On the Acceleration of Deep Learning Model Parallelism with Staleness

An Xu[1], Zhouyuan Huo[1], and Heng Huang*[1,2]

[1]Electrical and Computer Engineering Department, University of Pittsburgh, PA, USA
[2]JD Finance America Corporation, Mountain View, CA, USA
{an.xu, zhouyuan.huo, heng.huang}@pitt.edu

Abstract

Training the deep convolutional neural network for computer vision problems is slow and inefficient, especially when it is large and distributed across multiple devices. The inefficiency is caused by the backpropagation algorithm's forward locking, backward locking, and update locking problems. Existing solutions for acceleration either can only handle one locking problem or lead to severe accuracy loss or memory inefficiency. Moreover, none of them consider the straggler problem among devices. In this paper, we propose ***Layer-wise Staleness*** *and a novel efficient training algorithm,* ***Diversely Stale Parameters*** *(DSP), to address these challenges. We also analyze the convergence of DSP with two popular gradient-based methods and prove that both of them are guaranteed to converge to critical points for non-convex problems. Finally, extensive experimental results on training deep learning models demonstrate that our proposed DSP algorithm can achieve significant training speedup with stronger robustness than compared methods.*

1. Introduction

The deep convolutional neural network is an important method for solving computer vision problems such as classification, object detection, etc. However, as the neural networks get deeper and larger [8, 17, 10, 31, 34, 24], the required expensive training time has become the bottleneck. Data parallelism [33, 23, 3] and model parallelism [22, 20] are two standard parallelism techniques to utilize multiple devices for efficient training.

The data parallelism for efficient distributed training has been well studied and implemented in existing libraries [1, 4, 12, 35, 14, 16], but the model parallelism is still underexplored. In this paper, we focus on the model parallelism, where the deep neural network (DNN) benefits from being split onto multiple devices. But the resource utilization of standard model parallelism can be very low. The backpropagation algorithm [29, 21] typically requires two phases to update the model in each training step: the forward pass and backward pass. But the sequential propagation of activation and error gradient leads to *backward locking* and *forward locking* [18] respectively because of the computation dependencies between layers. The *update locking* [18] exists as the backward pass will not start until the forward pass has completed. This sequential execution keeps a device inefficiently waiting for the activation input and error gradient.

Several works have been proposed to address these locking issues (Figure 1). [18] uses Decoupled Neural Interfaces (DNI) to predict the error gradient via auxiliary networks, so that a layer uses the synthetic gradient and needs not to wait for the error gradient. [27] lets hidden layers receive error information directly from the output layer. However, these methods can not converge when dealing with very deep neural networks. [2] proposes layer-wise decoupled greedy learning (DGL), which introduces an auxiliary classifier for each block of layers so that a block updates its parameters according to its own classifier. But the objective function of DGL based on greedy local predictions can be very different from the original model. GPipe [11] proposes pipeline parallelism and divides each mini-batch into micro-batches, which can be regarded as a combination of model parallelism and data parallelism. However, the forward and backward lockings of the micro-batch still exist, and the update locking is not addressed because GPipe waits for the whole forward and backward pass to finish before updating the parameters. [15] proposes Decoupled Parallel Backpropagation (DDG), which divides the DNN into blocks and removes the backward locking by storing delayed error gradient and intermediate activations at each block. But

*Corresponding Author. This work was partially supported by U.S. NSF IIS 1836945, IIS 1836938, IIS 1845666, IIS 1852606, IIS 1838627, IIS 1837956.

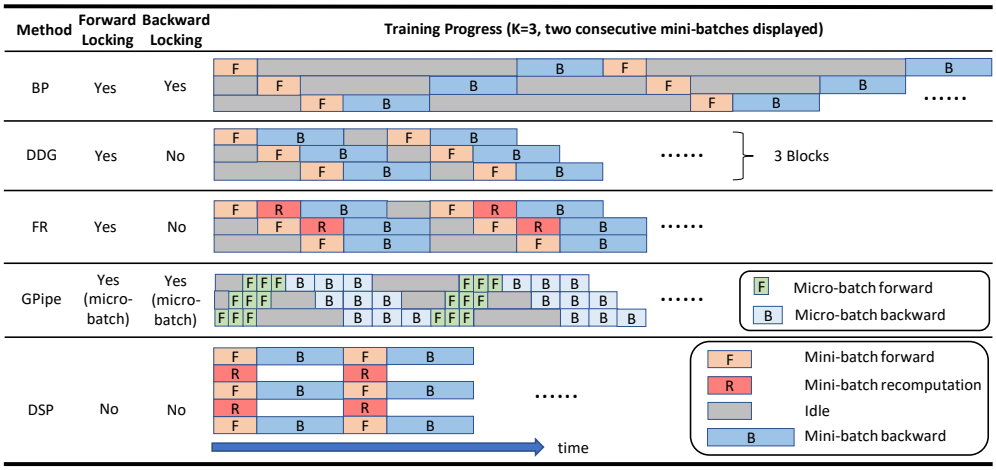

Figure 1. Sketches of different methods with three blocks. The forward and recomputation are overlapped in DSP.

DDG suffers from large memory consumption due to storing all the intermediate results, and cannot converge when the DNN goes further deeper. Features Replay (FR) [13, 36] improves DDG via storing the history inputs and recomputing the intermediate results. Nevertheless, blocks in DDG and FR still need to wait for the backward error gradient. Besides, neither DDG nor FR addresses the forward locking problem.

