Fully Decoupled Neural Network Learning Using Delayed Gradients

Huiping Zhuang, *Graduate Student Member, IEEE*, Yi Wang, *Member, IEEE*, Qinglai Liu, *Member, IEEE*, Zhiping Lin*, *Senior Member, IEEE*

***Abstract*—Training neural networks with back-propagation (BP) requires a sequential passing of activations and gradients. This has been recognized as the lockings (i.e., the forward, backward, and update lockings) among modules (each module contains a stack of layers) inherited from the BP. In this paper, we propose a fully decoupled training scheme using delayed gradients (FDG) to break all these lockings. The FDG splits a neural network into multiple modules and trains them independently and asynchronously using different workers (e.g., GPUs). We also introduce a gradient shrinking process to reduce the stale gradient effect caused by the delayed gradients. Our theoretical proofs show that the FDG can converge to critical points under certain conditions. Experiments are conducted by training deep convolutional neural networks to perform classification tasks on several benchmark datasets. These experiments show comparable or better results of our approach compared with the state-of-the-art methods in terms of generalization and acceleration. We also show that the FDG is able to train various networks, including extremely deep ones (e.g., ResNet-1202), in a decoupled fashion.**

***Index Terms*—Decoupled learning, delayed gradients, gradient shrinking, neural network lockings.**

I. Introduction

Deep neural networks, including convolutional neural network (CNN) [1] and recurrent neural network (RNN) [2], have demonstrated great success in various highly complex tasks. Such success is built, to a great extent, on the ability to train extremely deep networks enabled by ResNet [3] or other techniques with skip-connection-like structures [4]. Training networks with back-propagation (BP) [5] is a standard practice but it requires a complete forward and backward pass before the parameters are updated. This easily leads to inefficiency [6] especially for training deeper networks, which is recognized as the lockings [7] (i.e., forward, backward, and update lockings) among modules inherited from the standard BP. Each of these modules includes a stack of layers partitioned from the original network, and the existing lockings (see details in Section II) keep the majority of the modules on hold during training, leading to learning inefficiency.

In order to improve the efficiency, there have been a number of works on decoupling the training by splitting the network into multiple modules to facilitate inter-layer/inter-module model parallelization. These methods are also known as decoupled learning where the term *decoupled* indicates that one or more of the BP lockings among modules are removed. In terms of parallelization, yet the decoupled learning has several benefits over methods based on the data-parallel or mixed-parallel paradigm [8]–[10]. For instance, it does not change the batch size such that the performance of certain modules, such as Batch Normalization [11], can be maintained. In addition, it is able to straightforwardly parallelize the RNNs [7]. The decoupling techniques might be categorized into two groups: the backward-unlocking (BU) based methods [12], [13] and the local error learning (LEL) based methods [7].

The BU-based methods have access to the global information from the top layer and could break the backward locking. An additional benefit is that they often introduce no extra trainable parameters while enabling decoupled learning. Nonetheless, a full forward pass is still required before any parameter update. One important motivation for these techniques is to promote biological plausibility [14]–[16], which focuses on removing the weight symmetry and possible gradient propagation from the BP procedure. Feedback alignment (FA) [14] removes the weight symmetry by replacing symmetrical weights with random ones. Direct feedback alignment [15] following the FA replaces the BP with a random projection and potentially facilitates a simultaneous update for all layers. However, these biologically inspired approaches tend to suffer from performance losses, and are shown to scale poorly on more complex datasets [17]. Although recently a neural circuit called weight mirror shows impressive results in large-scale tasks, it is less biologically plausible than the other biologically inspired approaches. On the other hand, the *delayed gradient* provides another key to breaking the backward locking. The decoupled parallel BP using delayed gradients (DDG) [12] can train rather deep (up to 110 layers) CNNs, and shows no performance loss while reducing the training time. Since the DDG is still constrained by the forward locking, the acceleration is relatively limited even with multiple workers. The feature replay (FR) [13] following the DDG also breaks the backward locking through recomputation, and it has been shown to even outperform BP for several deep architectures. However, the FR introduces more computational burden and is even slower than the DDG.

The LEL-based methods make use of the local information, and are able to fully decouple the neural network training. Such a full decoupling is achieved by building auxiliary networks to generate local error gradients. These auxiliary networks sever the gradient flow between adjacent modules thereby achieving asynchronous model parallelization. The decoupled neural interface (DNI) [7] is among the pioneers exhibiting module-wise parallelization potential for neural networks. It utilizes a local network to generate synthetic

H. Zhuang, Y. Wang, Z. Lin are with School of EEE; Q. Liu is with Temasek Labs. All are from Nanyang Tech. Univ., Singapore. *EZPLin@ntu.edu.sg.

TABLE I: Comparison with state-of-the-art methods in terms of lockings and auxiliary networks.

Methods	DDG [12]	FR [13]	DNI [7]	DGL [20]	FDG (ours)
Lockings	Yes	Yes	No	No	No
Auxiliary networks	No	No	Yes	Yes	No

gradients for the hidden layers so that the update could happen before completing either the forward or the backward pass. However, the DNI has been shown to learn poorly and even exhibits convergence problems in deeper networks [12]. In [18], local classifiers with cross-entropy loss are adopted to generate gradients for hidden layers, showing potentials to train hidden layers simultaneously. It has been shown that the local classifier alone fails to match the performance of a standard BP. In [19], a similarity measure combined with the local classifier (pred-sim) is introduced to provide local error gradients. The mixed loss functions yield classification performance comparable to or even better than the BP baselines. However, this pred-sim method is only validated in VGG-like networks ($\leq$13 layers). Recently, the depth problem of the LEL-based methods has been alleviated by decoupled greedy learning (DGL) [20], which can train deeper networks ($\geq$100 layers) while maintaining comparable performance to that of a standard BP. In general, the common sacrifice for adopting the LEL technique is the introduction of extra trainable parameters imposed by the auxiliary networks. For instance, to match the standard BP, the local learning in [19] needs to train several times more parameters.

In summary, both BU-based and LEL-based methods can decouple the training of neural networks while showing potential in obtaining comparable performances to the BP baselines. In comparison, the LEL-based methods fully decouple the network learning but introduce extra trainable parameters. The BU-based methods behave in the opposite way. In this paper, we propose a fully decoupled learning scheme using delayed gradients (FDG) sharing both merits of the BU-based and the LEL-based techniques (see TABLE I). The main contributions of this work are as follows:

- We propose the FDG, a novel training scheme that breaks the forward, backward, and update lockings without introducing extra trainable parameters. A gradient shrinking (GS) process is developed to reduce the stale gradient effect caused by utilizing the delayed gradients.
- We illustrate that, in the ideal case, the FDG is able to achieve a linear speedup with respect to (w.r.t.) the number of split modules.
- We provide convergence analysis of the proposed method, and show that our method can converge to critical points for the non-convex optimization problem.
- We conduct experiments by training deep CNNs, revealing that the proposed FDG gives comparable or better results compared with other decoupling methods as well as the standard BP in terms of both generalization and acceleration.

