textstyle\sum_{j=0}^{M-1} \boldsymbol{g}_{\boldsymbol{\theta}_l}^{U_s+j}. \tag{12}$$

3.3. Gradient Staleness

The network is commonly updated with gradients obtained w.r.t. the current parameters. However, there are certain scenarios where the network has to update its parameters with gradients calculated based on "older" parameters. This introduces gradient staleness or stale gradient effect, as the gradients are not up-to-date, and are therefore less accurate.

We define the *level of staleness* (LoS) as the update index difference between the current parameter and the parameter used to calculate the stale gradient. Suppose a network is updated through

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \boldsymbol{g}_{\boldsymbol{\theta}_l}^{t-d},\ l = 1, ..., \mathcal{L} \tag{13}$$

where $\boldsymbol{g}_{\boldsymbol{\theta}_l}^{t-d} = \partial f_{\boldsymbol{x}_{t-d}}(\boldsymbol{\theta}^{t-d}) / \partial \boldsymbol{\theta}_l^{t-d}$. For a gGA step of M, we could calculate the LoS through

$$\text{LoS} = \lfloor t/M \rfloor - \lfloor (t-d)/M \rfloor \tag{14}$$

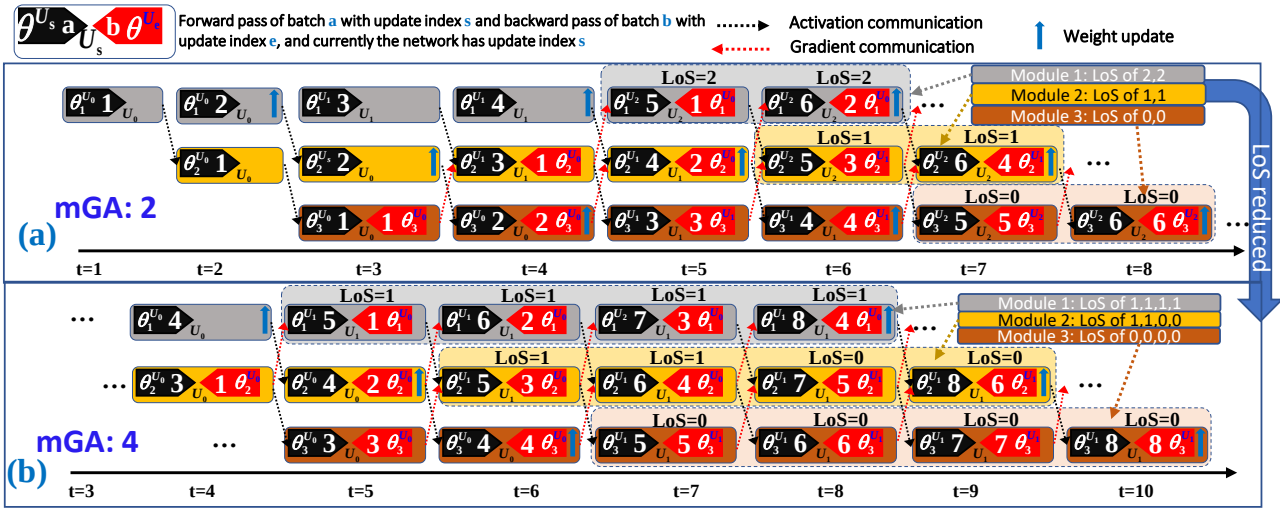

Figure 1. Training a 3-layer network by ADL with $K = 3$ and mGA steps of (a) $M = 2$ and (b) $M = 4$ suggesting that each module is updated every 2 and 4 iterations respectively. Note that there is a batch index difference of $2(K - k)$ between the forward and backward pass. For instance, for $K = 3$, module $k = 1$ (gray) generates an activation at iteration 1 but the gradient of this batch arrives at iteration 5. In addition, a larger M reduces the LoS according to Eq. (14). For instance , at $t = 5$ module 1 (gray) has LoS of 2 with $M = 2$, which is reduced to 1 with $M = 4$.

indicating that the current parameter is $\boldsymbol{\theta}_l^{U_{\lfloor t/M \rfloor}}$ while the parameter used to calculate gradient $\boldsymbol{g}_{\boldsymbol{\theta}_l}^{t-d}$ is $\boldsymbol{\theta}_l^{U_{\lfloor (t-d)/M \rfloor}}$.

4. An Accumulated Decoupled Learning

In this section, we show the algorithmic details of the proposed ADL, which include an asynchronous pipelining procedure to achieve model parallelism, and an mGA technique to mitigate the gradient staleness. In particular, we explicitly show how the staleness in each module is reduced.