To overcome the aforementioned drawbacks, we first propose *Layer-wise Staleness*, a fine-grained staleness within the model to allow different parts to be trained independently. Incorporating staleness is useful for efficient asynchronous execution without synchronization barrier [9], which can be interpreted as another form of locking/dependency. The introduction of preset Layer-wise Staleness enables each part of the convolutional neural network (CNN) to run in a very flexible way with a certain degree of asynchrony. Based on the concept of Layer-wise Staleness, we propose a novel parallel CNN training algorithm named as Diversely Stale Parameters (DSP), where lower layers use more stale information to update parameters. DSP also utilizes the recomputation technique [5, 7] to reduce memory consumption, which is overlapped with the forward pass. Our contributions are summarized as follows:

- We propose Layer-wise Staleness and Diversely Stale Parameters (§3) which breaks the forward, backward and update lockings without memory issues.
- To ensure the theoretical guarantee, we provide convergence analysis (§4) for the proposed method. Even faced with parameters of different Layer-wise Staleness, we prove that DSP converges to critical points for non-convex problems with SGD and momentum SGD.
- We evaluate our method via training deep convolutional neural networks (§5). Extensive empirical results show that DSP achieves significant training speedup and strong robustness against random stragglers.

2. Background

We divide a CNN into K consecutive blocks so that the whole parameters $x = (x_0, x_1, ..., x_{K-1}) \in \mathbb{R}^d$, where $x_k \in \mathbb{R}^{d_k}$ denotes the partial parameters at block $k \in \{0, 1, ..., K-1\}$ and $d = \sum_{k=0}^{K-1} d_k$. Each block k computes activation $h_{k+1} = f_k(h_k; x_k)$, where h_k denotes the input of block k. In particular, h_0 is the input data. For simplicity, we define $F(h_0; x_0; x_1; ...; x_k) := f_k(...f_1(f_0(h_0; x_0); x_1)...; x_k) = h_{k+1}$. The loss is $\mathcal{L}(h_K, l)$, where l is the label. Minimizing the loss of a K-block neural network can be represented by the following problem:

$$\min_{x \in \mathbb{R}^d} \quad f(x) := \mathcal{L}(F(h_0; x_0; x_1; ...; x_{K-1}), l). \tag{1}$$

Backpropagation algorithm computes the gradient for block k following chain rule via Eq. (2). The forward locking exists because the input of each block is dependent on the output from the lower block. The backward locking exists because each block cannot compute gradients until having received the error gradient $\mathcal{G}_h$ from the upper block. Besides, the backward process can not start until the whole forward process is completed, which is known as the update locking.

$$\begin{cases} \mathcal{G}_{h_k} = \frac{\partial f_k(h_k; x_k)}{\partial h_k} \mathcal{G}_{h_{k+1}}, & \mathcal{G}_{h_K} = \frac{\partial \mathcal{L}(h_K, l)}{\partial h_K} \\ \mathcal{G}_{x_k} = \frac{\partial f_k(h_k; x_k)}{\partial x_k} \mathcal{G}_{h_{k+1}}. \end{cases} \tag{2}$$

After computing the gradients, stochastic gradient descent (SGD) [28] and its variants such as stochastic unified momentum (SUM) [37], RMSPROP [32] and ADAM [19] are widely used for updating the model. SGD updates

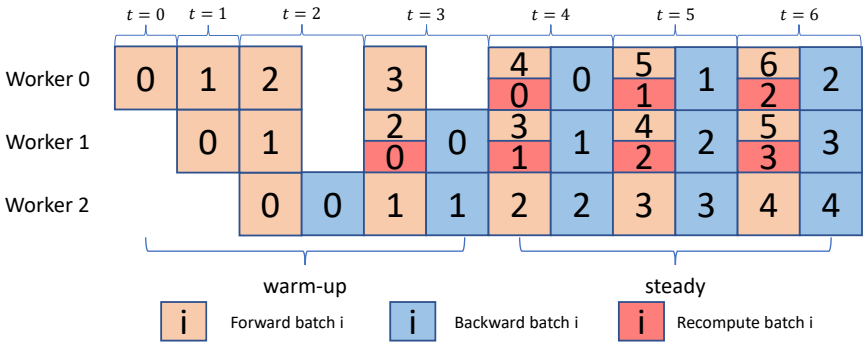

Figure 2. DSP(1,1,0;4,2,0) with Layer-wise Staleness of {4,2,0} (the index difference between the forward and backward batch). Worker $k \in \{0, 1, 2\}$ holds block k.

via $x^{n+1} = x^n - \alpha \mathcal{G}(x^n; \xi)$, where x^n is the parameters when feeding the n^{th} data (batch), α is the learning rate, and $\mathcal{G}(x^n; \xi)$ is the stochastic gradient. SUM updates the parameters via Eq. (3), where β is the momentum constant and y is the momentum term. When $s = 1$, SUM reduces to stochastic Nesterov's accelerated gradient (SNAG) [26].

$$\begin{cases} y^{n+1} = x^n - \alpha \mathcal{G}(x^n; \xi), \ y^{s,n+1} = x^n - s\alpha \mathcal{G}(x^n; \xi) \\ x^{n+1} = y^{n+1} + \beta(y^{s,n+1} - y^{s,n}). \end{cases} \tag{3}$$

3. Diversely Stale Parameters

In this section, we propose a novel training method named Diversely Stale Parameters (Figure 2). We first define layer-wise staleness and related notations in Section 3.1, then the motivation and formulation of DSP gradient in Section 3.2, finally the practical implementation using queues for pipelined batch input in Section 3.3.

3.1. Layer-Wise Staleness

Let the data be forwarded with parameters x_0 at timestamp t_0, x_1 at timestamp t_1, ..., and x_{K-1} at timestamp t_{K-1}. For simplicity we denote the **Forward Parameters** as $\{x_k^{t_k}\}_{k=0,\dots,K-1}$. Similarly we denote the **Backward Parameters** as $\{x_k^{t_{2K-1-k}}\}_{k=0,\dots,K-1}$. Then we define **Layer-wise Staleness** as $\Delta t_k = t_{2K-k-1} - t_k \geq 0$. We preset each block's Layer-wise Staleness to a different value to break the synchronization barrier of backpropagation.