The rest of the paper is summarized as follows. Section II gives the background knowledge to develop the FDG. Section III elaborates the algorithmic details of the proposed FDG. The convergence analysis is given in Section IV. Experimental validation is presented in Section V. Finally, the conclusion is drawn in Section VI.

II. Background

Here, we revisit some background knowledge for training a feedforward neural network. During this revisit, the forward, backward, and update lockings [7] are also explained.

Assume that we need to train an L-layer network. The l^{th} ($1 \leq l \leq L$) layer produces an activation $\boldsymbol{z}_l = A_l(\boldsymbol{z}_{l-1}; \boldsymbol{\theta}_l)$ by taking $\boldsymbol{z}_{l-1}$ as its input, where A_l is an activation function and $\boldsymbol{\theta}_l \in \mathbb{R}^{n_l}$ is a column vector representing the weights in layer l. The sequential generation of the activations leads to the *forward locking* since $\boldsymbol{z}_l$ will not be available before all the dependent activations are obtained. Let $\boldsymbol{\theta} = [\boldsymbol{\theta}_1^T, \boldsymbol{\theta}_2^T, ..., \boldsymbol{\theta}_L^T]^T \in \mathbb{R}^{\Sigma_{i=1}^L n_i}$ denote the parameter vector for the whole network. Assume f is a loss function that maps a high-dimensional vector to a scalar. The learning of the feedforward network can then be summarized as the following optimization problem:

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

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. At iteration t, we have

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

or equivalently,

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

where γ_t is the learning rate at iteration t, and $\bar{\boldsymbol{g}}_\theta^t = [(\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_1}^t)^T, (\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_2}^t)^T, ..., (\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_L}^t)^T]^T \in \mathbb{R}^{\Sigma_{i=1}^L n_i}$ is the gradient vector obtained by

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

If the training sample size is large, the stochastic gradient descent (SGD) is often used as a replacement such that

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

where $\boldsymbol{x}_t$ is a mini-batch of $\boldsymbol{x}$. Note that the bar "$\bar{}$" has been removed to indicate the difference from Eq. (4). Thus the parameter can be updated through

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

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

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

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

To obtain the gradient vectors, the BP can be employed. One could calculate the gradients in layer l using the gradients back-propagated from layers j and i ($l < j < i$):

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

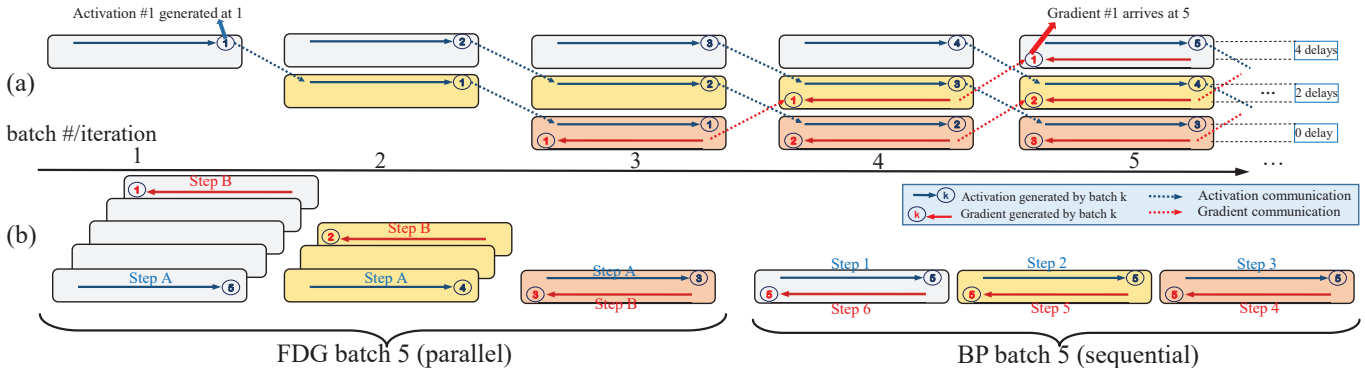

Fig. 1: Illustration of the proposed FDG: a $K = 3$ example. (a) By delaying the activations and gradients, the FDG allows the modules (in different colors) to be trained asynchronously in parallel. The communications among modules happen right before the iteration ends. We can see that there is a delay of $2(K-k)$ of gradients in module k with a split of K. 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. (b) In the FDG, the backward pass (step B) is executed in the previously saved computation graph while the forward pass (step A) happens in the current one. This is how the FDG overcomes the sequential nature of the standard BP.

where

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

Note that we introduce $\boldsymbol{g}^t_{\boldsymbol{z}_j}$ here —the gradient vector w.r.t. activation $\boldsymbol{z}_j$—because it travels between modules as an important part of our proposed FDG. Eq. (8) and Eq. (9) indicate that $\boldsymbol{g}^t_{\boldsymbol{\theta}_l}$ depends on $\boldsymbol{g}^t_{\boldsymbol{z}_j}$ and $\boldsymbol{g}^t_{\boldsymbol{z}_i}$. In other words, the gradient in layer l remains unavailable until all the dependent gradients are obtained. This is also known as the *backward locking*. In addition, the parameter update is not permitted before all layers complete executing their forward passes. This is recognized as the *update locking*. In the following, we show that a full decoupling (i.e., the forward, backward and update unlockings) can be achieved.

III. Fully Decoupled Neural Network Learning

In this section, we give the details of the proposed FDG. This technique gives a fully decoupled asynchronous learning algorithm with a gradient shrinking (GS) process to mitigate the accuracy decrease caused by the delayed gradients.

A. The Proposed FDG

We first split the network into K modules with each module containing a stack of layers. Accordingly, we split the set of the layer indices $\{1, \ldots, L\}$ 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.

As illustrated in Fig. 1(a), during the decoupled learning, module k performs a forward pass and a backward pass in each iteration. The forward pass is executed based on the delayed activation previously passed from module $k-1$ while the backward pass is conducted using the delayed gradient previously sent by module $k+1$. Note that all the modules are executed with the same procedures simultaneously, hence achieving the parallel training of different modules. After executing both passes, the gradient of the module input is passed to module $k-1$, while the module output is sent to module $k+1$ as its new input. This can be detailed by the following steps for module k ($1 < k < K$):

• **forward**: at iteration t, we feed the input $\boldsymbol{z}^{t-k+1}_{m_k-1}$ (previously sent by module $k-1$) into module k, which gives a module output (activation) $\boldsymbol{z}^{t-k+1}_{m_{k+1}-1}$. We adopt $t-k+1$ instead of t because at iteration t we are using the delayed activation generated by batch $t-k+1$ (see Fig. 1(a)).

