Not All Layers Are Equal: A Layer-Wise Adaptive Approach Toward Large-Scale DNN Training

Yunyong Ko
Hanyang University
Seoul, Republic of Korea
koyunyong@hanyang.ac.kr

Dongwon Lee
The Pennsylvania State University
University Park, PA, USA
dongwon@psu.edu

Sang-Wook Kim*
Hanyang University
Seoul, Republic of Korea
wook@hanyang.ac.kr

ABSTRACT

A large-batch training with data parallelism is a widely adopted approach to efficiently train a large deep neural network (DNN) model. Large-batch training, however, often suffers from the problem of the model quality degradation because of its fewer iterations. To alleviate this problem, in general, *learning rate* (*lr*) scaling methods have been applied, which increases the learning rate to make an update larger at each iteration. Unfortunately, however, we observe that large-batch training with state-of-the-art *lr* scaling methods still often degrade the model quality when a batch size crosses a specific limit, rendering such *lr* methods less useful. To this phenomenon, we hypothesize that existing *lr* scaling methods overlook the subtle but important differences across "layers" in training, which results in the degradation of the overall model quality. From this hypothesis, we propose a novel approach (**LENA**) toward the learning rate scaling for large-scale DNN training, employing: (1) a layer-wise adaptive *lr* scaling to adjust *lr* for each layer individually, and (2) a layer-wise state-aware warm-up to track the state of the training for each layer and finish its warm-up *automatically*. The comprehensive evaluation with variations of batch sizes demonstrates that LENA achieves the target accuracy (i.e., the accuracy of single-worker training): (1) within the fewest iterations across different batch sizes (up to 45.2% fewer iterations and 44.7% shorter time than the existing state-of-the-art method), and (2) for training very large-batch sizes, surpassing the limits of all baselines.

CCS CONCEPTS

• **Theory of computation** → **Distributed algorithms**.

KEYWORDS

large batch training, learning rate scaling, layer-wise approach

ACM Reference Format:
Yunyong Ko, Dongwon Lee, and Sang-Wook Kim. 2022. Not All Layers Are Equal: A Layer-Wise Adaptive Approach Toward Large-Scale DNN Training. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3511989

*Corresponding author

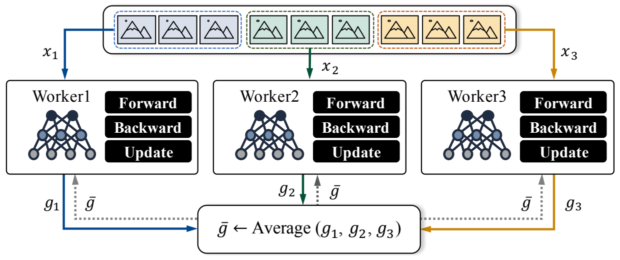


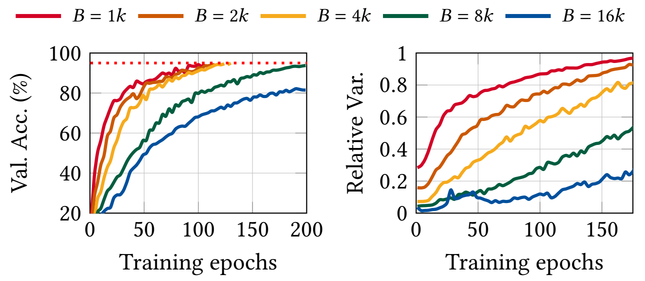
Figure 1: The architecture of a large-batch training with data parallelism.

1 INTRODUCTION

With the recent breakthroughs in deep learning (DL) [5, 6, 11, 19, 27, 34, 36], many web-based applications have adopted DL techniques to improve the quality of their services. In order for web-based applications to continuously provide high-quality services, it is important to efficiently learn a large amount of "web-scale" data generated from users [31, 39]. Training a large deep neural network (DNN) model with web-scale data, however, is very time-consuming, often taking several days or weeks despite using computational accelerators such as GPUs [5, 11, 36]. To speed up such training, a large-batch training with data parallelism has been widely adopted [1, 2, 7, 8, 12, 14, 23, 32, 37, 38, 44–46]. Figure 1 shows the process of a large-batch training with data parallelism, where a large amount of data (i.e., large batch) is split and processed by multiple workers at once. Then, the gradients computed by workers are averaged and used to update a DNN model. Thus, as the number of workers increases, the amount of data processed at each iteration increases proportionally, thereby reducing the total number of iterations (and time) in training.

However, if the reduction of the iterations is too severe for a model to converge, it is likely to degrade the quality of the learned model [12, 17, 28]. This problem may become more serious as the batch size increases and the number of iterations decreases, thereby having fewer chances to update gradients toward convergence. As such, it should be addressed to fully exploit the gain of large-batch training. To compensate for fewer iterations in large-batch training, in general, learning rate (*lr*) scaling methods have been applied [12, 20, 41–43]. *lr* scaling increases the *learning rate* to boost each gradient update to be larger. For example, the linear *lr* scaling method [12], the most widely used one, increases *lr* linearly as the batch size increases. Although the linear scaling method was successfully applied in several tasks (e.g., 8k of a batch size in ResNet-50 training on ImageNet-1k), it often causes significant degradation of the model quality or even divergence in larger scales [12, 40, 43].

On the other hand, some prior works [20, 21, 29, 40] take into account the *variance* of gradients in large-batch training. In [29, 40],



(a) Model accuracy. **(b) Relative variance.**

Figure 2: The relationship between the model quality and gradient variance in large-batch training with AdaScale [20] (B indicates the total batch size).

it is shown that the efficiency of scaling large-batch training improves when the variance of gradients is higher, which indicates that the variance of gradients can be an important factor to be considered for reliably increasing the scale of large-batch training. In particular, AdaScale [20], a state-of-the-art *lr* scaling method, adaptively adjusts *lr* depending on the variance of gradients computed by workers at each iteration. For instance, it increases *lr* almost linearly under high gradient variance, but rarely increases *lr* under low gradient variance. In this way, AdaScale successfully achieved the high quality of models across many large-batch sizes in various machine learning tasks, outperforming the linear scaling method. Despite its success, however, our preliminary experiment showed that the quality of the learned model by AdaScale still degraded when the batch size crossed a specific limit (e.g., 16k of the batch size in the training of ResNet-18 on CIFAR-10, see Figure 2(a)).