Prior to our development, the network is first partitioned depth-wise into K modules with a stack of layers in each module. That is, we partition the set of layer indices $\{1, \ldots, \mathcal{L}\}$ into $\{q(1), q(2), \ldots, q(K)\}$ where $q(k) = \{m_k, m_k + 1, ..., m_{k+1} - 1\}$ denotes the layer indices in module k. This leads to certain notation changes as follows:

$$\boldsymbol{\theta}^t = [(\boldsymbol{\theta}^t_{q(1)})^T, ..., (\boldsymbol{\theta}^t_{q(K)})^T]^T, \boldsymbol{\theta}^t_{q(k)} = [(\boldsymbol{\theta}^t_{m_k})^T, ..., (\boldsymbol{\theta}^t_{m_{k+1}-1})^T]^T$$
$$\boldsymbol{g}^t_{\boldsymbol{\theta}} = [(\boldsymbol{g}^t_{\boldsymbol{\theta}_{q(1)}})^T, ..., (\boldsymbol{g}^t_{\boldsymbol{\theta}_{q(K)}})^T]^T, \boldsymbol{g}^t_{\boldsymbol{\theta}_{q(k)}} = [(\boldsymbol{g}^t_{\boldsymbol{\theta}_{m_k}})^T, ..., (\boldsymbol{g}^t_{\boldsymbol{\theta}_{m_{k+1}-1}})^T]^T$$
$$\bar{\boldsymbol{g}}^t_{\theta} = [(\bar{\boldsymbol{g}}^t_{\boldsymbol{\theta}_{q(1)}})^T, ..., (\bar{\boldsymbol{g}}^t_{\boldsymbol{\theta}_{q(K)}})^T]^T, \bar{\boldsymbol{g}}^t_{\boldsymbol{\theta}_{q(k)}} = [(\bar{\boldsymbol{g}}^t_{\boldsymbol{\theta}_{m_k}})^T, ..., (\bar{\boldsymbol{g}}^t_{\boldsymbol{\theta}_{m_{k+1}-1}})^T]^T.$$

We depict the proposed ADL with an example of training a 3-layer network with a PS of $K = 3$ in Fig. 1(a) ($M = 2$) and 1(b) ($M = 4$) respectively. As illustrated in the figures, at every iteration, each module runs a forward pass and a backward pass. The forward pass is executed with a module input that comes from the output of the lower module at the previous instance. The backward pass calculates the gradients by resuming a local BP using gradients inherited from the upper module based on the "older" data batches. Note that all the partitioned modules can run in parallel due to asynchronism as the modules process data from different batches. Next, each module accumulates gradients for M steps before the gradients are applied to update the module weights.

4.1. Asynchronous Learning without BP Lockings

Consider the weights of module k ($k = 1, \ldots, K$) at update index s with $\boldsymbol{\theta}^{U_s}_{q(k)}$. We detail the learning procedures in module k to conduct update $s + 1$ as follows.

4.1.1. Forward Pass

Module k conducts the forward passes using data batches with indexes $U_s, U_s + 1, \ldots, U_s + M - 1$. Let $j = 0, 1, \ldots, M - 1$. In detail, we feed the module input $\boldsymbol{z}^{U_s+j}_{m_k-1}$ received from module[1] $k - 1$ to generate activations in each layer, which are obtained w.r.t. the same parameter $\boldsymbol{\theta}^{U_s}_{q(k)}$. Next, we obtain the activation $\boldsymbol{z}^{U_s+j}_{m_{k+1}-1}$ at the end of this module, and send this activation to module $k + 1$ (if any).

4.1.2. Backward Pass

During the backward pass, module k resumes BP locally using the gradient $\boldsymbol{g}^{U_s+j-2(K-k)}_{z_{m_{k+1}-1}}$ received from module[2] $k+1$. Note that the superscript $U_s + j - 2(K - k)$ indicates

[1] For $k = 1$ the module input is the training data.
[2] For $k = K$ the gradient is generated by the loss function.

that there are $2(K-k)$ steps of batch index delay w.r.t. the forward pass (see Fig. 1 for illustration). Accordingly, we calculate the gradients at each layer ($m_k \leq l \leq m_{k+1}-1$) within this module as follows:

$$\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^{U_s+j} = \frac{\partial \boldsymbol{z}_{m_{k+1}-1}^{U_s+j-2(K-k)}}{\partial \boldsymbol{\theta}_l^{U_{\lfloor (U_s+j-2(K-k))/M \rfloor}}} \boldsymbol{g}_{\boldsymbol{z}_{m_{k+1}-1}}^{U_s+j-2(K-k)}. \quad (15)$$

Note that Eq. (15) is obtained w.r.t. $\boldsymbol{\theta}_l^{U_{\lfloor (U_s+j-2(K-k))/M \rfloor}}$ with update index $\lfloor (U_s+j-2(K-k))/M \rfloor$ instead of s. This is because the gradient is calculated based on the "older" data batches, which can tell their corresponding update indexes from Eq. (10). At the end of the local BP, gradient $\boldsymbol{g}_{z_{m_k-1}}^{U_s+j-2(K-k)}$ w.r.t. the module input $z_{m_k-1}^{U_s+j-2(K-k)}$ is generated, which is then sent to module $k-1$ (if any).