• **backward**: at iteration t, we utilize the delayed gradient $\boldsymbol{g}^{t-2K+k+1}_{\boldsymbol{z}_{m_{k+1}-1}}$ previously received from module $k+1$ to resume the BP procedure. The superscript $t-2K+k+1$ is adopted because there is a delay of $2(K-k)$ of gradients (see Fig. 1(a)) w.r.t. the forward pass that uses batch $t-k+1$. Thus, for each layer ($m_k \le l \le m_{k+1}-1$), we obtain the gradient:

$$\hat{\boldsymbol{g}}^{t-k+1}_{\boldsymbol{\theta}_l} = \frac{\partial \boldsymbol{z}^{t-2K+k+1}_{m_{k+1}-1}}{\partial \boldsymbol{\theta}^{t-2K+k+1}_l}\boldsymbol{g}^{t-2K+k+1}_{\boldsymbol{z}_{m_{k+1}-1}}. \tag{10}$$

Note that Eq. (10) must utilize the delayed activation of batch $t-2K+k+1$. It is only reasonable to calculate the gradient based on the activation generated by the same training batch. Subsequently, the module can be updated through

$$\boldsymbol{\theta}^{t-k+2}_l = \boldsymbol{\theta}^{t-k+1}_l - \gamma_{t-k+1}\hat{\boldsymbol{g}}^{t-k+1}_{\boldsymbol{\theta}_l}. \tag{11}$$

After completing the BP in this module, we save $\boldsymbol{g}^{t-2K+k+1}_{\boldsymbol{z}_{m_k-1}}$, the gradient of the module input, for communication.

• **communication**: at iteration t, send $\boldsymbol{g}^{t-2K+k+1}_{\boldsymbol{z}_{m_k-1}}$ to module $k-1$ and pass $\boldsymbol{z}^{t-k+1}_{m_{k+1}-1}$ to module $k+1$ as its new input.

Note that the first ($k = 1$) and the last ($k = K$) modules are executed similarly but require less work in the communication step. The first module does not need to send the gradient as there is no lower module, and the last module receives its gradient propagated directly from the loss function.

The aforementioned forward, backward, and communication steps break all the lockings in [7]. Firstly, the global BP is broken into module-wise BP running in parallel, which achieves the *backward unlocking*. Secondly, each of the split modules processes the training data from different batches,

leading to an asynchronous module parallelization, hence the *forward unlocking*. Finally, each module can be immediately updated without waiting for other modules to complete their forward passes, so the *update unlocking* is also achieved.

Note that we utilize $t-k+1$ instead of t in the update formula Eq. (11) in correspondence to the activation index (e.g., $\boldsymbol{z}_{m_{k+1}-1}^{t-k+1}$) at iteration t. One could straightforwardly see that, for module k at iteration $t+k-1$, Eq. (11) can be equivalently shifted to

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t \tag{12}$$

with

$$\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t = \frac{\partial \boldsymbol{z}_{m_{k+1}-1}^{t-2K+2k}}{\partial \boldsymbol{\theta}_l^{t-2K+2k}} \boldsymbol{g}_{\boldsymbol{z}_{m_{k+1}-1}}^{t-2K+2k}. \tag{13}$$

Adopting Eq. (12) over Eq. (11) would benefit the subsequent convergence analysis in the next section. Let $d_{k,t} = t-2(K-k)$, we can further unpack $\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t$ such that

$$\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t = \frac{\partial \boldsymbol{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \boldsymbol{\theta}_l^{d_{k,t}}} \frac{\partial f_{\boldsymbol{w}_{d_{k,t}}}(\boldsymbol{\theta}^{d_{k,t}})}{\partial \boldsymbol{z}_{m_{k+1}-1}^{d_{k,t}}} = \frac{\partial f_{\boldsymbol{w}_{d_{k,t}}}(\boldsymbol{\theta}^{d_{k,t}})}{\partial \boldsymbol{\theta}_l^{d_{k,t}}} = \boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} \tag{14}$$

and rewrite Eq. (12) as

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} \tag{15}$$

which is observed to take delayed gradients with a delay of $2(K-k)$ compared with Eq. (6).

Analysis of Speedup: Assume a network is evenly split into K modules in terms of computing workload. In the ideal case where other time consumptions such as communication are excluded, TABLE II shows that a linear (K-time) speedup can be achieved with the FDG. This is the highest among all the decoupling methods.

TABLE II: Ideal speedup of different decoupling methods for a network evenly split into K modules. $\mathcal{T}_f$, $\mathcal{T}_b$ and $\mathcal{T}_{aux}$ denote the computation time executing the forward, the backward pass, and the auxiliary network, respectively.

Methods	BP [5]	DDG [12]	FR [13]	DNI [7] & DGL [20]	FDG (ours)
Time	$\mathcal{T}_f+\mathcal{T}_b$	$\mathcal{T}_f+\frac{\mathcal{T}_b}{K}$	$\mathcal{T}_f+\frac{\mathcal{T}_f+\mathcal{T}_b}{K}$	$\frac{\mathcal{T}_f+\mathcal{T}_b}{K}+\mathcal{T}_{aux}$	$\frac{\mathcal{T}_f+\mathcal{T}_b}{K}$

B. The Gradient Shrinking Process

Using the delayed gradients enables inter-layer model parallelization but could also lead to certain performance decrease. This is a common phenomenon observed in algorithms with stale gradients [21]. To compensate such performance loss, we introduce a gradient shrinking (GS) process before back-propagating the delayed gradients through each module.

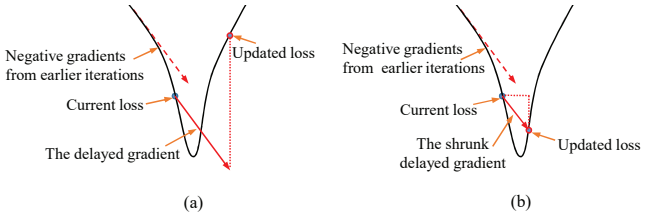

Fig. 2: An intuitive interpretation for the GS process.