Motivated from this phenomenon, we take one step further for understanding the meaning of the gradient variance in large-batch training. Generally, the gradient variance tends to increase as the model gets trained (see Figure 2(b)), which indicates that the gradients computed from different batches of data are becoming more diverse from each other. This tendency implies that the information that the model aims to train from each data batch becomes more unique as the model gets trained better. Therefore, we can estimate how well the current model has been trained so far (i.e., *state of the training*) by using the variance of gradients at each iteration. Figure 2 shows the relationship between the model quality and the gradient variance for training with different large-batch sizes.

On the other hand, the parameters of a DNN model have their own role and the information to capture from data is not equal across layers [3, 4]. Therefore, we hypothesize that "during the training of a DNN model, the state of the training can vary across layers". To verify our hypothesis, we conducted another preliminary experiment for comparing the gradient variances of parameters per layer, and observed that *the gradient variance tends to vary across layers and such a difference grows as the batch size gets larger* (see Figure 3 for details). Based on this observation, we posit that the potential cause of the model quality degradation in existing *lr* scaling methods is to overlook the subtle but important differences across layers in training. Then, for addressing this issue, we propose a novel approach toward *lr* scaling for large-scale DNN training, named as **L**ayer-wise adaptiv**E** learning rate scali**N**g and w**A**rm-up (**LENA**), that employs (1) a layer-wise adaptive *lr* scaling that adjusts *lr* for each layer individually, depending on its gradient variance, and (2) a layer-wise state-aware warm-up that tracks the training state for each layer and determines its endpoint of warm-up *automatically*.

Through comprehensive experiments, we demonstrate that LENA always achieves the target accuracy (i.e., the accuracy of single-worker training): (1) within the fewest iterations across variations of batch sizes (up to 45.2% fewer iterations and 44.7% shorter time than the existing state-of-the-art gradient variance-based method (AdaScale)), and (2) for training with very large-batch sizes, surpassing the limits of existing state-of-the-art methods. To the best of our knowledge, this is the first work to successfully scale up the large-batch training to these limits. We also verify that each of our layer-wise strategies significantly improves the model quality in large-batch training. The main contributions of this work are as follows:

- Identifying the cause of the degradation in the model quality in existing *lr* scaling methods – i.e., overlooking the difference across layers in training of a DNN model.
- Proposing a novel approach to *lr* scaling for large-batch training, LENA, by employing a layer-wise adaptive *lr* scaling and state-aware warm-up to effectively address the difference across layers.
- Comprehensive evaluation validating the effectiveness of LENA in large-batch training, successfully achieving the high quality of models within fewest iterations in training, with very large-batch sizes up to ×128 larger than the case of single-worker training.

2 RELATED WORK

2.1 Large-Batch Training with SGD

In this work, we consider the following problem:

$$\min_{w\in\mathbb{R}^d} F(w) = \frac{1}{|X|}\sum_{x\in X} f(w,x), \tag{1}$$

where w is the set of model parameters, X is the training dataset, and $f(w,x)$ is the loss function of the parameters w given data samples x. To solve the problem represented in Eq 1, we consider SGD as an optimization algorithm. Let w_t be the model parameters at iteration t. Then, w_t is updated iteratively by the following rule:

$$w_{t+1} = w_t - \eta \cdot g_t, \quad g_t = \frac{1}{|B|}\sum_{x\in B} \nabla f(w_t,x), \tag{2}$$

where g_t is the gradient computed at iteration t, B is a batch sampled from X, and η is the learning rate.

As the sizes of models [5, 11, 34, 36] and datasets [9, 26, 30, 33] increase, this iterative process requires more time. To accelerate this process, therefore, a large-batch training with data parallelism increases the total amount of data, B, processed at each iteration by having multiple workers process them in parallel as illustrated in Figure 1. In the case of training with n workers, each worker i samples a batch b from X and computes its gradient $g_t^{(i)} = \frac{1}{|b|}\sum_{x\in b}\nabla f(w_t,x)$, and then the gradients computed by n workers are averaged, $\bar{g} = \frac{1}{n}\sum_{i=1}^{n} g_t^{(i)}$. Finally, the model w_t is updated by applying the averaged gradient, $w_{t+1} = w_t - \eta \cdot \bar{g}_t$. Therefore, as the number of workers increases, the larger amount

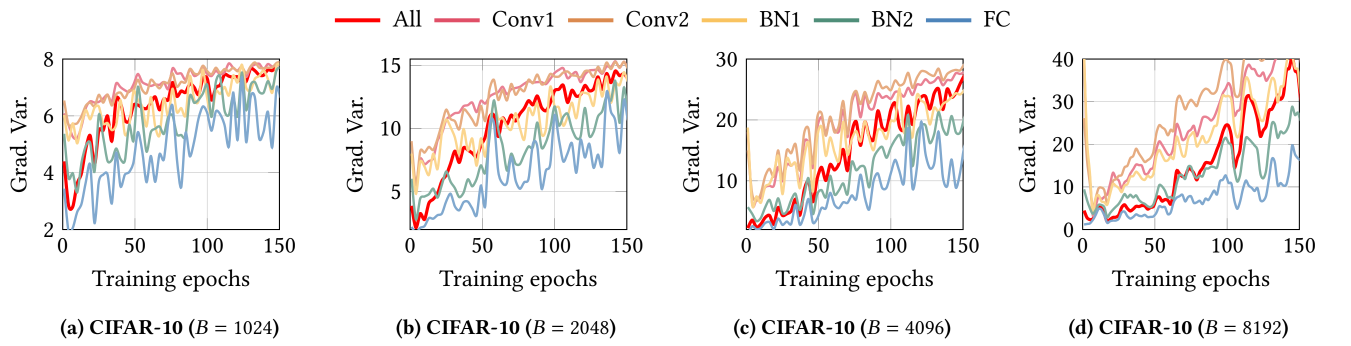


Figure 3: Observation: the gradient variance tends to vary across layers and such a difference grows as the batch size gets larger.

of data ($B = n \cdot b$) is processed at each iteration and the fewer number of iterations is required in training ($T_n = T_1/n$, where T_1 the number of iterations in the case of single-worker training).