We also denote the maximum Layer-wise Staleness as $\Delta t = \max_{k=0,1,\dots,K-1} \Delta t_k$. It is worth noting that a) in standard backpropagation algorithm (Eq. (2)), Layer-wise Staleness $\Delta t_k = 0$; and b) Feeding data index is not identical to timestamp/training step.

3.2. DSP Gradient

We first set the constraints of DSP as $t_0 < t_1 < \ldots < t_{K-1} \leq t_K < t_{K+1} < \ldots < t_{2K-1}$ such that both the dependencies in the forward and backward pass no longer exist, because we do not need them to finish in the same timestamp anymore. The non-decreasing property corresponds to the fact that the data needs to go through the bottom layers before the top layers, and the error gradient needs to go through the top layers before the bottom layers.

Based on backpropagation algorithm and Eq. (2), we should compute the gradients according to the following formulas as we are updating the Backward Parameters $\{x_k^{t_{2K-1-k}}\}_{k=0,\dots,K-1}$,

$$\begin{aligned} \mathcal{G}_{x_k} &= \frac{\partial F(h_0; x_0^{t_{2K-1}}; \ldots; x_k^{t_{2K-1-k}})}{\partial x_k^{t_{2K-1-k}}} \mathcal{G}_{h_{k+1}} \\ \mathcal{G}_{h_k} &= \frac{\partial F(h_0; x_0^{t_{2K-1}}; \ldots; x_k^{t_{2K-1-k}})}{\partial F(h_0; x_0^{t_{2K-1}}; \ldots; x_{k-1}^{t_{2K-2-k}})} \mathcal{G}_{h_{k+1}} \\ \mathcal{G}_{h_K} &= \frac{\partial \mathcal{L}(F(h_0; x_0^{t_{2K-1}}; \ldots; x_{K-1}^{t_K}), l)}{F(h_0; x_0^{t_{2K-1}}; \ldots; x_{K-1}^{t_K})}. \end{aligned} \tag{4}$$

However, during the forward pass the input of block k is $F(h_0; x_0^{t_0}; \ldots; x_{k-1}^{t_{k-1}})$. Therefore we incorporate the recomputation technique and utilize both the Forward Parameters and Backward Parameters to compute DSP gradient as follows,

$$\begin{aligned} \mathcal{G}_{x_k} &= \frac{\partial F(h_0; x_0^{t_0}; \ldots; x_{k-1}^{t_{k-1}}; \boldsymbol{x_k^{t_{2K-1-k}}})}{\partial \boldsymbol{x_k^{t_{2K-1-k}}}} \mathcal{G}_{h_{k+1}} \\ \mathcal{G}_{h_k} &= \frac{\partial F(h_0; x_0^{t_0}; \ldots; x_{k-1}^{t_{k-1}}; \boldsymbol{x_k^{t_{2K-1-k}}})}{\partial F(h_0; x_0^{t_0}; \ldots; x_{k-1}^{t_{k-1}})} \mathcal{G}_{h_{k+1}} \\ \mathcal{G}_{h_K} &= \frac{\partial \mathcal{L}(F(h_0; x_0^{t_0}; \ldots; x_{K-1}^{t_{K-1}}), l)}{F(h_0; x_0^{t_0}; \ldots; x_{K-1}^{t_{K-1}})}. \end{aligned} \tag{5}$$

The intuition behind the DSP gradient of Eq. (5) is that it is equivalent to Eq. (4) when the model converges to a local optimum where the gradient is zero ($x_k^{t_k} = x_k^{t_{2K-1-k}}$ afterwards).

3.3. Batch Pipeline Input

The computation of the DSP gradient breaks the forward and backward dependencies/lockings of the same data as it will not appear in different blocks at the same timestamp. The update locking is naturally broken.

For the parallel implementation of DSP as shown in Figure 2, we incorporate the data batch pipeline to keep all the blocks being fed with different data batches and running. The data source consecutively feeds data input. Different blocks transport and process different data via FIFO queues. As a result, the data travels each block at different timestamps. Specifically, each block k maintains an input queue $\mathcal{M}_k$, output queue $\mathcal{P}_k$ and gradient queue $\mathcal{Q}_k$ of length $1+m_k$, $1+p_k$ and $1+q_k$ respectively. We denote it as $DSP(p_0, ..., p_{K-1}; m_0, ..., m_{K-1})$. $\{q_k\}$ is determined by $\{p_k\}$ and $\{m_k\}$ because the input should match the corresponding error gradient. We manually split the model to different workers to balance the workload at the steady stage.

Apart from adopting recomputation to reduce memory consumption, DSP overlaps recomputation with the forward pass to save time. Using queues also make DSP overlap the communication between blocks with computation. The FIFO queues allow for some asynchrony which is effective for dealing with random stragglers. The ideal time complexity of DSP is $\mathcal{O}(\frac{T_F+T_B}{K})$ and the space complexity is $\mathcal{O}(L+\sum_{k=0}^{K-1}(m_k+p_k+q_k))$, where T_F and T_B are serial forward and backward time, and L is the number of layers. m_k also represents the Layer-wise Staleness Δt_k of block k. K and the FIFO queues length $m_k+1, p_k+1, q_k+1 \ll L$ for deep models, so the extra space cost is trivial.

4. Convergence Analysis

The convergence of DSP with SGD is first analyzed, then DSP with Momentum SGD. For simplicity, we denote the Forward and Backward Parameters of data n as $x^{n'}$ and x^n respectively.