Algorithm I: FDG with SGD

Required: learning rate γ_t, number of split modules K, gradient shrinking factor β.
• Split the network into K modules and initialize them.
for $t = 1, 2, ..., T$:
  Parallel for $k = 1, \ldots, K$:
    • execute the forward pass to generate the module output $\boldsymbol{z}_{m_{k+1}-1}^{t-k+1}$
    • compute the shrunk delayed gradient in each layer $\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^{t-k+1}$ with Eq. (16), and gradient of the module input $\hat{\boldsymbol{g}}_{\boldsymbol{z}_{m_k-1}}^{t-2K+k+1}$.
    • update the module through Eq. (11).
    • pass $\boldsymbol{z}_{m_{k+1}-1}^{t-k+1}$ to module $k+1$ as its input.
    • send $\hat{\boldsymbol{g}}_{\boldsymbol{z}_{m_k-1}}^{t-2K+k+1}$ to module $k-1$.
  End for
End for

The GS process works in a direct manner. At iteration t, before executing the local BP in module k, we shrink the gradient by multiplying it with a shrinking factor β ($0 < \beta \leq 1$). This can be shown by modifying Eq. (13) as

$$\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t = \beta \frac{\partial \boldsymbol{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \boldsymbol{\theta}_l^{d_{k,t}}} \hat{\boldsymbol{g}}_{\boldsymbol{z}_{m_{k+1}-1}}^{d_{k,t}}. \tag{16}$$

The module is then updated through Eq. (12). Note that unlike Eq. (13) where the received gradient is denoted by $\boldsymbol{g}_{\boldsymbol{z}_{m_{k+1}-1}}^{d_{k,t}}$, the received gradient in Eq. (16) is represented by $\hat{\boldsymbol{g}}_{\boldsymbol{z}_{m_{k+1}-1}}^{d_{k,t}}$. This is because the gradient in module k is affected by multiple GS processes starting from the top module where the gradient is not shrunk. This can be illustrated by unpacking Eq. (16):

$$\begin{aligned}
\hat{\boldsymbol{g}}_{\boldsymbol{\theta}_l}^t =& \beta^2 \frac{\partial \boldsymbol{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \boldsymbol{\theta}_l^{d_{k,t}}} \frac{\partial \boldsymbol{z}_{m_{k+2}-1}^{d_{k,t}}}{\partial \boldsymbol{z}_{m_{k+1}-1}^{d_{k,t}}} \hat{\boldsymbol{g}}_{\boldsymbol{z}_{m_{k+2}-1}}^{d_{k,t}} \\
&\vdots \\
=& \beta^{K-k} \frac{\partial \boldsymbol{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \boldsymbol{\theta}_l^{d_{k,t}}} \cdots \frac{\partial \boldsymbol{z}_{m_K-1}^{d_{k,t}}}{\partial \boldsymbol{z}_{m_{K-1}-1}^{d_{k,t}}} \boldsymbol{g}_{\boldsymbol{z}_{m_K-1}}^{d_{k,t}} \\
=& \beta^{K-k} \boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}}
\end{aligned} \tag{17}$$

which shrinks the gradient by a factor of β^{K-k} in module k. Note that if $\beta = 1$, the GS process is not used as $\beta^{K-k} = 1$.

We can interpret the GS process in an intuitive way shown in Fig. 2. The delayed gradients, especially with longer delays, would lead to deteriorated performance [21]. Fig. 2(a) shows a scenario where the delayed gradients cause the learning to miss the local minimum to a large margin. By using the shrunk delayed gradients, there is a better chance of reducing the stale gradient effect (see Fig. 2(b) for an illustration). The GS process works similarly to scaling down the learning rate, but they are not necessarily identical when certain regularization techniques are involved. Take the widely used *weight decay* [22] as an example. Involving such a technique, the update rule in Eq. (15) is modified as

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} - \gamma_t \lambda \boldsymbol{\theta}_l^t \tag{18}$$

where λ is the decay factor usually taking a small value. The GS process, i.e., $\boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} \rightarrow \beta^{K-k}\boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}}$, rewrites Eq. (18) into $\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t\beta^{K-k}\boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} - \gamma_t\lambda\boldsymbol{\theta}_l^t$. This differs from reducing the learning rate correspondingly, i.e., $\gamma_t \rightarrow \beta^{K-k}\gamma_t$, which rewrites Eq. (18) into $\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t\beta^{K-k}\boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} - \gamma_t\beta^{K-k}\lambda\boldsymbol{\theta}_l^t$. To mitigate the stale gradient effect, we adopt the GS process rather than scaling the learning rate for the following reasons. The GS process allows us to appreciate the stale gradient mitigation more directly and intuitively, i.e., how much we trust the calculated gradient, while the learning rate scaling tends to also rescale other factors (e.g., weight decay) apart from the gradients, which complicates the interpretation. In addition, our empirical results (see Fig. 3(b)) show that, in general, adopting the GS process leads to some performance gain in comparison. The proposed FDG with SGD optimizer is summarized in Algorithm I.

Comparison among Methods Adopting Delayed Gradient: Although we adopt delayed gradients like the DDG [12], FR [13], and asynchronous stochastic gradient descent (ASGD) methods [21], the FDG is different from these techniques in several ways. The DDG and FR only break the backward locking, but the FDG is able to break all the lockings, leading to a more efficient training. Unlike the ASGD-based methods which let each worker handle the entire network, the FDG trains the network by splitting it into modules where each module is handled by one specific worker.

IV. Convergence Analysis

In this section, the convergence analysis is given for the proposed FDG. We show that the proposed method can converge to critical points under certain conditions.

Prior to the analysis, we rewrite $\boldsymbol{\theta}^t, \boldsymbol{g}_{\boldsymbol{\theta}}^t$ and $\bar{\boldsymbol{g}}_{\theta}^t$ in terms of modules as follows:

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

Our convergence analysis is based on the below assumptions.

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

$$||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{t_1} - \bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{t_2}||_2 \leq L||\boldsymbol{\theta}^{t_1} - \boldsymbol{\theta}^{t_2}||_2 \tag{19}$$

$$||\boldsymbol{g}_{\boldsymbol{\theta}_{q(k)}}^{t_1} - \boldsymbol{g}_{\boldsymbol{\theta}_{q(k)}}^{t_2}||_2 \leq L||\boldsymbol{\theta}_{q(k)}^{t_1} - \boldsymbol{\theta}_{q(k)}^{t_2}||_2 \tag{20}$$

where $\|.\|_2$ is the 2-norm operator.

Assumption 2. *Bounded variance of the stochastic gradient, which means that $\forall t$, there exists a constant $M > 0$ such that:*

$$||\boldsymbol{g}_{\boldsymbol{\theta}}^t||_2^2 \leq M. \tag{21}$$

These two assumptions are commonly adopted in convergence analysis [12], [23] in deep learning. Accordingly, we can obtain the FDG's convergence property in the following theoretical findings. Detailed proofs of the following theorems can be found in the supplementary material[1].