2.2 Learning Rate Scaling

However, if the number of iterations in training is reduced too excessively (thus, too small number of iterations for a model to converge), it may lead to degradation of the quality of the trained model Goyal et al. [12], Jain et al. [17], Ma et al. [28]. To compensate for this shortage of iterations in large-batch training, therefore, *learning rate scaling* methods have been proposed [12, 16, 20, 35, 41–43], which increases the *learning rate* (*lr*) to make each update larger. In a large-batch training, the goal of *lr* scaling is to train a model with the quality, as good as the quality of single-worker training (i.e., target accuracy), with fewer iterations (thus faster). Formally, given the model w, dataset X, and n workers, let $F(w,x)^{(1)}$ and $F(w,x)^{(n)}$ be the results of single-worker training and multi-worker (i.e., large-batch) training, respectively. Then, the goal of *lr* scaling is defined as follows.

$$F(w,x)^{(1)} \approx F(w,x)^{(k)} \tag{3}$$

Fixed *lr* scaling methods increase *lr* as the batch size increases for training [10, 12, 18, 24, 25, 41–43]. The square root *lr* scaling multiplies *lr* by the square root of the increase of the batch size [15, 25]. For example, when the batch size is multiplied by 4, it multiplies *lr* by 2. The linear scaling [12], the most widely used one, multiplies *lr* by k when the batch size increases by k times. The model trained with fixed scaling methods, however, tends to fluctuate steeply in the early stage of training. To alleviate this problem, *lr* warm-up is often applied [12]. It gradually increases *lr* from a small value such that *lr* reaches the target value at a specific epoch (e.g., the first 5.5% of training epochs). The fixed scaling with the warm-up method was successfully applied in several tasks; however, it often causes significant degradation of the model quality [12, 20].

To alleviate the problem of fixed scaling, then, adaptive scaling methods were proposed to adaptively adjust *lr* as the training progresses [41–43]. You et al. [41, 43] adjust *lr* based on the ratio of the parameter value and the gradients computed at each iteration. You et al. [42], that was proposed for the large-batch training for LSTM models, increases the period of warm-up in proportion to the batch size. Some works [18, 20, 21, 29, 40] consider the gradient variance for *lr* scaling in large-batch training. In [29, 40], it is shown that the gradient variance is a crucial factor to consider for reliably scaling large-batch training. AdaScale [20], a state-of-the-art *lr* scaling method, estimates the state of the training by using gradient variance and adaptively adjusts *lr* depending on the variance of gradients computed by workers at each iteration (i.e., called the gain ratio). For instance, it increases *lr* almost linearly in case of high gradient variance, but rarely increases *lr* in case of low gradient variance. In this way, AdaScale successfully achieved the higher accuracies than those of the linear scaling method in various machine learning tasks [20].

3 THE PROPOSED METHOD: LENA

In this section, we first introduce our observations on the important feature in the training of DNN models, and point out the limitations of existing *lr* scaling methods based on our observation. To overcome the limitations, then, we propose a novel *lr* scaling approach for large-scale DNN training, named as **L**ayer-wise adaptiv**E** learning rate scali**N**g and w**A**rm-up (**LENA**).

3.1 Layer-Wise *lr* Scaling

As we explained in Section 1, inspired by [3, 4], we hypothesize that "during the training of a DNN model, the state of the training can vary across layers." To verify our hypothesis, we trained the ResNet-18 model [13] with four different batch sizes on the CIFAR-10 dataset [26] and measured the gradient variance of parameters per layer. Figure 3 shows the results where the x-axis represents the training epoch and the y-axis represents the gradient variance. We observe that *the gradient variance of parameters tends to vary across layers, and such differences get larger as the batch size increases.* This observation indicates that the status of the training of parameters could be different depending on layers and their differences get larger in larger-batch training.

Based on this observation, we closely look into the existing state-of-the-art gradient variance-based method (i.e., AdaScale) [20]. The existing method adjusts *lr* based on the gradient variance of parameters for *the entire model* and applies the adjusted *lr* to *all* model parameters. In other words, it does not consider the difference across layers in training. As clearly shown in Figure 3, however, the gradient variance for the entire model cannot faithfully represent that for each layer, which can cause the following limitations. For layers with lower gradient variance than that of the entire model, *lr* is likely to be adjusted to be larger (than it should be). Symmetrically, for layers with higher gradient variance than that of the entire model, *lr* is likely to be adjusted to be smaller (than it should

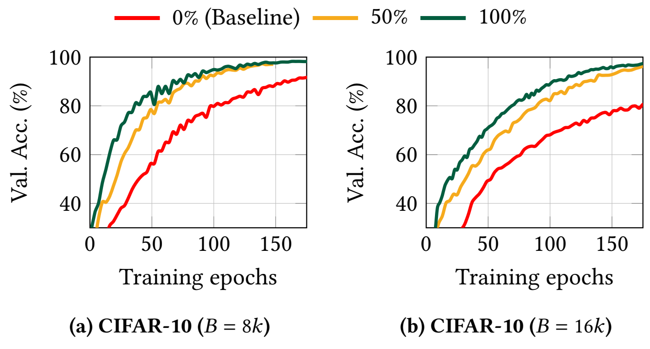


Figure 4: The potential of layer-wise adaptive *lr* scaling for improving the model quality in large-batch training.

be) Such limitations of the existing *lr* scaling method can adversely affect the model quality and become an obstacle to further scaling large-batch training.

Therefore, to address these limitations, we propose a novel layer-wise adaptive *lr* scaling that considers the difference across layers in training. The proposed *lr* scaling computes the gradient variance of each layer *individually* and adjusts *lr* for each layer based on the computed gradient variance of every layer at every iteration. Thus, each layer has it own *lr* at every iteration. Inspired by [20, 40], we define the gradient variance of layer k at iteration t as

$$grad\_var_{(t,k)} = \frac{\frac{1}{n}\sum_{i=0}^{n-1} \|g_{(t,k)}^{(i)}\|^2}{\|\bar{g}_{(t,k)}\|^2}, \quad (4)$$

where $g_{(t,k)}^{(i)}$ is the local gradient for layer k computed by worker i at iteration t, and $\bar{g}_{(t,k)}$ is the average of all local gradients for layer k at iteration t. Then, given the base learning rate η_0, which is a user-defined hyperparameter, we adjust the learning rate for the parameters of layer k at iteration t by $grad\_var_{(t,k)} \in [1, n]$,

$$\eta_{(t,k)} = grad\_var_{(t,k)} \cdot \eta_0. \quad (5)$$

Thus, for each layer, the larger gradient variance is (i.e., $grad\_var_{(t,k)}$ closes to n), the learning rate is adjusted to be larger (i.e., closing to linear scaling), but the smaller gradient variance is, the learning rate is adjusted to be smaller.