Assumption 1. ***(Bounded variance)*** *Assume that the DSP stochastic gradient $\mathcal{G}(x;\xi)$ satisfies Var$[\mathcal{G}(x;\xi)] \leq \sigma^2$. Note $\mathbb{E}[\mathcal{G}(x;\xi)] = \mathcal{G}(x) \neq \nabla f(x)$.*

Assumption 2. ***(Lipschitz continuous gradient)*** *Assume that the loss and the output of the blocks have Lipschitz continuous gradient, that is, $\forall k \in \{0, 1, .., K-1\}$, and $\forall (x_{0,1}, ..., x_{k,1}), (x_{0,2}, ..., x_{k,2}) \in \mathbb{R}^{d_0+d_1+...+d_k}$, we have $\|\nabla F(h_0; x_{0,1}; ...; x_{k,1}) - \nabla F(h_0; x_{0,2}; ...; x_{k,2})\| \leq L_k \|(x_{0,1}, ..., x_{k,1}) - (x_{0,2}, ..., x_{k,2})\|$; and $\forall x_1, x_2 \in \mathbb{R}^d$, $\|\nabla f(x_1) - \nabla f(x_2)\| \leq L_K \|x_1 - x_2\|$.*

We define $L := \max_{k\in\{0,1,...,K\}} L_k$. Note that $\nabla F(h_0; x_{0,1}; ...; x_{k,1})$ and $\nabla F(h_0; x_{0,2}; ...; x_{k,2})$ regarding parameters are Jacobian matrices. In fact, this is assuming that the partial model consisted of the blocks that the data has traveled, has Lipschitz continuous gradient.

Assumption 3. ***(Bounded error gradient)*** *Assume that the norm of the error gradient that a block receives is bounded, that is, for any $x \in \mathbb{R}^d$, $\forall k \in \{0, 1, ..., K-2\}$, we have $\left\| \frac{\partial f_{k+1}(h_{k+1}; x_{k+1})}{\partial h_{k+1}} ... \frac{\partial f_{K-1}(h_{K-1}; x_{K-1})}{\partial h_{K-1}} \frac{\partial \mathcal{L}(h_K, l)}{\partial h_K} \right\| \leq M$ and $\left\| \frac{\partial \mathcal{L}(h_K, l)}{\partial h_K} \right\| \leq M$.*

This is assuming that the error gradient at each block does not explode. It is natural to make the above two blockwise assumptions as we are breaking the neural networks into blocks.

Lemma 1. *If Assumptions 2 and 3 hold, the difference between DSP gradient and BP gradient regarding the parameters of block $k \in \{0, 1, ..., K-1\}$ satisfies $\|\nabla_{x_k} \mathcal{L}(F(h_0; x_0^{t_0}; ...; x_{K-1}^{t_{K-1}}), y) - \mathcal{G}_{x_k}(x_0^{t_{2K-1}}; ...; x_{K-1}^{t_K})\| \leq LM \sum_{i=k}^{K-1} \left\| x_i^{t_{2K-1-i}} - x_i^{t_i} \right\|$.*

4.1. DSP with SGD

Theorem 1. *Assume Assumptions 1, 2 and 3 hold. Let $c_0 = M^2K(K+1)^2$, and $c_1 = -(\Delta t^2 + 2) + \sqrt{(\Delta t^2+2)^2 + 2c_0\Delta t^2}$. If the learning rate $\alpha_n \leq \frac{c_1}{Lc_0\Delta t^2}$, then $\frac{\sum_{n=0}^{N-1} \alpha_n \mathbb{E}\left\|\nabla f(x^{n'})\right\|^2}{\sum_{n=0}^{N-1} \alpha_n} \leq \frac{2\left[f(x^0) - f^*\right]}{\sum_{n=0}^{N-1} \alpha_n} + \frac{L\sigma^2(2+K\Delta t^2+\frac{1}{4}Kc_1)\sum_{n=0}^{N-1}\alpha_n^2}{\sum_{n=0}^{N-1}\alpha_n}$.*

Corollary 1. *(Sublinear convergence rate) According to Theorem 1, by setting the learning rate $\alpha_n = \min\left\{\frac{1}{\sqrt{N}}, \frac{c_1}{Lc_0\Delta t^2}\right\}$, when N is large enough we have $\alpha_n = \frac{1}{\sqrt{N}}$ and $\min_{n=0,...,N-1} \mathbb{E}\left\|\nabla f(x^{n'})\right\|^2 \leq \frac{2(f(x^0)-f^*)}{\sqrt{N}} + \frac{L\sigma^2(2+K\Delta t^2+\frac{1}{4}Kc_1)}{\sqrt{N}}$.*

Corollary 2. *According to Theorem 1, if the learning rate α_n diminishes and satisfies the requirements in [28]: $\lim_{N\to\infty}\sum_{n=0}^{N-1}\alpha_n = \infty$ and $\lim_{N\to\infty}\sum_{n=0}^{N-1}\alpha_n^2 < \infty$, choose x^n randomly from $\{x^n\}_{n=0}^{N-1}$ with probabilities proportional to $\{\alpha_n\}_{n=0}^{N-1}$. Then we can prove that it converges to critical points for the non-convex problem due to $\lim_{n\to\infty}\mathbb{E}\left\|\nabla f(x^n)\right\|^2 = 0$.*