[1]https://personal.ntu.edu.sg/ezplin/TNNLS-appendices.pdf

Theorem 1. *Let Assumptions 1 and 2 hold. Assume that the learning rate is non-increasing and $L\gamma_t \leq 1$. The proposed FDG in Algorithm I satisfies*

$$\mathbb{E}_{\boldsymbol{x}_t}[f(\boldsymbol{\theta}^{t+1})] - f(\boldsymbol{\theta}^t) \leq -\frac{\gamma_t}{2}Z_1 + \gamma_t^2 Z_2 \tag{22}$$

where

$$Z_1 = \sum_{k=1}^{K}\beta^{K-k}\left\|\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{q(k)}}^t\right\|_2^2,\ Z_2 = LM\sum_{k=1}^{K}\beta^{K-k} + LM\sum_{k=1}^{K}\beta^{3(K-k)}2(K-k).$$

Proof. See supplementary material A. □

Theorem 1 gives an important indication of the FDG's convergence behavior. That is, if the right hand side (RHS) of Eq. (22) is negative, i.e.,

$$-\frac{\gamma_t}{2}Z_1 + \gamma_t^2 Z_2 < 0 \ \Rightarrow\ \gamma_t < \min\left\{1/L, Z_1/2Z_2\right\},$$

the expected loss $\mathbb{E}_{\boldsymbol{x}_t}[f(\boldsymbol{\theta}^{t+1})]$ would decrease. We give the convergence evidence in the following theorems.

Theorem 2. *Assume Assumptions 1 and 2 hold. The learning rate is non-increasing and satisfies $L\gamma_t \leq 1$. Let $\boldsymbol{\theta}^*$ be the global minimizer and $\mathcal{T}_T = \sum_{t=0}^{T-1}\gamma_t$. Then*

$$\frac{1}{\mathcal{T}_T}\sum_{t=0}^{T-1}\gamma_t\mathbb{E}[\|\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^t\|_2^2] \leq \frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{\beta^K\mathcal{T}_T} + \frac{2Z_2\sum_{t=0}^{T-1}\gamma_t^2}{\beta^K\mathcal{T}_T}. \tag{23}$$

Proof. See supplementary material B. □

Unlike conducting convergence analysis in convex problems, we adopt the *ergodic convergence* (same as [12], [23]) as the metric for convergence analysis. Theorem 2 indicates that, for a randomly selected index $\mathfrak{q}$ from $\{0, 1, \ldots, T-1\}$ with probability $\{\gamma_{\mathfrak{q}}/\mathcal{T}_T\}$, $\mathbb{E}||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{\mathfrak{q}}||_2^2$ is bounded by the RHS of Eq. (23).

Corollary 1. *If the learning rate satisfies 1) non-increasing, 2) $\lim_{T\rightarrow\infty}\sum_{t=0}^{T-1}\gamma_t = \infty$, and 3) $\lim_{T\rightarrow\infty}\sum_{t=0}^{T-1}\gamma_t^2 < \infty$, then the RHS of Eq.* (23) *converges to 0.*

According to Corollary 1, by properly scheduling the learning rate, the expected norm of the gradients could converge to 0, i.e., $\lim_{T\rightarrow\infty}\mathbb{E}||\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^{\mathfrak{q}}||_2^2 = 0$. That is, the proposed method is able to converge to critical points. In addition, the lower bound in Eq. (23) is shown to increase with a larger K. That is, splitting the network into more modules can slow down the convergence. This is consistent with our understanding of the stale gradient effect which increases with more split modules, thereby rendering the network learning more difficult. On the other hand, the convergence can also be achieved by setting an appropriate constant learning rate (see the following theorem).

Theorem 3. *Let Assumptions 1 and 2 hold. Set the learning rate to be a constant:*

$$\gamma = \epsilon\sqrt{(\mathbb{E}[f(\boldsymbol{\theta}^t)] - \mathbb{E}[f(\boldsymbol{\theta}^{t+1})])/TZ_2}$$

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

$$\min_{t\in\{0,1,\ldots,T-1\}}\mathbb{E}[\|\bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^t\|_2^2] \leq \frac{(2+2\epsilon^2)}{\epsilon\beta^K}\sqrt{(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))Z_2/T}. \tag{24}$$

Proof. See supplementary material C. □

As shown in Theorem 3, for $T \rightarrow \infty$, the RHS of Eq. (24) converges to 0. This indicates that expected gradient norm could reach 0, i.e., the FDG converging to critical points.

TABLE III: The validation errors of the compared methods on (a) CIFAR-10, (b) CIFAR-100, and (c) ImageNet for $K = 2$. Results with * are rerun using our training strategy.

Dataset	Architecture	# params	BP [5]	DDG [12]	DGL [20]	FR [13]	FDG
(a) CIFAR-10	ResNet-20	0.27M	8.75%/7.78%*	-	-	-	7.92%(β=1)/**7.23%**(β=0.2)
	ResNet-56	0.46M	6.97%/6.19%*	6.89%/6.63%*	6.77%*	6.07%*	6.20%(β=1)/**5.90%**(β=0.5)
	ResNet-110	1.70M	6.43%/5.79%*	6.59%/6.26%*	6.50%/6.26%*	5.76%*	5.79%(β=1)/**5.73%**(β=0.2)
	ResNet-18	11.2M	6.48%/4.87%*	5.00%*	5.21%*	4.80%*	4.82%(β=1)/**4.79%**(β=0.8)
	ResNet-1202	19.4M	7.93%/5.51%*	-	-	-	5.50%(β=1)/**5.49%**(β=0.5)
	WRN-28-10	36.5M	4.00%/4.01%*	4.05%	4.12%	3.87%	4.13%(β=1)/**3.85%**(β=0.7)

Dataset	Architecture	# params	BP [5]	DDG [12]	DGL [20]	FR [13]	FDG
(b) CIFAR-100	ResNet-56	0.46M	30.21%/27.68%*	29.83%/28.44 %*	29.51%*	28.39%*	27.87%(β=1)/**27.49%**(β=0.4)
	ResNet-110	1.70M	28.10%/25.82%*	28.61%/27.16%*	26.80%*	26.31%*	25.73%(β=1)/**25.43%**(β=0.5)
	ResNet-18	11.2M	22.35%*	22.74%*	22.24%*	22.88%*	22.78%(β=1)/**22.18%**(β=0.5)
	WRN-28-10	36.5M	19.2%/19.6%*	-	-	-	20.28%(β=1)/**19.08%**(β=0.6)

Dataset	Architecture	# params	BP [5]		FDG	
			Top1	Top5	Top1	Top5
(c) ImageNet	ResNet-18	11.2M	29.79%*	10.92%*	29.72%(β=1)/**29.60%**(β=0.3)	**10.42%**(β=1)/10.51%(β=0.3)
	ResNet-50	25.6M	23.65%*	**7.13%***	**23.65%**(β=1)/24.08%(β=0.5)	7.23%(β=1)/7.23%(β=0.5)

V. Experiments

In this section, we conduct experiments on CIFAR-10, CIFAR-100 [24], and ImageNet [25] datasets to compare the generalization and acceleration abilities among different decoupling methods. These experiments show that the proposed FDG performs comparably to or better than the standard BP as well as the state-of-the-art methods.