Now, let us demonstrate the potential of our layer-wise *lr* scaling. We apply the layer-wise *lr* scaling to randomly select 50% and 100% of layers, train ResNet-18 with very large batch sizes (8k and 16k) on CIFAR-10 using two versions, and compare their model accuracies with the accuracy of the baseline (i.e., AdaScale). Figure 4 shows the results where the x-axis represents the training epoch and the y-axis represents the model accuracy. As clearly demonstrated in Figure 4, the more layers our layer-wise *lr* scaling idea is applied to, the higher the model accuracy becomes. This result indicates that our layer-wise *lr* scaling successfully addresses the limitations of the existing *lr* scaling method and has a potential for improving the model quality in large-batch training. From this result, we obtain the two important lessons: (1) considering the training differences across layers is a critical factor for improving the model quality in large-batch training and (2) the existing *lr* scaling methods ignore the subtle but important difference across layers in training, thus suffering from the degradation of the model quality.

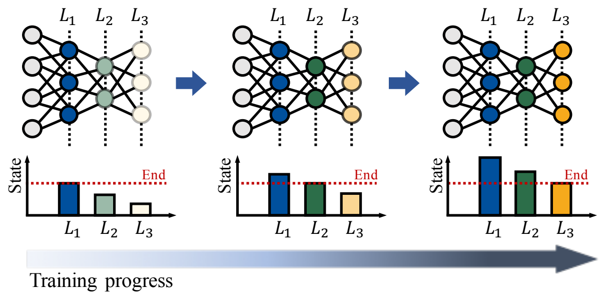


Figure 5: The process of our layer-wise state-aware warm-up (The warm-up for each layer ends at a different point based on its training state).

3.2 Layer-Wise State-Aware Warm-Up

As we observed in Section 3.1, the state of the training is not uniform across layers, each of which is a building block of a DNN model with its own role. Thus, it implies that the training speed also can vary across layers. From this understanding, we posit that in large-batch training, *the optimal endpoint of the warm-up may differ across layers*. The *warm-up* technique [12, 42] starts from a small *lr* in the early stage of training where a model tends to change rapidly, and gradually increases *lr* until the model becomes stable. However, since the training speed can differ across layers, the point at which each layer becomes stable can be also different across layers (i.e., the different endpoint of the warm-up). For example, the layers with fast training speeds do not need a long period of warm-up, while the layers with slow training speeds would benefit from a longer period of warm-up.

Despite the different training speeds across layers, however, the existing warm-up method does not consider such a difference and simply applies the same warm-up period to all parameters in a DNN model [12, 42]. That is, the endpoint of warm-up is uniform across the entire model, possibly leading to the following defects. For the layers with fast training speeds (i.e., layers with high gradient variance), the warm-up is likely to be applied unnecessarily long, delaying the model convergence. On the other hand, for the layers with slow training speeds (i.e., layers with low gradient variance), the warm-up is likely to end too early, adversely affecting the model quality. In addition, the endpoint of warm-up is often a user-defined hyperparameter, requiring much trial-and-error tuning to find a proper point.

In order to address these limitations, therefore, we propose a *layer-wise state-aware warm-up* that tracks the state of the training for each layer individually and determines its endpoint of the warm-up automatically. Figure 5 illustrates how our idea could adaptably change the warm-up across layers. In the training with our warm-up, for the layers with fast training speeds, the warm-up ends early since their training states quickly become stable. That is, *for the layers that are unlikely to degrade the model quality, the warm-up period ends quickly* (e.g., L_1 in Figure 5). On the other hand, for layers with slow training speeds, the warm-up is applied sufficiently long since their training states become stable slowly (e.g., L_3 in Figure 5). That is, *layers likely to adversely affect the model quality are trained with a small learning rate for many iterations*, which helps improve the model quality. As a result, our layer-wise state-aware warm-up improves the model quality in large-batch training by precisely tracking the training state of each layer and determining the proper endpoint of warm-up.

In our warm-up, to determine the proper endpoint of the warm-up for each layer, it is important to accurately estimate the training state of each layer. To this end, we define the state of the training for layer k at iteration t as the accumulation of the gradient variance of the layer,

$$S_{(t,k)} = \sum_{i=0}^{t-1} grad\_var_{(t,k)} \tag{6}$$

To estimate the current state of the training for layers, we use the *cumulative gradient variance* instead of the gradient variance at a particular iteration since the gradient variance at each iteration provides the information only for the data sampled at a particular iteration and tends to fluctuate as illustrated in Figure 3. On the other hand, the cumulative gradient variance provides the information for all data that have been trained since the beginning of the training. Then, given the threshold for determining the endpoint of the warm-up θ and the training state of layer k, $S_{(t,k)}$, LENA adjusts the learning rate, $\eta_{(t,k)}$, for the parameters of layer k at iteration t as follows:

$$\begin{cases} \eta_{(t,k)} \cdot \dfrac{S_{(t,k)}}{\theta}, & \text{if } S_{(t,k)} \le \theta, \\ \eta_{(t,k)}, & \text{otherwise.} \end{cases} \tag{7}$$

In our layer-wise state-aware warm-up, the training state of each layer $S_{(t,k)}$ increases at a different rate until it reaches to the threshold θ, depending on its gradient variance (i.e., $grad\_var_{(t,k)} \in [1, n]$). Thus, the warm-up for each layer ends at a different point and the endpoint of the warm-up for every layer is automatically determined with a single threshold θ. We set $\theta = T_1 \cdot \alpha$, where T_1 is the number of iterations in the case of single-worker training and α is a user-defined hyperparameter. We will empirically evaluate the impact of α on the model quality in Section 4.2.

3.3 Algorithm & Performance Consideration

Let us describe the process of large-batch training with LENA and several performance consideration of LENA. Algorithm 1 shows the whole training process of LENA. At iteration t, for n parallel workers in a distributed cluster, each worker samples a batch of training data b from data X and computes the local gradient based on the data (lines 3-6). Then, the gradients computed by n workers are averaged (line 7). Note that to efficiently aggregate the gradients of multiple workers, we use the distributed communication package of PyTorch[1]. Next, for each layer j, the gradient variance of the layer is computed by Eq. 4 (lines 8-10). Before updating the model, the learning rate for each layer is adjusted by LENA's learning rate function. For each layer j, LENA adjusts its learning rate by the gradient variance of the layer and updates the training state (i.e., cumulative gradient variance) of the layer (lines 17-18). Then, it determines whether applying warm-up or not based on the training state $S_{(t,j)}$ (lines 19-21). After all learning rates for all layers are adjusted, it returns the set of the adjusted learning rates, η_t. Finally,