4.2. Update with Module-wise Gradient Accumulation

After obtaining the gradients using Eq. (15), the module is not updated immediately. Instead, we applied the mGA by accumulating these gradients for M steps before they are applied to update the corresponding module as follows:

$$\boldsymbol{\theta}_l^{U_{s+1}} = \boldsymbol{\theta}_l^{U_s} - \gamma_s (1/M) \textstyle\sum_{j=0}^{M-1} \hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^{U_s+j}. \quad (16)$$

Note that all the modules must consistently accumulate gradients calculated w.r.t. the same group of data batches, though the updates would happen asynchronously. For instance, with an mGA step of 2, if module 1 decides to accumulate batch 10 and 11, the other modules must accumulate these two batches, instead of other possible combinations such as 9 and 10, or 10 and 11. This module-wise nursing step differentiates the mGA from the gGA in the synchronized learning. We summarize the proposed ADL in Algorithm 1.

Note that the above ADL is a lock-free decoupled technique. Firstly, the global BP is cast into local BPs in each module running in parallel, which removes the backward locking. Secondly, the split modules adopt training data from different batches so that the forward passes can be executed without waiting for the data from the lower layers. This tackles the forward locking. Finally, each module is updated immediately without waiting for other modules to complete their forward passes, hence addressing the update locking.

4.3. Impact of Module-wise Gradient Accumulation

Indicated by Eq. (15), the gradients are obtained based on $\boldsymbol{\theta}^{U_{\lfloor (U_s+j-2(K-k))/M \rfloor}}$ while the parameter state is $\boldsymbol{\theta}^{U_s}$. Therefore, according to Eq. (14), the LoS for module k is shown as follows ($j = 0, 1, \ldots, M-1$):

$$d_{k,j} = s - \lfloor (U_s + j - 2(K-k))/M \rfloor. \quad (17)$$

Algorithm 1 The proposed ADL

Partition the network into K modules, and set mGA step of M;
for *each iteration* **do**
  for $k \leftarrow 1$ **to** K **(Parallel) do**
    Forward pass: generate the activations with module input (e.g., $\boldsymbol{z}_{m_k-1}^{U_s+j}$), and send the module output (e.g., $\boldsymbol{z}_{m_{k+1}-1}^{U_s+j}$) to module $k+1$ (if any);
    Backward pass: using gradient (e.g., $\boldsymbol{g}_{z_{m_{k+1}-1}}^{U_s+j-2(K-k)}$) received from module $k+1$ to calculate the gradients in each layer following Eq. (15), and send the gradient w.r.t. the module input (e.g., $\boldsymbol{g}_{z_{m_k-1}}^{U_s+j-2(K-k)}$) to module $k-1$ (if any);
    Update: **if** *module k accumulates gradients from batches $U_s, U_s+1, \ldots, U_s+M-1$* **then**
      Update the module using Eq. (16);
    end
  end
end

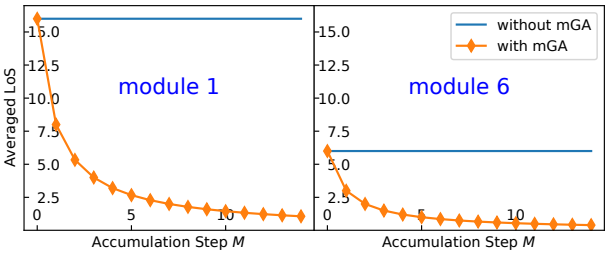

Figure 2. The averaged LoS w.r.t. mGA step of M.

For instance, as shown in Fig. 1(b), with $M=4$ module 2 updates its parameters using gradients with staleness $d_{k,j}$ ($1 \leq k \leq K, 0 \leq j < M$) of $d_{2,0}=1$, $d_{2,1}=1$, $d_{2,2}=0$, and $d_{2,3}=0$. Eq. (17) also indicates the staleness range:

$$0 \leq d_{k,j} \leq 2(K-k) \quad (18)$$

with the minimum $d_{k,j}$ reached for $j - 2(K-k) > 0$, and the maximum $d_{k,j} = 2(K-k)$ obtained at $M=1$ indicating no mGA involved. For convenience, we adopt the *averaged LoS*:

$$\bar{d}_k = (1/M) \textstyle\sum_{j=0}^{M-1} d_{k,j} \quad (19)$$

to evaluate the staleness in module k. As an example, Fig. 2 shows the averaged LoS in module 1 and 6 w.r.t. the accumulation step M with $K=8$, where the gradient staleness is shown to reduce with increasing M.

A large M leads to low gradient staleness, but it does not necessarily guarantee improvement in network generalization. This is because a larger M also indicates that the overall mini-batch size is large, which could weaken the network's ability to generalize (Keskar et al., 2016). As a result, M is an additional hyperparameter that handles the delicate balance between gradient staleness and generalization. In

fact, we do not need a large M to help the optimization because a small M has a rather significant impact on staleness reduction. As shown in Fig. 2, with $M = 4$ the averaged LoS has already been reduced by 75% (e.g., from 16 to 4).