A. Comparison of Classification Performance

Implementation Details: The experiments are conducted in the PyTorch platform with datasets pre-processed using standard data augmentation (i.e., random cropping, random horizontal flip and normalizing [3]). We use the SGD optimizer with an initial learning rate of 0.1 and a momentum of 0.9. For CIFAR datasets, the weight decay is set to 5×10^{-4} and the models are trained using a batch size of 128 for 300 epochs. The learning rate is divided by 10 at 150, 225 and 275 epochs. For ImageNet dataset, a 224×224 crop is randomly sampled and we train the networks for 90 epochs and decay the learning rate at 30, 60, and 90 epochs. The weight decay is set to 1×10^{-4}. We use $\gamma_t = 0.01$ to warm up the training for 3 epochs. The validation errors of all the experiments are reported at the *last epoch* by the median of 3 runs. In particular, the evaluated networks are initialized with *Kaiming Initialization* [26].

We compare the performances of five methods, including the BP [5], DDG [12], FR [13], DGL [20] and our proposed FDG. The DNI [7] is not included as its performance has been shown to deteriorate severely with deeper networks [12].

CIFAR-10: We begin by reporting the classification results on the CIFAR-10 dataset, which includes 50000 training and 10000 validation samples of 32x32 color images with 10 classes. In this experiment, we split the original network into two modules ($K = 2$). To ensure fair comparisons, we also give the *rerun* results of the compared methods using our training strategy.

The Top1 validation errors are reported in TABLE III-(a). The FR outperforms the DDG consistent with the claim in [13]. It also outperforms the DGL, and even obtains results slightly surpassing the BP baselines. For our proposed method,

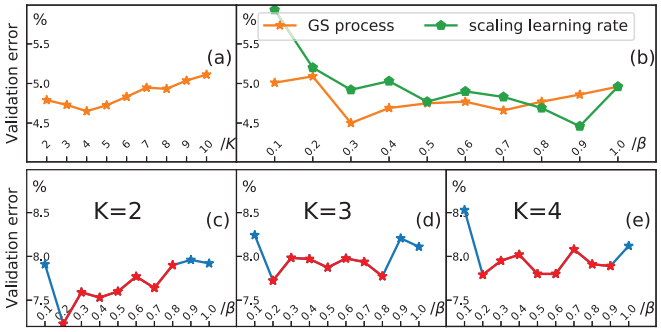

Fig. 3: (a) Performance of ResNet-18 trained on CIFAR-10 for various K with a maximum splittable number $K = 10$. (b) The difference between the GS process and the learning rate scaling (case of ResNet-18, K=3). (c)-(e) Impact of β on the performance conducted on ResNet-20.

TABLE IV: Top 1 errors of ResNet-56 and WRN-28-10 on CIFAR-10 for K=2, 3, 4. "-SK" means K modules.

	BP	DDG	DGL	FR	FDG (β=1/$\beta \neq$ 1)
ResNet-56-S2	6.19%	6.60%*	6.77%*	6.07%*	6.20%/**5.90%**(β=0.5)
ResNet-56-S3	6.19%	6.50%*	8.88%*	6.33%*	6.40%/**6.08%**(β=0.2)
ResNet-56-S4	6.19%	6.61%*	9.65%*	6.48%*	6.83%/**6.14%**(β=0.3)
WRN-28-10-S2	4.01%	4.05%*	4.12%*	3.87%*	4.13%/**3.85%**(β=0.7)
WRN-28-10-S3	**4.01%**	4.12%*	4.91%*	6.16%*	4.19%/4.07%(β=0.5)
WRN-28-10-S4	**4.01%**	6.61%*	5.64%*	5.39%*	6.50%/4.42%(β=0.5)

without the GS process, the FDG overtakes the DDG and DGL, and achieves comparable results with the BP baselines. With a GS process, networks trained by the FDG are able to generalize better than those trained by the BP and FR.

Additionally, to show that the FDG is able to handle networks with various widths and depths, we provide the decoupled training for the smaller network (ResNet-20), wider network (WRN-28-10) and extremely deep network (ResNet-1202). All of these trained networks also generalize better than those trained by the BP.

CIFAR-100: The CIFAR-100 dataset contains the same number of training and validation samples as those in CIFAR-10 but with 100 classes. We again give the *rerun* experiments of the compared methods with our training strategy for fairness.

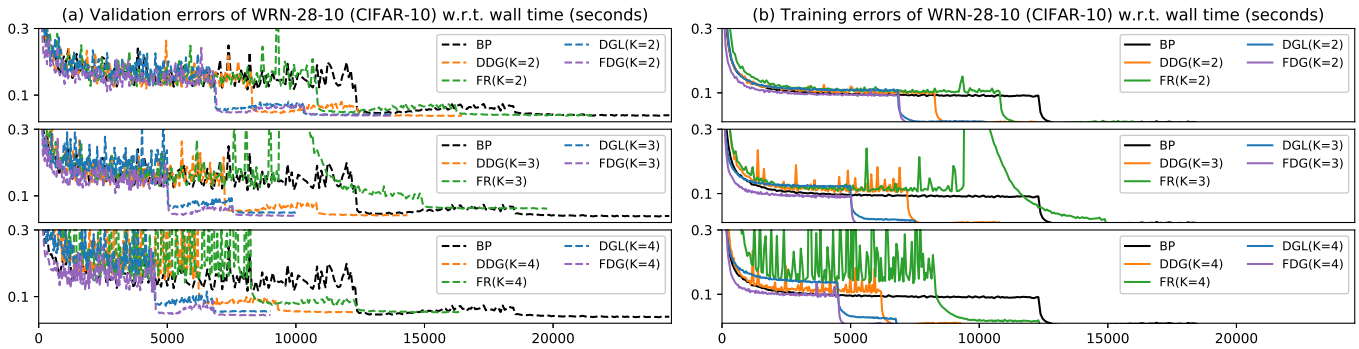

Fig. 4: (a) Validation and (b) training error curves of the compared methods for WRN-28-10 on CIFAR-10 ($K = 2, 3, 4$).

The results are reported in TABLE III-(b). Although overall the FR still overtakes the DDG and DGL, it falls behind the BP. With the GS process, the FDG beats the BP as well as other state-of-the-art methods. In addition, training WRN-28-10 with FDG outperforms the standard BP.

ImageNet: The FDG is also evaluated on the ImageNet 2012 dataset [25] of 1000 classes, which contains 1.28 million training images and 50000 validation images. We train the ResNet-18 as well as ResNet-50 and report their Top1 and Top5 error rates in TABLE III-(c). Similar to the CIFAR cases, the FDG also obtains comparable or better results to BP's. This ImageNet experiment shows that our FDG can handle large-scale datasets.