[1]https://pytorch.org/docs/stable/distributed.html

Algorithm 1 Large-batch training with LENA

```
1:  Function LargeBatchSGD_LENA($w_0, X, f, n, k, \eta_0, \theta$):
2:      for $t = 0, 1, \ldots$ do
3:          for $i = 0, \ldots, n-1$ do   # in n parallel workers
4:              $b \leftarrow$ sample_batch($X$)
5:              $g_t^{(i)} \leftarrow \frac{1}{|b|} \sum_{x \in b} \nabla f(w_t, x)$
6:          end for
7:          $\bar{g}_t \leftarrow \frac{1}{n} \sum_{i=1}^{n} g_t^{(i)}$
8:          for $j = 0, \ldots, k-1$ do   # Computing gradient variance
9:              $grad\_var_{(t,j)} \leftarrow \frac{\frac{1}{n}\sum_{i=0}^{n-1} \|g_{(t,j)}^{(i)}\|^2}{\|\bar{g}_{(t,j)}\|^2}$
10:         end for
11:         $\eta_t \leftarrow$ LENA_lr($grad\_var, t, k, \eta_0, \theta$)
12:         $w_{t+1} \leftarrow w_t - \eta_t \odot \bar{g}_t$
13:     end for
14:     return $w_t$

15: Function LENA_lr($grad\_var, t, k, \eta_0, \theta$):
16:     for $j = 0, \ldots, k-1$ do   # Layer-wise lr scaling
17:         $\eta_{(t,j)} \leftarrow grad\_var_{(t,j)} \cdot \eta_0$
18:         $S_{(t,j)} \leftarrow S_{(t,j)} + grad\_var_{(t,j)}$
19:         if $S_{(t,j)} \le \theta$ then   # State-aware warm-up
20:             $\eta_{(t,j)} \leftarrow \eta_{(t,j)} \cdot \frac{S_{(t,j)}}{\theta}$
21:         end if
22:     end for
23:     return $\eta_t$
```

the model is updated by applying the averaged gradient $\bar{g}_t$ with η_t, where '$\odot$' means the layer-wise product (line 12).

In order to adjust the learning rate of the parameters for each layer more reliably, we keep tracking the moving averages ma_n and ma_d of the numerator and the denominator in Eq. 4 as recommended in [20, 22, 29]. Also, to prevent the division-by-zero problem, we add a small constant ϵ to the numerator and the denominator, respectively. Then, we estimate the gradient variance of layer k at iteration t as

$$grad\_var_{(t,j)} = \frac{ma_n + \epsilon}{ma_d + \epsilon}. \tag{8}$$

Finally, we conduct the complexity analysis for LENA. In general, the overhead of large-batch training with data parallelism is divided into two parts: (1) computation overhead and (2) communication overhead. In the computation part, forward/backward passes and update operations are performed. In the communication part, the computed gradients of workers are aggregated via network communication. Here, the computation overhead depends on the size of the model and input data, approximately $O(m \cdot b)$ where m is the size of the model and b is the batch size. Similarly, the communication overhead depends on the size of the model and the number of workers, approximately $O(m \cdot n)$ where n is the number of workers. Note that these overheads are common to every *lr* scaling method. Then, we focus on the overhead required in the process of *lr* scaling.

In the fixed scaling methods (e.g., linear scaling), there is little additional overhead since they simply multiply *lr* by a constant value. In gradient variance-based methods (e.g., AdaScale), to compute the gradient variance for the entire model, additionally, the square sum of the local gradient should be computed across workers (i.e., $O(m)$ per worker), and aggregated via network communication (i.e., $O(n)$). Lastly, in LENA, the following additional overheads are expected: (1) the overhead required for obtaining the numerator in Eq. 4–i.e., computing the square sums of the local gradients across workers ($O(m)$ per worker) and aggregating them via communication ($O(n)$), and (2) tracking the training state of each layer (i.e., Eq. 6) for the layer-wise state-aware warm-up ($O(k)$ per worker, where k is the number of layers in a DNN model). Despite the additional overhead of LENA, however, it does not significantly affect the training performance since it is usually much smaller than the common overhead of large-batch training, (i.e., $O(m)+O(n)+O(k) \ll O(m\cdot b)+O(m\cdot n)$). We empirically observed that the training performance of LENA is about 5% less than that of the fixed scaling method, and 1% less than that of the gradient variance-based method. However, when we consider the benefits of LENA such that LENA achieves the target accuracy (i.e., the accuracy of single-worker training) with a much fewer iterations than existing *lr* scaling methods, we argue that the small additional overhead of LENA be not a critical issue. We will demonstrate the effectiveness of LENA in terms of the training time in Section 4.2.

4 EXPERIMENTAL VALIDATION

We evaluate LENA by answering the following evaluation questions:

- EQ1. Does LENA improve the model quality in large-batch training better than existing *lr* scaling methods?
- EQ2. How effective are the layer-wise strategies of LENA in improving the model quality in large-batch training?
- EQ3. How sensitive is the model quality of LENA to the hyperparameter α?

4.1 Set-Up

Datasets and models. We evaluate LENA with two widely used CNN models, ResNet-18 with 11 M parameters and ResNet-50 with 23 M parameters [13]. As the training datasets, we use CIFAR-10 and CIFAR-100 [26]. CIFAR-10 and CIFAR-100 consist of 50K images with 10 labels and 100 labels for training, respectively. For both datasets, we randomly select 45K images as training samples and the remaining 5K images as validation samples to evaluate the model quality.

System configuration. We use PyTorch 1.9.0 to implement all methods including LENA on Ubuntu 18.04 OS. We run our experiments on the cluster with four machines, each of which has two NVIDIA RTX 2080 Ti GPUs installed with CUDA 10.2 and cuDNN 8.2.4, and an Intel i7-9700k CPU with 64 GB memory. All machines are inter-connected by 10Gbps Ethernet.

Learning rate scaling methods. We compare LENA with two state-of-the-art *lr* scaling methods, LSW [12] and AdaScale (AS) [20] and a baseline method of single-worker training (with an ideal accuracy). LSW is a method to use the linear *lr* scaling with fixed warm-up. For LSW, we set the warm-up period as the first 5.5% of training epochs as recommended in [12]. AS adaptively adjusts the learning rate based on the gradient variance for the entire model without warm-up [20]. For AS, we set a moving average rate for reliable *lr* scaling as $\max(1-n/1000, 0)$, as recommended in [20]. For LENA, we empirically found the best value for the hyperparameter α and set α as 5% for all experiments. The experimental results about the impact of hyperparameter α on the model quality in large-batch training are included in Section 4.2. Finally, we use single-worker training (i.e., no scaling without warm-up) as a baseline.