In addition, using Eq. (17), we can unpack Eq. (15):

$$\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l^{U_s}}^{U_s+j} = \frac{\partial \boldsymbol{z}_{m_{k+1}-1}^{U_s+j-2(K-k)}}{\partial \boldsymbol{\theta}_l^{U_s-d_{k,j}}} \frac{\partial f_{\boldsymbol{x}_{U_s+j-2(K-k)}}(\boldsymbol{\theta}^{U_s-d_{k,j}})}{\partial \boldsymbol{z}_{m_{k+1}-1}^{U_s+j-2(K-k)}}$$
$$= \frac{\partial f_{\boldsymbol{x}_{U_s+j-2(K-k)}}(\boldsymbol{\theta}^{U_s-d_{k,j}})}{\partial \boldsymbol{\theta}_l^{U_s-d_{k,j}}} = \boldsymbol{g}_{\boldsymbol{\theta}_l}^{U_s+j-2(K-k)} \quad (20)$$

and then rewrites Eq. (16) as

$$\boldsymbol{\theta}_l^{U_{s+1}} = \boldsymbol{\theta}_l^{U_s} - \gamma_s (1/M) \textstyle\sum_{j=0}^{M-1} \boldsymbol{g}_{\boldsymbol{\theta}_l}^{U_s+j-2(K-k)}. \quad (21)$$

That is, the proposed ADL accumulates gradients that are $2(K-k)$ steps "older", while each of these accumulated gradients admits an LoS of $d_{k,j}$ as shown in Eq. (17).

5. Convergence Analysis

In this section, we conduct convergence analysis for the proposed method. The analysis shows that the ADL can converge to critical points based on the following assumptions.

Assumption 1. *Lipschitz continuity of gradients for loss functions $f(\boldsymbol{\theta})$, which means $\exists L \in \mathbb{R}^+$ such that:*

$$||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^{U_\alpha} - \bar{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^{U_\beta}||_2 \leq L||\boldsymbol{\theta}_l^{U_\alpha} - \boldsymbol{\theta}_l^{U_\beta}||_2 \quad (22)$$

where $\|.\|_2$ is an l_2-norm operator. This also leads to

$$||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_\alpha} - \bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_\beta}||_2 \leq L||\boldsymbol{\theta}_{q(k)}^{U_\alpha} - \boldsymbol{\theta}_{q(k)}^{U_\beta}||_2,\ ||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_\alpha} - \bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_\beta}||_2 \leq L||\boldsymbol{\theta}^{U_\alpha} - \boldsymbol{\theta}^{U_\beta}||_2.$$

Assumption 2. *Bounded variance of the stochastic gradient, which means that $\forall s,\ \exists A > 0$ such that:*

$$||\boldsymbol{g}_{\boldsymbol{\theta}_l}^{U_s}||_2^2 \leq A, \text{ which leads to } ||\boldsymbol{g}_{\boldsymbol{\theta}_{q(k)}}^{U_s}||_2^2 \leq A,\ ||\boldsymbol{g}_{\boldsymbol{\theta}}^{U_s}||_2^2 \leq A. \quad (23)$$

Assumptions 1 and 2 are commonly adopted in convergence analysis of neural networks (see (Bottou et al., 2018; Huo et al., 2018b)). In particular, these assumptions do not assume convexity of function f.

Theorem 1. *Let Assumptions 1 and 2 hold. Suppose that the learning rate is non-increasing and $L\gamma_s \leq 1$. The proposed ADL has the following lower bound:*

$$\mathbb{E}_{\boldsymbol{x}}\{f(\boldsymbol{\theta}^{U_{s+1}})\} - f(\boldsymbol{\theta}^{U_s}) \leq -\tfrac{\gamma_s}{2}\|\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2 + \gamma_s^2 AL(1 + (1/M)\textstyle\sum_{k=1}^{K}\bar{d}_k)/M. \quad (24)$$

Proof. See supplementary material A. □

Theorem 1 gives an important indication for convergence. If the RHS of Eq. (24) is negative, i.e.,

$$\gamma_s < \min\left\{1/L,\ M\|\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2/(2AL(1 + (1/M)\textstyle\sum_{k=1}^{K}\bar{d}_k))\right\},$$

the expected loss $\mathbb{E}_{\boldsymbol{x}}\{f(\boldsymbol{\theta}^{U_{s+1}})\}$ decreases. We further give the convergence evidence in the following theorems. Detailed proofs of the following theorems can be found in the supplementary material [3].