More Split Modules: In this experiment, we study the performance of ResNet-56 and WRN-28-10 on CIFAR-10 by splitting them into $K = 3$ and $K = 4$ modules with each module trained in an independent GPU. The results are shown in TABLE IV where we list the validation errors with $K = 2, 3, 4$. It becomes noticeable that more split modules have caused all these methods to reduce accuracy. However, we also observe that the GS process allows the FDG to maintain the performance comparable to the BP baseline level. This is an empirical proof that the GS process plays an essential role in reducing the stale gradient effect, which becomes even more significant as K increases. To further reveal the impact of K on the classification error, the ResNet-18 (with $\beta = 0.7$) is trained on CIFAR-10 with various K, and its performance is depicted in Fig 3(a). In particular, $K = 10$ is the maximum splittable number of modules in ResNet-18, with each module being either one layer or one residual block. In general we observe an increase of error rate as K increases, which is reasonable as a large K amplifies the stale gradient effect.

To better appreciate the decoupled learning, we also provide the learning curves of the compared methods w.r.t. wall time (see WRN-28-10 on CIFAR-10 in Fig. 4). It is observed that the fully decoupled methods (DGL and FDG) are much faster than the BU-based methods. We will give detailed and quantitative comparisons in terms of acceleration in the following subsection. On the other hand, the proposed FDG tends to converge more smoothly than the DDG and FR, though delayed gradients are involved in all these methods.

From TABLE III, we observe that although the stale gradient effect exists, the FDG ($K = 2$) tends to outperform the compared methods including the traditional BP. This can be explained as follows. The FDG adopts delayed gradients, which can be treated as real gradients obtained by BP, yet contaminated with noises drawn from an unknown distribution. Although the contaminated gradients could affect the network training, they could however boost the network's generalization with their uncertainties (see [27] for details). Furthermore, the introduced GS process mitigates the stale gradient effect while keeping the gradient uncertainty. Therefore, our FDG can provide more promising performance. However, if K keeps increasing, the stale gradient effect grows and begins to reduce the network's generalization ability to a greater extent than the gain contributed by its uncertainty (see Fig. 3(a)).

The Impact of the GS Process: In TABLE III we have shown that the GS process with a proper β could enhance the FDG's generalization ability. We now give an additional empirical evaluation by setting various values of β. This evaluation is conducted by training the ResNet-20 ($K = 2, 3, 4$) on CIFAR-10 dataset. As shown in Fig. 3(c)-(e), the performance is not very sensitive to the values of β. Furthermore, networks trained with $0.2 \leq \beta \leq 0.8$ (marked red in Fig. 3(c)-(e)) generally outperform those with smaller or larger β (i.e., $\beta < 0.2$ or $\beta > 0.8$) including the case without the GS process (i.e., $\beta = 1$). Intuitively, a small β could reduce the network learning capability due to the rapid decrease of gradient strength in the beginning modules, especially for $K > 2$ as shown in Eq. (17). On the other hand, networks trained with a large β could be more vulnerable to the stale gradient effect. The results in Fig. 3(c)-(e) provide an empirical guideline for choosing a moderate β ranging from 0.2 to 0.8. Within this range, the FDG could consistently outperform its "no-GS" form, which agrees with our results in TABLE III. This is another empirical evidence showing the positive impact of the GS process on a network's ability to generalize.

On the other hand, to distinguish the GS process from the learning rate scaling (see discussion in Section III-B), we train the ResNet-18 ($K = 3$) with various values of β for GS process (i.e., $\boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} \rightarrow \beta^{K-k}\boldsymbol{g}_{\boldsymbol{\theta}_l}^{d_{k,t}}$), and for learning rate scaling (i.e., $\gamma_t \rightarrow \beta^{K-k}\gamma_t$), respectively. As shown in Fig. 3(b), in general, the networks trained involving the GS process slightly outperform those trained with the learning rate scaling. This supports our claim of utilizing the GS process.

TABLE V: Speed comparisons of ResNet-101 among the compared methods on ImageNet. The communication ratio (Cm. ratio) is also given to measure the communication cost in the proposed FDG.

	ResNet-101 ($K=2$) on ImageNet			ResNet-101 ($K=3$) on ImageNet			ResNet-101 ($K=4$) on ImageNet		
	images/s (GPU utilizations)	Id. acc.	**Cm. ratio**	images/s (GPU utilizations)	Id. acc.	**Cm. ratio**	images/s (GPU utilizations)	Id. acc.	**Cm. ratio**
BP	220.5 (98%)	1×	-	220.5 (98%)	1×	-	220.5 (98%)	1×	-
DDG	291.8 (83%,88%), 1.32×	≈1.50×	-	346.9 (56%,63%,51%), 1.57×	≈1.81×	-	369.9 (54%,45%,46%,43%), 1.68×	≈2.01×	-
FR	262.4 (75%,67%), 1.19×	≈1.20×	-	285.0 (65%,57%,69%), 1.29×	≈1.51×	-	320.0 (35%,67%,58%,61%), 1.45×	≈1.72×	-
DGL	401.3 (97%,89%), 1.82×	≈2×	-	576.0 (85%,90%88%), 2.61×	≈3×	-	**748.8 (82%,84%,89%,85%), 3.39**×	≈4×	-
FDG	**421.5 (95%,90%), 1.91**×	2×	0.185%	**594.0 (91%,82%,77%), 2.69**×	3×	0.137%	732.6 (92%,79%,82%,86%), 3.32×	4×	0.450%

Note: To appreciate the speedup, we include the GPU utilization and speedup ratio. "421.5 (95%,90%), 1.91×" indicates 421.5 images/s with utilizations of 95% and 90% in two GPUs, and a 1.91× acceleration compared with the BP. **Cm. ratio**: The time percentage used by communication. **Id. acc.**: Ideal acceleration.

B. Comparison of Acceleration Performance

We conduct experiments by training ResNet-101 on ImageNet (image size of $224\times224\times3$) with various K to compare the acceleration abilities among the decoupling techniques. We choose a relatively deep network as an example because deeper network encourages more balanced workload allocation among different workers. To ensure fair comparisons, we reimplement the DDG, FR and DGL in our framework using the identical communication protocols. In particular, we adopt a pipeline parallelization version for the sequential DGL [20]. This is to encourage a fair comparison of speedup among all the methods by allowing each worker to handle one module alone. The training speed is reported by the number of images per second averaged in 50 iterations after the training is stabilized. We adjust the batch size to maximize the training speed. In addition, the split locations of the network are tuned to allocate the computation workload as even as possible. These experiments are conducted in a server with Tesla V100 GPUs.