4.2. DSP with Momentum SGD

Theorem 2. *Assume Assumption 1, 2 and 3 hold. Let $c_2 = \frac{((1-\beta)s-1)^2}{(1-\beta)^2}$, $c_3 = M^2K(K+1)^2\Delta t^2(c_2+s^2)$, $c_4 = 3+\beta^2c_2+2(1-\beta)^2\Delta t^2(c_2+s^2)$, and $c_5 = \frac{2+\beta^2c_2}{1-\beta} + 2(1-\beta)\Delta t^2(c_2+s^2) + \frac{-c_4+\sqrt{c_4^2+4(1-\beta)^2c_3}}{2(1-\beta)}$. If the fixed learning rate α satisfies $\alpha \leq \frac{-c_4+\sqrt{c_4^2+4(1-\beta)^2c_3}}{2(1-\beta)c_3L}$, then $\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}\left\|\nabla f(x^{n'})\right\|^2 \leq \frac{2(1-\beta)(f(x^0)-f^*)}{N\alpha} + c_5\sigma^2L\alpha$.*

Corollary 3. *(Sublinear convergence rate) According to Theorem 2, by setting the learning rate $\alpha =$*

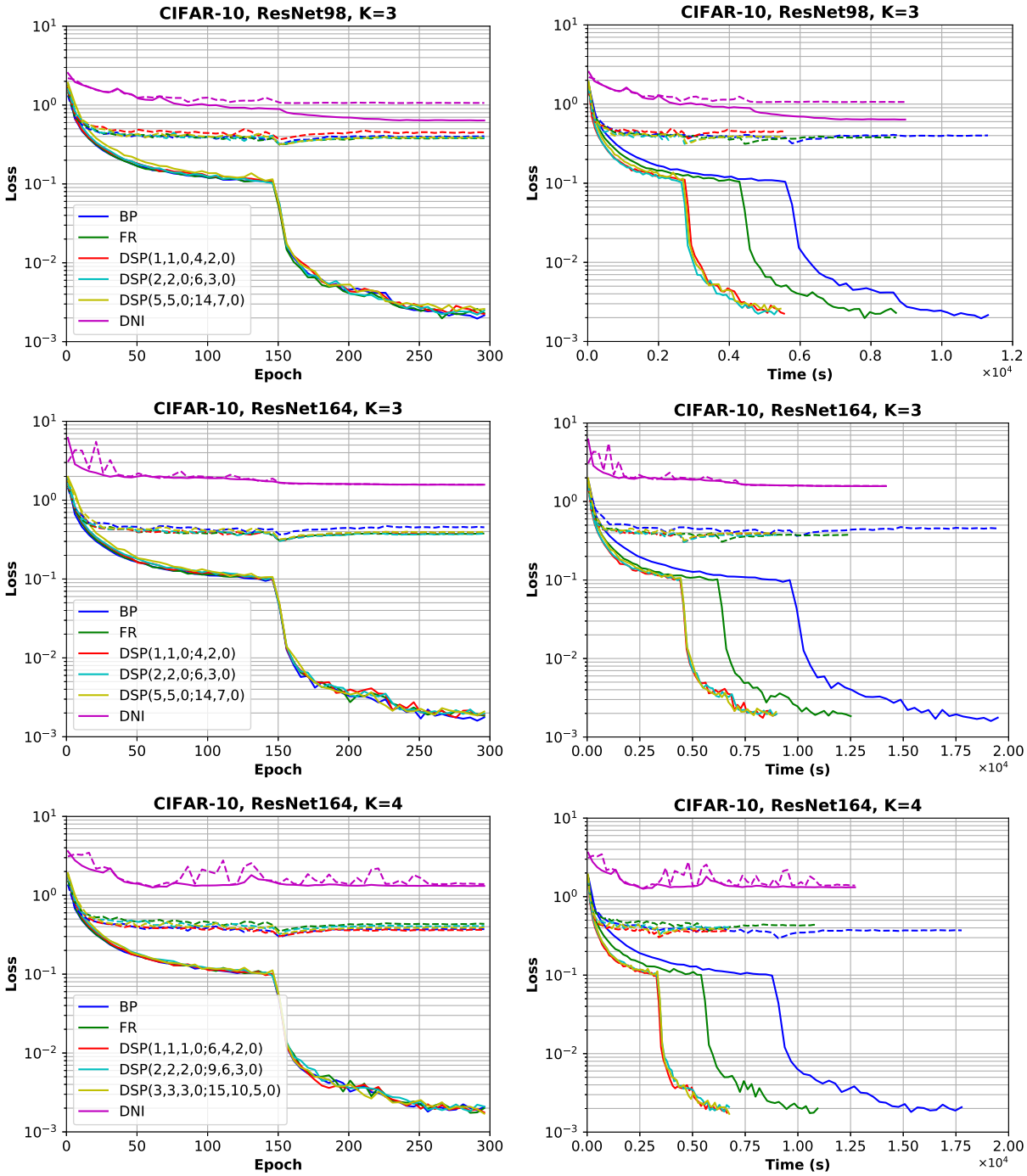

Figure 3. Training loss (solid line) and testing loss (dash line) for ResNet98, ResNet164 on CIFAR-10. The first row and second row plots the loss regarding the training epochs and time respectively.

$\min\{\frac{1}{\sqrt{N}}, \frac{-c_4+\sqrt{c_4^2+4(1-\beta)^2 c_3}}{2(1-\beta)c_3 L}\}$, *when* N *is large enough we have* $\alpha = \frac{1}{\sqrt{N}}$ *and* $\min_{n=0,\ldots,N-1} \mathbb{E}\left\|\nabla f(x^{n'})\right\|^2 \leq \frac{2(1-\beta)(f(x^0)-f^*)}{\sqrt{N}} + \frac{c_5\sigma^2 L}{\sqrt{N}}$.