Theorem 2. *Suppose Assumptions 1 and 2 hold, and the learning rate is non-increasing as well as satisfies $L\gamma_s \leq 1$. Let $\boldsymbol{\theta}^*$ be the global minimizer and $\mathbb{T}_S = \sum_{s=0}^{S-1}\gamma_s$ where S indicates the network will be updated S times. Then*

$$(1/\mathbb{T}_S)\textstyle\sum_{s=0}^{S-1}\gamma_s\mathbb{E}\{||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_s}||_2^2\} \leq 2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))/\mathbb{T}_S$$
$$+ (2AL(1 + (1/M)\textstyle\sum_{k=1}^{K}\bar{d}_k)\sum_{s=0}^{S-1}\gamma_s^2)/(M\mathbb{T}_S). \quad (25)$$

Proof. See supplementary material B. □

The lower bound in Theorem 2 indicates that, for a randomly selected $\mathfrak{q}$ from $\{0, 1, \ldots, S-1\}$ with probability $\{\gamma_\mathfrak{q}/\mathbb{T}_S\}$, $\mathbb{E}\{||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_s}||_2^2\}$ is bounded by the RHS of Eq. (25). More importantly, a larger M leads to a smaller lower bound in Eq. (25) because the $\bar{d}_k$ decreases, and thus benefits the convergence. Another observation is that larger PS hinders the convergence as $\sum_{k=1}^{K}\bar{d}_k$ increases. These observations are consistent with our understanding that the mGA helps the optimization by mitigating staleness, and partitioning the network into more modules is harmful.

Corollary 1. *If γ_s further satisfies $\lim_{S\to\infty}\mathbb{T}_S = \infty$ and $\lim_{S\to\infty}\sum_{s=0}^{S-1}\gamma_s^2 < \infty$, the RHS of Eq. (25) converges to 0.*

According to Corollary 1, by properly scheduling the learning rate, the lower bound for the expected gradient would converge to 0, i.e., $\lim_{S\to\infty}\mathbb{E}\{||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_s}||_2^2\} = 0$. That is, the proposed ADL can converge to critical points. Alternatively, the convergence can be revealed by setting a constant learning rate as indicated in the following theorem.

Theorem 3. *Let Assumptions 1 and 2 hold. Suppose the learning rate is set as a constant:*

$$\gamma = \epsilon\sqrt{M(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))/\left(SAL(1 + \textstyle\sum_{k=1}^{K}\bar{d}_k)\right)}$$

where ϵ is a scaling factor such that $L\gamma \leq 1$. Let $\boldsymbol{\theta}^*$ be the global minimizer. Then we have

$$\min_{s\in\{0,1,\ldots,S-1\}}\mathbb{E}\{\|\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\} \leq \tfrac{(2+2\epsilon^2)}{\epsilon}\sqrt{AL(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))\left(1 + (1/M)\textstyle\sum_{k=1}^{K}\bar{d}_k\right)/(MS)}, \quad (26)$$

where the lower bound converges to 0 when $S \to \infty$.

Proof. See supplementary material C □

In summary, although the ADL attains model parallelism and introduces asynchronism, we show that our method can converge to critical points as well as revealing how it can be affected by the mGA and the PS.

[3] https://personal.ntu.edu.sg/ezplin/ICML2021-appendices.pdf

Table 2. Validation errors (%) for (a) the ADL training RN-56 on CIFAR-10, and for the compared methods training various networks on (b) CIFAR-10 and (c) ImageNet. ⊗ indicates divergence.

(a)

$M \rightarrow$	1	2	3	4	5	6	7	8	9	10	11	12
RN-56 ($K = 16$)	⊗	⊗	⊗	6.38	6.90	6.92	6.79	6.49	6.78	6.44	6.65	6.86

(b)

Architecture	BP	DDG	DGL	FR	DSP*	GPipe‡	ADL
RN-56 ($K = 2$)	6.19	6.63	6.77	6.07	-	6.04	**5.99**
RN-56 ($K = 3$)	6.19	6.50	8.88	6.33	-	**5.94**	6.09
RN-56 ($K = 4$)	6.19	6.61	9.65	6.48	-	**6.03**	6.16
RN-56 ($K = 8$)	6.19	⊗	13.26	6.64	-	**6.08**	6.18
RN-56 ($K = 16$)	**6.19**	⊗	13.36	11.51	-	6.23	6.38
RN-110 ($K = 2$)	5.79	6.26	6.26	5.76	-	**5.70**	5.87
RN-110 ($K = 8$)	5.79	⊗	11.96	6.56	-	**5.69**	5.80
RN-98 ($K = 4$)	6.01	6.19	9.7	6.01	6.59	6.00	**5.92**
RN-98 ($K = 8$)	6.01	⊗	12.37	6.35	-	5.99	**5.93**
RN-34 ($K = 12$)	4.50	⊗	9.56	⊗	-	4.65	4.63
RN-164 ($K = 4$)	**5.36**	5.43	8.77	5.60	5.58	5.52	5.46
RN-164 ($K = 10$)	**5.36**	5.58	10.70	5.89	-	5.39	5.52

(c)

Architecture	BP	ADL
RN-18 ($K = 3$)	29.79/10.92	**29.52/10.42**
RN-18 ($K = 4$)	29.79/10.92	**29.64/10.56**
RN-18 ($K = 8$)	29.79/10.92	**29.75/10.55**
RN-18 ($K = 10$, max.)†	**29.79**/10.92	29.84/**10.76**
RN-50 ($K = 4$)	23.65/**7.13**	**23.43**/7.45
SE-RN-18 ($K = 8$)	29.09/**9.89**	**29.01**/10.14
SE-RN-18 ($K = 10$, max.)†	29.09/**9.89**	**29.07**/10.32

* We only provide results from DSP's paper due to no available source code.