Comparisons Among Decoupling Methods: The acceleration results are shown in TABLE V with ideal accelerations included for reference. The ideal accelerations for DDG and FR are calculated based on the empirical evidence [28] suggesting that the backward pass takes about 67% of the whole computation. The ideal acceleration for DGL is similar to that of FDG as the auxiliary network adopted needs negligible computation time. For $K=2,3,4$, the DDG and FR obtain speedups of $1.32\times$, $1.57\times$, $1.68\times$ and $1.19\times$, $1.29\times$, $1.45\times$, respectively. These results are consistent with their ideal accelerations marked by $1.50\times$, $1.81\times$, $2.01\times$ and $1.20\times$, $1.51\times$, $1.72\times$. As shown in TABLE V, the fully decoupled methods provide higher accelerations than those by the BU-based methods. This is because the fully decoupled methods have much higher ideal accelerations, and tend to use the GPU more efficiently (higher GPU utilizations). Overall, the FDG and the DGL achieve comparable speedups but the FDG delivers more promising classification results especially for $K>2$ (see TABLE IV). For $K=2$, the proposed FDG achieves a $\mathbf{1.91}\times$ speedup using 2 GPUs. This is very close to the linear speedup shown in TABLE II, empirically demonstrating the realization of addressing all the lockings with an ideal speedup of $2\times$. For $K=3,4$, the FDG obtains the speedups of $\mathbf{2.69}\times$ and $\mathbf{3.32}\times$ respectively, with the ideal speedups being $3\times$ and $4\times$. In general, the FDG accelerates the learning by an average speedup (defined by speedup/K) of 83%-96% for an additional GPU, which is close to the ideal 100%. This is consistent with our claim of a fully decoupled learning that corresponds to a linear speedup. For demonstration purpose, we also point out that the speedup results are comparable to those from the state-of-the-art data parallelization (around 90%) [9].

Communication Cost: In the proposed FDG, the communication across different workers is essential, yet relatively inexpensive. As reported in TABLE V, the communication only takes 0.45% of the training expense. For instance, in our experiments, the FDG ($K=4$) spends around 175 milliseconds to process a batch of 128 images but only about 0.8 milliseconds are allocated to passing the activations and gradients across different workers. Note that unlike in the data parallelization [9], the communication time here is measured without any overlap with the computation.

It is noted that the average speedup slightly decreases for more GPUs. According to TABLE V, the FDG gives a 96% (1.91/2) average speed gain for a 2nd GPU, but the average speed gain drops to 90% (2.69/3) and 83% (3.32/4) for the 3rd and 4th GPUs. Since we have known that the communication is not the bottleneck, the main cause of this slow-down is likely the uneven distribution of computation in each worker. This can be indicated by the GPU utilizations reported in TABLE V. For $K=3,4$, the GPU utilizations are less balanced among different workers. This suggests that some GPUs are working at nearly full capacity (>90%) while others (<80%) might be idle for some time. Such imbalanced workload is inevitable in the decoupled learning without a custom design to evenly distribute the computation into each worker. The imbalance becomes more observable as K increases. We did not include experiments with $K>5$, which would introduce more imbalance such that the acceleration could be slowed down when more GPUs are utilized.

Ablation study: Here we also provide ablation study (see TABLE VI) to separately demonstrate the effect of each unlocking by training WRN-28-10 ($K=2$) on CIFAR-10 with $\beta=0.7$. We make some adjustments in order to separate a particular unlocking. According to the definition of locking (see Section II), the update unlocking can be detached by forcing the network to update only after all the modules have conducted their forward passes. The forward/backward unlocking can be detached by not performing any asynchronous forward/backward pass among modules. Without any of the lockings (see the 1st row in TABLE VI), the FDG is just the BP algorithm. With merely the backward unlocking (see the 2nd row), the FDG becomes a DDG-like decoupled learning, which runs significantly slower ($1.92\times \rightarrow 1.20\times$) though a slightly better performance is obtained possibly due to less stale gradient effect [12]. The acceleration achieved by having both the backward and forward unlockings but without the update unlocking is remarkable (see the 3rd row), but it

TABLE VI: Ablation study of unlockings for FDG on WRN-28-10.

Backward unlock	Forward unlock	Update unlock	Error	Speedup
×	×	×	4.01%	1×
✓	×	×	**3.77**%	1.20×
✓	✓	×	4.25%	1.91×
✓	✓	✓	3.85%	**1.92**×

results in certain performance deterioration. The FDG with all the three unlockings (see the 4$^{\text{th}}$ row) provides the best acceleration while maintaining good performance.

In summary, we have shown that the proposed FDG can train various networks in fully decoupled manner with no or very limited performance loss. The FDG can also handle large-scale decoupled learning, such as training networks on ImageNet. Through the module-wise model parallelization, the proposed method achieves impressive speed gains (83%-96% average speedup), which are close to the ideal acceleration. We notice that for large K, the acceleration becomes less efficient due to the uneven distribution of computation in each worker. This inefficiency is currently unresolved in decoupled learning and will be mitigated in future work.

VI. Conclusion

In this paper, we have proposed a fully decoupled method using the delayed gradients (FDG) to break the forward, backward, and update lockings for neural network learning. The breaking of these lockings leads to a module-wise parallelization, which enables the network to achieve up to a linear speedup in the ideal case. To enhance the FDG, we further introduce the gradient shrinking process that has been empirically shown to improve the network's ability to generalize, especially when the network is split into more modules. Theoretical analysis has shown that the FDG can converge to critical points of the non-convex optimization problem. We validate the proposed method by training CNNs on CIFAR-10, CIFAR-100 and ImageNet. In these classification tasks, our method outperforms the state-of-the-arts and even overtakes the standard BP while achieving a significant acceleration (e.g., 1.91x with 2 GPUs and 2.69x with 3 GPUs). The FDG also succeeds in training networks with various structures, e.g., extremely deep ResNet-1202 and wide WRN-28-10.

Acknowledgement

This work was partially supported by the Science and Engineering Research Council, Agency of Science, Technology and Research, Singapore, through the National Robotics Program under Grant No. 1922500054.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

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

[4] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2016, pp. 87.1–87.12.

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

[6] D. Balduzzi, H. Vanchinathan, and J. Buhmann, "Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[7] M. Jaderberg, W. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *International Conference on Machine Learning*, 2016.

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

[9] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[10] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2019.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[12] Z. Huo, B. Gu, H. Huang *et al.*, "Decoupled parallel backpropagation with convergence guarantee," in *International Conference on Machine Learning*, 2018, pp. 2103–2111.

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

[14] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, p. 13276, 2016.

[15] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 1037–1045.

[16] D. Xu, A. Clappison, C. Seth, and J. Orchard, "Symmetric predictive estimator for biologically plausible neural learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 9, pp. 4140–4151, 2018.

[17] S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," in *Advances in Neural Information Processing Systems*, 2018, pp. 9368–9378.

[18] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers in neuroscience*, vol. 12, p. 608, 2018.

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

[20] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled greedy learning of cnns," in *International Conference on Machine Learning*. PMLR, 2020, pp. 736–745.

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

[22] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," *Advances in neural information processing systems*, vol. 4, pp. 950–957, 1991.

[23] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[24] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

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

[28] Johnson, "cnn-benchmarks," in *https://github.com/jcjohnson/cnn-benchmarks.git*, 2017.