Hyperparameter setting. We set per-worker batch size b as 128 for both ResNet-18 and ResNet-50 to fully utilize the GPU memory. We evaluate LENA with many large batch sizes B (1024, 2048, ..., and at most 16384). In our cluster, however, the maximum total batch size is 1024 since there are 8 workers ($n = 8$). To evaluate LENA with larger batch sizes than 1024 (i.e., $n \geq 16, B \geq 2048$), therefore, we implement large-batch training as follows. Given a period each worker computes its local gradient iteratively without updating the model. At the end of the period, each local gradient (i.e., $g^i_{(t,k)}$) is used for computing Eq. 4. We note that the training result of this implementation is theoretically the same as the original result. We use momentum SGD and set momentum as 0.9, weight decay factor as 0.0005, and the base learning rate η_0 as 0.1 for both datasets. We apply the exponential learning rate decaying scheduler with the decaying factor $d = 0.01$.

Metrics. The goal of this work is to train a model with the quality, as good as the quality of single-worker training, with fewer iterations (i.e., less time). Thus, we compare the model accuracy of each method with that of single-worker training (i.e., target accuracy) when training the same amount of training data. We also compare the required iterations and time of each method to achieve the target accuracy. Via preliminary experiments using single-worker training, we set the target accuracy as 95% for the training of ResNet-18 on CIFAR-10 (100 epochs) and 94% for the training of ResNet-50 on CIFAR-100 (150 epochs).

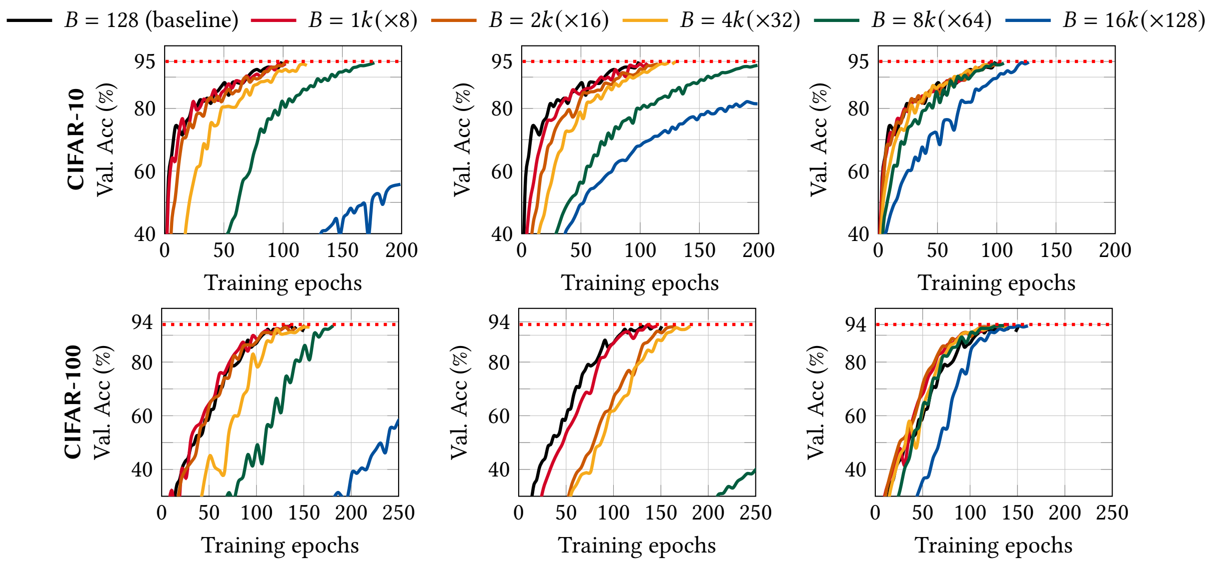
4.2 Experimental Results

EQ1. Model quality. First, we evaluate the quality of the model trained with each *lr* scaling method. We train ResNet-18 on CIFAR-10 (200 epochs) and ResNet-50 on CIFAR-100 (300 epochs) with five different batch sizes, and measure top-1 accuracy at 100th epoch for CIFAR-10 and 150th epoch for CIFAR-100. Table 1 and Figure 6 show the results. The results demonstrate that for all batch sizes, LENA achieves (1) *the highest accuracy when training the same amount of training data*, (2) *the target accuracy within the fewest iterations across five different batch sizes* (up to 45.2% fewer iterations and 44.7% shorter time than the existing state-of-the-art gradient variance-based method (AS)), and (3) *the target accuracy for the training with very large-batch sizes* (e.g., 8.19k and 16.4k) that surpass the limits of existing *lr* scaling methods.

In addition, in terms of training time, LENA outperforms existing *lr* scaling methods only except for the training on CIFAR-10 with the batch size 1024 (This size is small enough for all *lr* scaling methods to perform well). This is because, despite the additional overhead of LENA, LENA achieves the target accuracy within much fewer iterations than other *lr* scaling methods. This result indicates that the additional overhead of LENA to consider the difference across layers in training is worthwhile in improving both the model quality and the training performance.

Table 1: Comparison of the model accuracy and the required epochs/iterations/time to achieve the target accuracy for various batch sizes n (The bold font indicates the best results, and the underlined results represent the results even “better" than the single-worker training).

Dataset	n	Total batch size	Valid. Acc.			Epochs			Iterations			Time (sec.)		
			LSW	AS	LENA	LSW	AS	LENA	LSW	AS	LENA	LSW	AS	LENA
CIFAR-10	8	1.02k	0.9494	0.9424	**0.9488**	104	109	**104**	4.58k	4.80k	**4.58k**	**579**	625	636
	16	2.05k	0.9444	0.9296	**0.9498**	109	117	**102**	2.40k	2.57k	**2.24k**	582	628	**566**
	32	4.10k	0.9186	0.9104	**0.9514**	124	126	**98**	1.36k	1.39k	**1.07k**	636	650	**515**
	64	8.19k	0.8224	0.7980	**0.9446**	183	195	**107**	1.01k	1.07k	**588**	923	987	**546**
	128	16.4k	0.3084	0.6856	**0.9058**	N/A	N/A	**128**	N/A	N/A	**352**	N/A	N/A	**645**
CIFAR-100	8	1.02k	0.9410	0.9406	**0.9454**	152	151	**141**	6.69k	6.64k	**6.20k**	1737	1794	**1692**
	16	2.05k	0.9352	0.9094	**0.9478**	145	171	**133**	3.19k	3.76k	**2.92k**	1534	1881	**1477**
	32	4.10k	0.9334	0.8856	**0.9434**	167	183	**141**	1.83k	2.01k	**1.55k**	1704	1941	**1510**
	64	8.19k	0.8428	0.1664	**0.9450**	183	N/A	**141**	1.00k	N/A	**775**	1823	N/A	**1475**
	128	16.4k	0.1650	0.0914	**0.9352**	N/A	N/A	**174**	N/A	N/A	**478**	N/A	N/A	**1779**