‡ The GPipe is included as a synchronous baseline.

† The largest PS with each module being one layer or one residual block.

6. Experiments

For validation we conduct learning of classification tasks on the CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet 2012 (Russakovsky et al., 2015) datasets. We mainly focus on examining the generalization of networks trained by ADL in order to obtain empirical evidence of gradient staleness mitigation in decoupled learning. We then touch lightly on its acceleration performance and point out an imbalance issue affecting the acceleration, which is currently unresolved in the area of decoupled learning. Finally, we present the memory consumption. We compare our method with several state-of-the-arts, including DDG (Huo et al., 2018b), FR (Huo et al., 2018a), DGL (Belilovsky et al., 2020), DSP (Xu et al., 2020), and BP (Werbos, 1974). In particular, we include the GPipe (Huang et al., 2019) as a baseline for synchronous model parallelism. We focus on the comparison among methods of decoupled learning, and exclude the ASGD- and pipeline-based methods though they also involve gradient staleness.

Implementation details: The experiments are performed with PyTorch (Paszke et al., 2019) where we pre-process the datasets using standard data augmentation (i.e., random cropping, random horizontal flip and normalizing). We adopt the same training strategy for all the compared methods (except DSP due to no source code) for fairness. The SGD optimizer with a momentum of 0.9 is adopted, and gradual warm-up in (Goyal et al., 2017) for 3 epochs is used. We adopt an overall batch size of 128 for all methods. Specially, since the mGA accumulates the data batches, for ADL the batch size in each iteration is set to $\lfloor 128/M \rfloor$. The GPipe adopts a similar setting with an overall batch size of 128 scattered into M micro-batches. For CIFAR-10, the weight decay is set to 5×10^{-4} with an initial learning rate of 0.1, and the networks are trained for 300 epochs with the learning rate divided by 10 at 150, 225 and 275 epochs. For ImageNet, a 224××224 crop is randomly sampled, and the weight decay is set to 1×10^{-4} with an initial learning rate of 0.05. We train the networks for 90 epochs, and divide the learning rate by 10 at 30, 60, and 80 epochs. Finally, the validation results are reported by the median of 3 runs.

Datasets: The CIFAR-10 dataset includes 32x32 color images with 10 classes, and has 50000 and 10000 samples for training and validation respectively. The ImageNet dataset contains 1000 classes, with 1.28 million and 50000 images of various sizes for training and validation.

6.1. Generalization Performance

Firstly, we evaluate the impact of mGA by training RN-56 ($K = 16$) on CIFAR-10 with various M. As shown in Table 2(a), the ADL diverges with $M \leq 3$ due to strong gradient staleness. Once the learning converges, the ADL performs rather robustly to various K. For convenience, we pick $M = 4$ (i.e,, reducing 75% staleness as shown in Eq. (19)) for the ADL in the following experiments, which should stabilize the decoupled learning for $K \leq 16$.

Later on, on CIFAR-10 we train various RN architectures with different PS ranging from $K = 2$ to $K = 16$. The classification results are shown in Table 2(b). In general our ADL delivers results comparable to or better than those from the compared methods including the synchronous methods (i.e., BP and GPipe), while the other methods of decoupled learning would encounter learning issues as K increases. The GPipe shows results as good as the BP baselines without receiving visible impacts as PS increases. This is reasonable as its synchronous nature does not invite gradient staleness. The DGL experiences a growing loss of performance as the PS increases, e.g., $6.77\% \rightarrow 9.65\% \rightarrow 13.26\%$ for RN-56 with $K = 2, 4, 8$ respectively. The DGL severs the gradient flow between adjacent modules. It renders the lower modules under-developed without any feedback from the upper modules, causing inferior learning results to the BP baselines.