Remark 1. *The convergence performance of DSP is affected by Layer-wise Staleness rather than the staleness between different blocks.*

Table 1. Best Top-1 Test Accuracy

		ResNet164		ResNet98	
		CIFAR-10	CIFAR-100	CIFAR-10	CIFAR-100
	BP	94.41%	75.66%	93.38%	72.66%
K=3	FR	94.55%	76.25%	93.60%	73.27%
	DSP(1,1,0;4,2,0)	**94.68%**	76.05%	93.36%	72.99%
	DSP(2,2,0;6,3,0)	93.98%	76.00%	**93.68%**	**73.70%**
	DSP(3,3,0;10,5,0)	93.37%	**76.29%**	93.27%	73.38%
K=4	FR	94.44%	75.84%	93.26%	72.41%
	DSP(1,1,1,0;6,4,2,0)	94.32%	**76.22%**	93.41%	**73.14%**
	DSP(2,2,2,0;9,6,3,0)	**94.87%**	75.59%	93.06%	72.89%
	DSP(3,3,3,0;15,10,5,0)	93.34%	75.15%	**93.45%**	72.96%

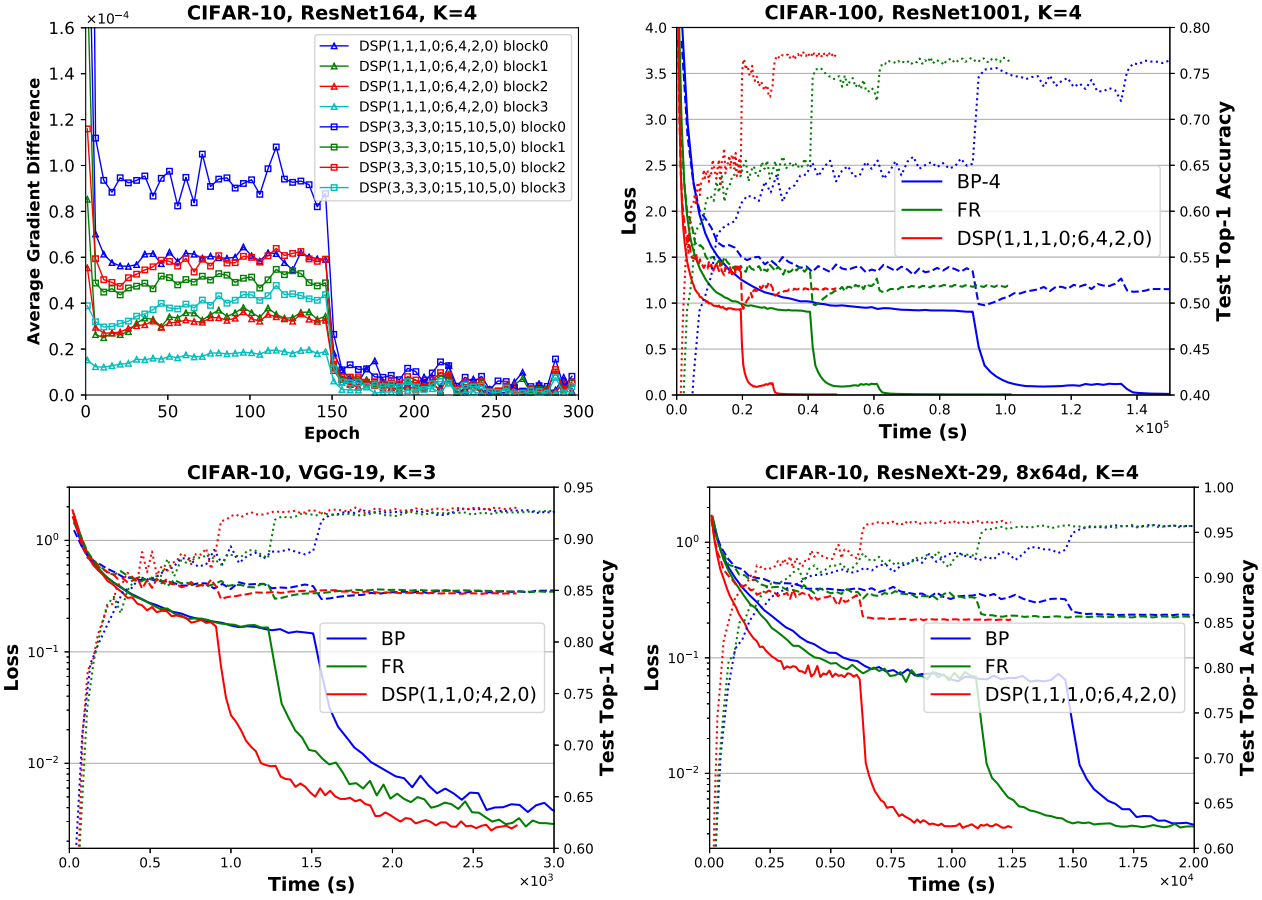

Figure 4. Top left: Average difference of DSP and BP gradient regarding the number of parameters. The rest: Training loss (solid line), testing loss (dash line) and test top-1 accuracy(dot line).

5. Experiments

Experiment Settings We implement DSP in TensorFlow [1] and run the experiments on Nvidia Tesla P40 GPUs. The model is divided into K blocks and distributed onto K GPUs. Data augmentation procedures include random cropping, random flipping, and standardization. We use SGD with the momentum constant of 0.9. In CIFAR experiments, the batch size is 128. We train ResNet98 and ResNet164 for 300 epochs. The weight decay is 5×10^{-4} and the initial learning rate is 0.01 (test performance could be a little lower than 0.1 [25]) with a decay of 0.1 at epoch 150, 225; ResNet1001 is trained for 250 epochs. The weight decay is 2×10^{-4} and the initial learning rate is 0.1 with a decay of

Table 2. Robustness (ResNet164, CIFAR-10, K=3). Each GPU is randomly slowed down.