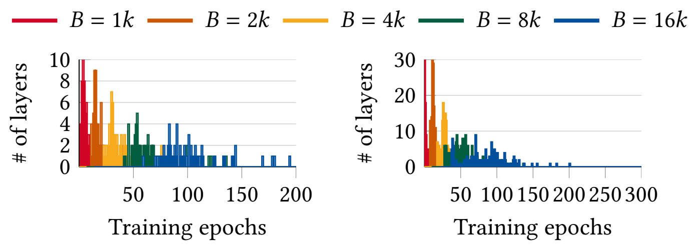
(a) LSW **(b) AdaScale** **(c) LENA**

Figure 6: Top-1 accuracies of each *lr* scaling method for various batch sizes B. LENA always achieves the target accuracy within the fewest iterations. The dotted lines in figures indicate the target accuracies of the single-worker case.

EQ2. Effectiveness of layer-wise strategies. In this experiment, we verify the effectiveness of the layer-wise *lr* scaling and state-aware warm-up of LENA on the model quality. We compare all combinations for *lr* scaling methods (LSW, AS, and LENA) and warm-up methods (no warm-up, fixed warm-up, and our warm-up). Table 2 shows the results. When the same warm-up method is applied to each *lr* scaling method, LENA can achieve the highest accuracy compared to other *lr* scaling methods (only except for $n = 8$). In particular, LENA improves the model quality by 12.4% without warm-up, 7.23% with the fixed warm-up, and 5.61% with our warm-up on average compared to AS, where these improvements in the model quality indicate the effectiveness of the layer-wise *lr* scaling of LENA in large-batch training. Next, for each *lr* scaling method, let us evaluate the effect of each warm-up method on the model quality. Our state-aware warm-up improves the model quality more than the fixed warm-up, especially in the training with 16k of batch size. This result implies that the significant degradation of the model quality can occur unless addressing the issue that the model fluctuates steeply in the early stage of large-batch training, and our layer-wise state-aware warm-up successfully addresses the issue by tracking the training state for each individual layer. For more in-depth analysis of our layer-wise state-ware warm-up, we measure the warm-up endpoint of each layer. Figure 7 shows the results, where the x-axis represents the training epoch and the y-axis represents the number of layers ending its warm-up at the epoch. As clearly shown in Figure 7, the warm-up endpoint differs across layers and the warm-up is applied to each layer for a longer period as the batch size increases. This result indicates that the

Table 2: Comparison of effects of warm-up methods on the model accuracy. Our layer-wise state-aware warm-up significantly improves the model quality of *lr* scaling methods (The underlined warm-up indicates the default warm-up of each *lr* scaling).

Dataset	n	Total batch size	Linear Scaling (LS)			AdaScale (AS)			LENA		
			No	Fixed	Ours	No	Fixed	Ours	No	Fixed	Ours
CIFAR-10	8	1.02k	0.9340	**0.9494**	0.9462	0.9424	**0.9478**	0.9438	0.9482	0.9466	**0.9488**
	16	2.05k	0.9326	0.9444	**0.9448**	0.9296	0.9436	**0.9486**	0.9456	0.9484	**0.9498**
	32	4.10k	0.8952	0.9186	**0.9364**	0.9104	0.9434	**0.9480**	0.9472	0.9488	**0.9514**
	64	8.19k	0.8414	0.8224	**0.8836**	0.7980	0.9024	**0.9376**	0.9340	0.9444	**0.9446**
	128	16.4k	0.1594	0.3084	**0.6364**	0.6856	0.7274	**0.8946**	0.8378	0.8854	**0.9058**
CIFAR-100	8	1.02k	0.9300	0.9410	**0.9450**	0.9406	**0.9478**	0.9446	0.9398	0.9454	**0.9454**
	16	2.05k	0.9218	0.9352	**0.9446**	0.9094	0.9434	**0.9438**	0.9442	0.9456	**0.9478**
	32	4.10k	0.9198	0.9334	**0.9352**	0.8856	0.9358	**0.9384**	0.9378	0.9410	**0.9434**
	64	8.19k	0.8578	0.8428	**0.8932**	0.1664	0.8560	**0.9016**	0.6210	0.9390	**0.9450**
	128	16.4k	0.1482	0.1650	**0.2842**	0.0914	0.0681	**0.4548**	0.4442	0.4946	**0.9352**



(a) CIFAR-10 **(b) CIFAR-100**

Figure 7: Different warm-up endpoints of layers in large-batch training with LENA.

point at which each layer becomes stable (i.e., the optimal endpoint) differs and such differences across layers should be considered for improving the model quality, as we claimed in Section 3.2.

EQ3. Hyperparameter sensitivity. Finally, we evaluate the hyperparameter sensitivity of LENA and provide the best value for the hyperparameter α, improving the model quality most in large-batch training. As explained in Section 3.2, hyperparameter α determines the threshold θ for the layer-wise state-aware warm-up. As α becomes larger, the longer the warm-up is applied to layers. We compare the model quality of LENA with varying $\alpha = 1\%, 3\%, 5\%, 7\%, 9\%$ in the training with five different batch sizes. Table 3 shows the results. LENA achieves high quality of the model with a wide range of α and batch sizes (except for the very short period of the warm-up and very large batch size together, $\alpha = 1\%$ and $B = 16k$). Based on these results, we conclude that the model quality of LENA is insensitive to α, and recommend to set α as 5%. As a result, LENA not only improve the model quality much better than existing *lr* scaling methods, but also rarely requires much trial-and-error tuning to find a best value for the hyperparameter.

5 CONCLUSION

In this work, we observed that the state of the training tends to vary across layers in a DNN model training, and identified the cause of the degradation in the model quality in existing *lr* scaling methods – i.e., overlooking the difference across layers in training. Based on this observation, we proposed a novel *lr* scaling approach toward large-scale DNN training, named as LENA, that successfully addresses the issue with two layer-wise strategies: (1) a layer-wise adaptive learning rate scaling and (2) a layer-wise state-aware warm-up. Through the comprehensive evaluation with variations of batch sizes, we demonstrated that LENA achieves the target accuracy (i.e., the accuracy of single-worker training) with the fewest iterations across different batch sizes, and for training very large-batch sizes that surpass the limits of all state-of-the-art methods. We also verified that each of our layer-wise strategies significantly improves the model quality in large-batch training. In future work, we plan to evaluate LENA with larger sizes of models and data, and in different machine learning tasks such as speech recognition and machine translation.