The FR gives promising results for small PS (e.g., $K = 2$) which is consistent with the claim made in (Huo et al., 2018a). However, its performance shows a significant deterioration when training with a very large PS. For instance, training RN-56 with $K = 16$ leads to an error rate of 11.51% which is significantly worse than that of 6.07% with $K = 2$. In the case of RN-34 ($K = 12$), the FR even fails to con-

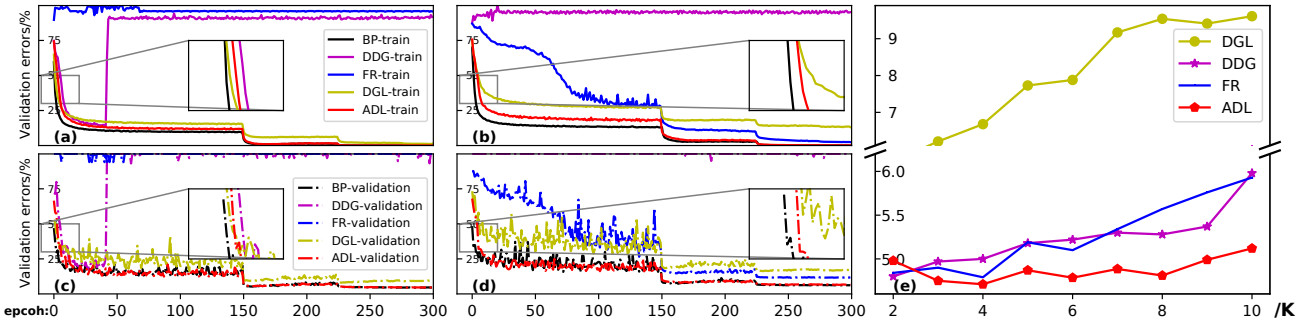

Figure 3. Learning curves of compared methods for training RN-34 with $K = 12$ in sub-figures (a) and (e), and RN-56 with $K = 16$ in sub-figures (b) and (d), as well as the error rates w.r.t. PS in sub-figure (e) on RN-18. Note that the DSP is excluded as no source code is available.

verge. The performance drop or divergence is likely resulted from the recomputation unit in FR which adopts inconsistent weights during the forward and backward passes. The DSP gives a moderate performance for training RN-98. We cannot provide other corresponding results as no source code is released and it is difficult to reproduce the algorithm based on the manuscript, but we would expect a similar behaving pattern of DSP to that of FR as they share the same recomputation technique.

The DDG, on the other hand, is more prone to divergence. As shown in Table 2(b), the trainings of RN-56, ResNet98 and RN-110 exhibit several divergence cases (mainly happen with large PS, e.g., $K \geq 8$), while our method successfully trains these networks with comparable performance to the BP baselines. However, when the DDG does converge (e.g., training RN-164 with $K = 10$), unlike the FR or DGL with rather sharply decreased performance, it tends to deliver very close results to those of BP (e.g., 5.58% versus 5.36%). Such an observation reveals that increasing staleness invites instability yet it only affects the generalization to a limited extent. This empirically supports and further justifies our motivation of developing the proposed ADL aiming to mitigate gradient staleness thereby avoiding divergence.

We provide the learning curves of RN-34 ($K = 12$) and RN-56 ($K = 16$) in Figure 3. The DDG diverges in both cases, i.e., at epoch 48 and 2 for RN-34 and RN-56. The FR diverges at epoch 2 for RN-34, and experiences a "bumpy" convergence for RN-56. Our ADL converges smoothly in both cases while achieving equally good validation results to the BP baselines. We also observe a clear pattern in these learning curves showing that the ADL gives a slightly slower convergence than that from BP at the beginning of network learning (see the zoomed plots in Figure 3). This is consistent with our theoretical findings (e.g., Theorem 2) suggesting the decoupled learning would slow down the convergence especially with a large PS. We also provide the study to examine the impact of PS on the error rate by training RN-18 with various K. Figure 3(e) shows that our method is relatively robust to the change of K with the staleness mitigated, while the compared methods encounter much rapid rise of errors as K increases.

We also provide the results of ADL trained on ImageNet in comparison with BP. We train several architectures from ResNet and SENet (Hu et al., 2018), and report both Top1 and Top5 error rates. As shown in Table 2(c), in general the ADL outperforms its BP counterpart even for a large PS of $K = 10$. The observation of ADL outperforming the BP can be explained as follows. The ADL adopts DGs, which can be treated as real gradients obtained by BP, yet contaminated with noises drawn from an unknown distribution. Although the contaminated gradients could slow down the convergence (e.g., see the zoomed plots in Figure 3), they could however improve the network's generalization with their uncertainties (see (Neelakantan et al., 2015)). The mGA restraints the staleness so it does not go out of bound while keeping its uncertainty, thereby leading to more promising results in certain cases.

6.2. Acceleration Performance

Here we show the acceleration performance of the ADL in comparison with other methods of decoupled learning by training RN-101 on ImageNet with various K. The experiments are conducted on a server with Tesla V100 GPUs with each module running in a separate GPU worker. Batch size is adjusted to maximize the training speed, and the network split locations are tuned to distribute the computational workload (i.e., running time for each iteration) as evenly as possible in each GPU. Note that the mGA has little effect on the running speed, so we set $M = 4$ as usual.

As shown in Table 3, by addressing all three lockings of BP, the ADL achieves a remarkable acceleration over BP and is much faster than the DDG and FR which address only the backward locking. For instance, for $K = 2$ the ADL

Table 3. Speedups (over BP) in training RN-101 (ImageNet).