	Slow down percentage			
GPU	20%	50%	100%	150%
FR	8.977%	28.52%	97.06%	359.2%
DSP(1,1,0;4,2,0)	**6.017%**	16.14%	37.44%	70.99%
DSP(2,2,0;6,3,0)	7.465%	**16.01%**	36.57%	54.57%
DSP(3,3,0;10,5,0)	7.391%	18.15%	**32.10%**	**53.42%**

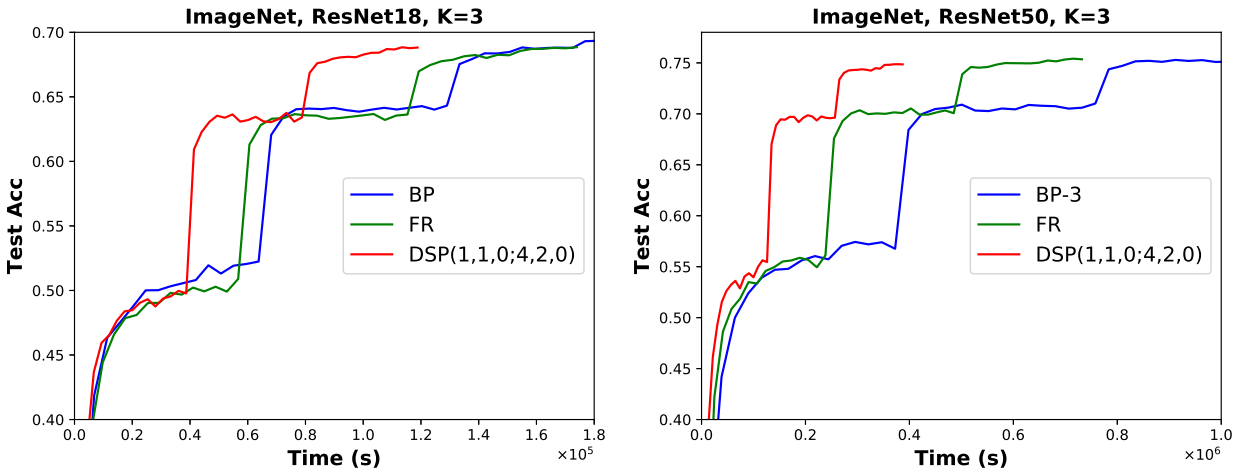

Figure 5. Test accuracy@1 on the ImageNet dataset.

0.1 at epoch 100, 150, 200; VGG-19 and ResNext-29 are trained for 200 epochs. The weight decay is 5×10^{-4} and the initial learning rate is 0.01 with a decay of 0.1 at epoch 100, 150. We also train ResNet on ImageNet for 90 epochs. The batch size is 256, the weight decay is 1×10^{-4} and the initial learning rate is 0.1 with a decay of 0.1 at epoch 30, 60, 80. There are four compared methods:

- BP: The standard implementation in TensorFlow. BP (or BP-K) runs on one (or K) GPUs.
- DNI: The Decoupled Neural Interface algorithm in [18]. The auxiliary network consists of two hidden and one output convolution layers with 5×5 filters and padding size of 2. The hidden layers also use batch-normalization and ReLU.
- FR: The Features Replay algorithm proposed by [13].
- DSP: Our Diversely Stale Parameters.

5.1. Faster Training

The DSP convergence curves regarding training epochs are nearly the same as FR and BP, while DNI does not converge as shown in Figure 3. But the epoch time of DSP is much less. Due to the overlap of communication and computation, the overheads of DSP are much less than model parallel BP and the speedup can even exceed K. However, it is important that the model should be properly distributed onto different blocks such that the workload of each computing device is balanced. If not, the overall speed will be mostly determined by the slowest device. To further demonstrate the scalability of DSP, we also run experiments on VGG-19 [30], ResNeXt-29 [34], ResNet1001 on the CIFAR dataset, and ResNet18 and ResNet50 on the ImageNet [6] dataset as shown in Figure 4 and Figure 5 respectively. The speedup is summarized in Table 3 (GPipe paper only reports speedup of ResNet101 and AmoebaNet-D (4,512)). Our proposed DSP improves the speedup compared with its counterparts from x0.5 to x3.1 based on different datasets, model and the value of K. Note that the implementation of DSP involves some inefficient copy operations due to limited supported features of the deep learning framework, which means that DSP could achieve a potentially even faster speedup.

5.2. Robustness

To show that DSP is more resilient to the straggle problem due to the FIFO queues introduced, we randomly slow down each GPU by a certain percentage with a probability of $1/3$ and run the experiments on ResNet164 (Table 2). The performance of FR degrades a lot because it does not break the forward locking nor completely decouple the backward pass. In comparison, DSP is very robust with the best slow down percentage always less than $1/3$ of the cor-

Table 3. Speedup Comparison Results.

	CIFAR-10			CIFAR-100	ImageNet	
K, batch size	ResNet164 (4, 128)	ResNext-29 (4, 128)	VGG-19 (3, 128)	ResNet1001 (4, 128)	ResNet50 (3, 256)	ResNet101 (4, 128)
BP / BP-K	x1 / -	x1 / -	x1 / -	- / x1	- / x1	x1 / -
FR	x1.7	x1.3	x1.1	x1.9	x1.6	x1.7
GPipe	-	-	-	-	-	x2.2
DSP	**x2.7**	**x2.4**	**x1.5**	**x4.8**	**x3.0**	**x2.7**

Table 4. Best Top-1 Test Accuracy on ImageNet (K=3).