Table 3: The impact of hyperparameter α on the model quality of LENA in the training with different batch sizes.

	n	B	Warm-up threshold α				
			1%	3%	5%	7%	9%
CIFAR-10	8	1k	0.9488	0.9418	0.9448	**0.9514**	0.9492
	16	2k	0.9420	0.9454	**0.9498**	0.9432	0.9413
	32	4k	0.9428	0.9508	**0.9514**	0.9504	0.9490
	64	8k	0.9432	0.9438	**0.9446**	0.9436	0.9434
	128	16k	0.8412	0.8932	**0.9058**	0.8928	0.8892
CIFAR-100	8	1k	0.9448	0.9458	0.9454	0.9444	**0.9456**
	16	2k	0.9380	0.9358	**0.9478**	0.9468	0.9446
	32	4k	0.9410	0.9426	**0.9434**	0.9412	0.9392
	64	8k	0.9402	0.9430	**0.9450**	0.9416	0.9396
	128	16k	0.6802	0.9336	**0.9352**	0.9290	0.9266

ACKNOWLEDGMENTS

The work of Sang-Wook Kim was supported by the National Research Foundation of Korea (NRF) under Project Number NRF-2020R1A2B5B03001960, and Institute of Information & Communications Technology Planning & Evaluation (IITP) under Project Number 2020-0-01373. The work of Dongwon Lee was supported by the NSF award #1820609.

REFERENCES

[1] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. 2019. Stochastic gradient push for distributed deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 344–353.

[2] Medha Atre, Birendra Jha, and Ashwini Rao. 2021. Distributed Deep Learning Using Volunteer Computing-Like Paradigm. *arXiv preprint arXiv:2103.08894* (2021).

[3] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. 2020. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)* 117, 48 (2020), 30071–30078.

[4] Oded Ben-David and Zohar Ringel. 2019. The role of a layer in deep neural networks: a Gaussian Process perspective. *arXiv preprint arXiv:1902.02354* (2019).

[5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[6] Dong-Kyu Chae, Jin-Soo Kang, Sang-Wook Kim, and Jaeho Choi. 2019. Rating augmentation with generative adversarial networks towards accurate collaborative filtering. In *Proceedings of the World Wide Web Conference (WWW)*. 2616–2622.

[7] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI))*. 571–582.

[8] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Proceedings of the Advances in Neural Information Processing Systems*. 1223–1231.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 248–255.

[10] Aditya Devarakonda, Maxim Naumov, and Michael Garland. 2017. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029* (2017).

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[12] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[14] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Proceedings of the Advances in Neural Information Processing Systems*. 1223–1231.

[15] Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 1729–1739.

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

[17] Prateek Jain, Sham Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. 2018. Parallelizing stochastic gradient descent for least squares regression: mini-batching, averaging, and model misspecification. *Journal of Machine Learning Research* 18 (2018).

[18] Stanisław Jastrzębski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. 2017. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623* (2017).

[19] Yuting Jia, Qinqin Zhang, Weinan Zhang, and Xinbing Wang. 2019. Communitygan: Community detection with generative adversarial nets. In *Proceedings of the World Wide Web Conference (WWW)*. 784–794.

[20] Tyler Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. 2020. AdaScale SGD: A User-Friendly Algorithm for Distributed Training. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 4911–4920.

[21] Tyler B Johnson and Carlos Guestrin. 2018. Training deep models faster with robust, approximate importance sampling. *Advances in Neural Information Processing Systems* 31 (2018), 7265–7275.

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

[23] Yunyong Ko, Kibong Choi, Hyunseung Jei, Dongwon Lee, and Sang-Wook Kim. 2021. ALADDIN: Asymmetric Centralized Training for Distributed Deep Learning. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*.

[24] Yunyong Ko, Kibong Choi, Jiwon Seo, and Sang-Wook Kim. 2021. An In-Depth Analysis of Distributed Training of Deep Neural Networks. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 994–1003.

[25] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997* (2014).

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

[27] Youngnam Lee, Sang-Wook Kim, Sunju Park, and Xing Xie. 2018. How to impute missing ratings? Claims, solution, and its application to collaborative filtering. In *Proceedings of the World Wide Web Conference (WWW)*. 783–792.

[28] Siyuan Ma, Raef Bassily, and Mikhail Belkin. 2018. The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 3325–3334.

[29] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. 2018. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162* (2018).

[30] Anshul Mittal, Noveen Sachdeva, Sheshansh Agrawal, Sumeet Agarwal, Purushottam Kar, and Manik Varma. 2021. ECLARE: Extreme Classification with Label Graph Correlations. In *Proceedings of the World Wide Web Conference (WWW)*. 3721–3732.

[31] Junjie Qian, Taeyoon Kim, and Myeongjae Jeon. 2021. Reliability of Large Scale GPU Clusters for Deep Learning Workloads. In *Companion Proceedings of the Web Conference (WWW)*. 179–181.

[32] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*. 693–701.

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[34] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[35] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. 2018. Don't Decay the Learning Rate, Increase the Batch Size. In *Proceedings of International Conference on Learning Representations (ICLR)*.

[36] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 6105–6114.

[37] Ran Xin, Soummya Kar, and Usman A Khan. 2020. Decentralized stochastic optimization and machine learning: A unified variance-reduction framework for robust performance and fast convergence. *IEEE Signal Processing Magazine* 37, 3 (2020), 102–113.

[38] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE transactions on Big Data* 1, 2 (2015), 49–67.

[39] Weizheng Xu, Youtao Zhang, and Xulong Tang. 2021. Parallelizing DNN Training on GPUs: Challenges and Opportunities. In *Companion Proceedings of the Web Conference (WWW)*. 174–178.

[40] Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. 2018. Gradient diversity: a key ingredient for scalable distributed learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 1998–2007.

[41] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888* (2017).

[42] Yang You, Jonathan Hseu, Chris Ying, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large-batch training for LSTM and beyond. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–16.

[43] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962* (2019).

[44] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. 2017. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 181–193.

[45] Sixin Zhang, Anna E Choromanska, and Yann LeCun. 2015. Deep learning with elastic averaging SGD. In *Proceedings of the Advances in Neural Information Processing Systems*. 685–693.

[46] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. 2017. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning (ICML)*. PMLR, 4120–4129.