	BP	DDG	FR	DGL	DSP	GPipe	ADL
K=2	1×	1.32×	1.19×	1.82×	-	1.41×	**1.92**×
K=3	1×	1.57×	1.29×	2.61×	-	1.87×	**2.69**×
K=4	1×	1.68×	1.45×	**3.39**×	2.70×	2.23×	3.32×

Table 4. Memory consumption of the compared methods.

	BP	DDG	FR	DGL	GPipe	ADL
Mem. (GB)	**1.14**	1.69	1.46	1.49	1.52	1.84

accelerates the learning with a speedup of **1.92**× while the DDG and FR only obtain **1.32**× and **1.19**× respectively. Although the DSP also addresses all the unlockings, it is lower than the ADL due to the recomputation unit that demands extra computing power. The DGL delivers comparable acceleration results to those of ADL, but it leads to weaker generalization performance (see Table 2(b)). In comparison, the GPipe is slower than our method due to the introduced computational "bubbles" (Huang et al., 2019) resulted from the synchronism.

Here we only demonstrate the acceleration results for $K \leq 4$ following the DDG, FR and DSP without moving forward to $K \geq 5$. This is because the depth-wise partition introduces imbalance of computation workload among different workers since the smallest partitionable unit a either one layer or one residual block. The imbalance increases rapidly with larger K leading to inefficiency in decoupled learning. For instance, 2 GPUs accelerate the learning to **1.92**× but doubling the resources to 4 GPUs only receives **3.32**× instead of a double acceleration, indicating that certain modules are idle for some time. Our ADL obtains the fastest learning speed, but still requires a further assistance on balancing the workload distribution if we are to fully demonstrate its acceleration potential (e.g., obtaining a speedup of $\approx 7.5\times$ for $K = 8$). Addressing or mitigating the imbalance issue is not trivial and is to be investigated in future work.

6.3. Memory Analysis

Here we conduct a brief comparison regarding the memory consumption among the compared methods. The results are measured based on the average GPU memory in training RN-18 of $K = 4$ with a batch size of 128. As indicated in Table 4, the compared methods used more memory than that used by BP. Compared with the FR and GPipe, the DDG and ADL require additional memory. This is because, unlike the trainings in FR and GPipe, no recomputation is required, which however allow them, especially the ADL, to run much faster (e.g., 3.32× of ADL v.s. 1.45× of FR for 4 GPUs). Reducing the memory usage without affecting the ADL's generalization and speed is a nontrivial task, which will be our future work.

7. Conclusion

In this paper, we proposed an accumulated decoupled learning (ADL) algorithm to address the inefficiency in BP lockings thereby achieving model parallelism. Particularly, a module-wise gradient accumulation (mGA) technique, which mitigates the gradient staleness that hinders the scaling ability of decoupled learning, has been incorporated. The effect of mitigation has been demonstrated theoretically and empirically. Theoretically, we showed that the ADL converged to critical points, and also demonstrated how the partition size (PS) and the mGA could affect the lower bound categorizing the convergence. The classification tasks conducted validated our claim as the ADL outperformed the state-of-the-art counterparts especially in the case of a large PS. We finally showed the running speed, demonstrating a remarkable acceleration for the ADL as a model parallelism tool.

Acknowledgment

We thank the anonymous reviewers for their very constructive comments for improving this manuscript. This work was supported in part by the Science and Engineering Research Council, Agency of Science, Technology and Research, Singapore, through the National Robotics Program under Grant 1922500054.

References

Belilovsky, E., Eickenberg, M., and Oyallon, E. Decoupled greedy learning of CNNs. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 736–745. PMLR, 13–18 Jul 2020.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, pp. 103–112, 2019.

Huo, Z., Gu, B., and Huang, H. Training neural networks using features replay. In *Advances in Neural Information Processing Systems*, pp. 6659–6668, 2018a.

Huo, Z., Gu, B., Huang, H., et al. Decoupled parallel backpropagation with convergence guarantee. In *International Conference on Machine Learning*, pp. 2103–2111, 2018b.

Jaderberg, M., Czarnecki, W., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. Decoupled neural interfaces using synthetic gradients. In *ICML*, 2016.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, 2009.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lian, X., Huang, Y., Li, Y., and Liu, J. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 2737–2745, 2015.

Mostafa, H., Ramesh, V., and Cauwenberghs, G. Deep supervised learning using local errors. *Frontiers in neuroscience*, 12:608, 2018.

Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Nøkland, A. and Eidnes, L. H. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.

Werbos, P. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

Xu, A., Huo, Z., and Huang, H. On the acceleration of deep learning model parallelism with staleness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2088–2097, 2020.

Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z.-M., and Liu, T.-Y. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 4120–4129. JMLR. org, 2017.

Zhuang, H., Wang, Y., Liu, Q., and Lin, Z. Fully decoupled neural network learning using delayed gradients. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–8, 2021. doi: 10.1109/TNNLS.2021.3069883.