Method	ResNet18	ResNet50
BP	69.89%	75.35%
FR	68.94%	74.47%
DSP(1,1,0;4,2,0)	68.95%	74.91%

responding GPU slow down percentage. When the upper or lower block suddenly slows down, the current block's feeding data and gradient queues are less likely to be empty if the length of the queue is long. When the straggler effect is not serious, increasing the Layer-wise Staleness will not bring performance gain; when it is serious instead, DSP benefits a lot from increasing the Layer-wise Staleness. Generally speaking, longer queues improve DSP's resilience to random stragglers, which is shown in Table 2.

5.3. Generalization

Table 1 and Tabel 4 show the best top-1 test accuracy on the CIFAR and ImageNet dataset respectively. The test performance of DSP is better than BP and FR on the CIFAR dataset. From Lemma 1 we know that the DSP gradient deviates from the BP gradient due to the Layer-wise Staleness. This difference becomes small as the training proceeds but could impose small noise and help find a better local minimum on the comparatively less complex CIFAR classification problem.

In comparison, on the ImageNet dataset, the Layer-wise Staleness can lead to performance degradation. By intuition, it is similar to asynchronous distributed training where the whole gradient is of the same staleness. But in DSP, the more fine-grained Layer-wise Staleness will impose different blocks with different staleness effects. Potential solutions could be using staleness-aware methods as proposed in asynchronous distributed training area, e.g. gradient compensation and staleness-aware learning rate, to alleviate the staleness effect. Another possible direction is to balance the staleness effect between all the blocks. Moreover, when compared with FR, DSP's test accuracy is slightly better. On ResNet18, the test accuracy of FR and DSP is very similar, but on ResNet50 there is a 0.44% gain using DSP. Besides, on the more complicated ResNet50 architecture, the performance degradation resulting from the staleness effect is smaller than that on ResNet18.

5.4. Gradient Difference

Here we attest our theoretical analysis of Lemma 1 via checking the difference between the DSP and the BP gradient on the CIFAR dataset with the ResNet164 model. From the top-left figure of Figure 4, we can see that the difference between the DSP and BP gradient drops very fast to the converged value as the training proceeds. This difference drops even faster for upper blocks where the Layer-wise Staleness effect is milder. It confirms the motivation behind the DSP algorithm that the DSP gradient will finally be similar to the BP gradient. Moreover, the lower blocks suffer from a larger difference. When the Layer-wise Staleness keeps increasing, the difference will also increase, which matches Lemma 1 well. Moreover, as the learning rate drops, the difference between the DSP gradient and the BP gradient will drop a lot. This implies that a smaller learning rate should be used when we need to deal with a larger number of blocks where the Layer-wise Staleness effect becomes non-trivial. This is also shown in Theorem 1 and 2 that the learning rate should be decreased to make sure it converges at the stated speed.

6. Conclusion

In this paper, we have proposed Layer-wise Staleness and DSP, a novel way to fast train neural networks. DSP is proved to converge to critical points for non-convex problems with SGD and Momentum SGD optimizer. We apply DSP to train CNNs in parallel and the experiment results confirm our theoretical analysis. Our proposed method achieves significant training speedup, strong resilience to random stragglers, better generalization on the CIFAR dataset and reasonable performance on the ImageNet dataset. The speedup can exceed K compared with the model parallel BP. Potential future works include how to alleviate the staleness effect when we need to utilize a further larger number of blocks; how to automatically determine the proper model splitting strategy for load balance among devices; efficiently incorporating DSP with data parallelism to achieve even faster training speed.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 16)*, pages 265–283, 2016.

[2] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. *arXiv preprint arXiv:1901.08164*, 2019.

[3] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD.

[4] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Andreas Griewank. An implementation of checkpointing for the reverse or adjoint model of differentiation. *ACM Trans. Math. Software*, 26(1):1–19, 1999.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[9] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pages 1223–1231, 2013.

[10] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[11] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

[12] Yuzhen Huang, Xiao Yan, Guanxian Jiang, Tatiana Jin, James Cheng, An Xu, Zhanhao Liu, and Shuo Tu. Tangram: bridging immutable and mutable abstractions for distributed data analytics. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 191–206, 2019.

[13] Zhouyuan Huo, Bin Gu, and Heng Huang. Training neural networks using features replay. In *Advances in Neural Information Processing Systems*, pages 6659–6668, 2018.

[14] Zhouyuan Huo, Bin Gu, and Heng Huang. Large batch training does not need warmup. *arXiv preprint arXiv:2002.01576*, 2020.

[15] Zhouyuan Huo, Bin Gu, qian Yang, and Heng Huang. Decoupled parallel backpropagation with convergence guarantee. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2098–2106, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[16] Zhouyuan Huo and Heng Huang. Straggler-agnostic and communication-efficient distributed primal-dual algorithm for high-dimensional data mining. *arXiv preprint arXiv:1910.04235*, 2019.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[18] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1627–1635. JMLR. org, 2017.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[22] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. On model parallelization and scheduling strategies for distributed machine learning. In *Advances in neural information processing systems*, pages 2834–2842, 2014.

[23] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[24] Yuejiang Liu, An Xu, and Zichong Chen. Map-based deep imitation learning for obstacle avoidance. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8644–8649. IEEE, 2018.

[25] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[26] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[27] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045, 2016.

[28] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[29] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[32] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[33] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[34] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017.

[35] An Xu, Zhouyuan Huo, and Heng Huang. Optimal gradient quantization condition for communication-efficient distributed training. *arXiv preprint arXiv:2002.11082*, 2020.

[36] Qian Yang, Zhouyuan Huo, Wenlin Wang, and Lawrence Carin. Ouroboros: On accelerating training of transformer-based language models. In *Advances in Neural Information Processing Systems 32*, pages 5519–5529. Curran Associates, Inc., 2019.

[37] Tianbao Yang, Qihang Lin, and Zhe Li. Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization. *arXiv preprint arXiv:1604.03257*, 2016.