ACHIEVING SMALL-BATCH ACCURACY WITH LARGE-BATCH SCALABILITY VIA ADAPTIVE LEARNING RATE ADJUSTMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider synchronous data-parallel neural network training with fixed large batch sizes. While the large batch size provides a high degree of parallelism, it likely degrades the generalization performance due to the low gradient noise scale. We propose a two-phase adaptive learning rate adjustment framework that tackles the poor generalization issue in large-batch training. Our empirical study shows that the number of training epochs before decaying the learning rate strongly affects the final accuracy. The framework performs extra epochs using the large learning rate even after the loss is flattened. After sufficient training under the noisy condition, the framework decays the learning rate based on the observed loss landscape at run-time. Our experimental results demonstrate that the proposed heuristics and algorithm enable to use an extremely large batch size while maintaining the model accuracy. For CIFAR-10 classification with ResNet20, our method achieves 92.66% accuracy using $8,192$ batch size, which is close to 92.83% achieved using 128 batch size, at a negligible extra computational cost.

1 INTRODUCTION

Synchronous Stochastic Gradient Descent (SGD) with data parallelism is the most popular parallelization strategy for neural network training. In data parallel training, the degree of parallelism is limited by the number of samples in each mini-batch. Thus, increasing the batch size is an intuitive solution to improve the scalability. However, many researchers have theoretically and empirically shown that the large batch size can degrade generalization performance (Keskar et al. (2016); You et al. (2017); Goyal et al. (2017); Li et al. (2019); Lewkowycz et al. (2020)). The large batch size reduces the variance of stochastic gradients, and it results in making the model rapidly converge to a local minimum. Such a fast convergence under a low noise condition is known to harm the generalization performance in non-convex optimization problems.

Researchers have put much effort into improving the generalization performance of large-batch training. There are offline learning rate scaling methods such as 'linear scaling rule' (Goyal et al. (2017)) and 'root scaling rule' (Hoffer et al. (2017)). Layer-wise Adaptive Rate Scaling (LARS) is the most popular online learning rate adjustment method (You et al. (2017)). AdaScale SGD (Johnson et al. (2020)) adaptively adjusts the learning rate based on estimated gradient variance at run-time. Recently, cosine annealing (Loshchilov & Hutter (2016)) has been highlighted based on its empirical successes. There are also several optimizers that tackle the generalization performance issue, such as Sharpness-Aware Minimization (SAM) (Foret et al. (2020)), SmoothOut (Wen et al. (2018)), and EXTRAP-SGD (Lin et al. (2020)). While all these works tackle the poor generalization issue from different angles, the impact of learning rate decay is largely overlooked.

In this work, we focus on how to decay the learning rate for accurate large-batch training of neural networks. Specifically, we address the following two questions:

- *When should the learning rate be decayed?*
- *How much should the learning rate be decayed?*

By answering these, one can better understand the impact of learning rate decay on the generalization performance and design learning rate schedules that achieve the small-batch accuracy together with the large-batch scalability. The most conventional learning rate adjustment strategy is to decay the learning rate by a small constant factor once the training loss is flattened. It is also common to employ a deterministic schedule such that the learning rate is reduced after 50% and 75% of the pre-defined epoch budget (He et al. (2016); Goyal et al. (2017); Lin et al. (2018; 2020)). Such naive decay schedules can significantly harm the generalization performance.

We propose a two-phase learning rate adjustment framework for large-batch neural network training. Our framework is built upon critical heuristics regarding the impact of the learning rate on the generalization performance. The first phase begins with a large learning rate that encourages the model to explore parameter space rather than quickly falling into a minimum. We find that the length of the training under such a noisy condition strongly affects the generalization performance. Based on our observations and analysis, we present a simple but effective heuristic about how to determine the length of training before decaying the learning rate. The proposed heuristic is verified by analyzing how the margin distribution of neural networks evolves during the training. After the model is trained enough under the noisy condition, in the second phase, our framework enforces the convergence by adaptively decaying the learning rate based on the observed geometric information of parameter space. Our empirical study demonstrates that the Hessian-aware learning rate decay enables the model to converge to a flatter minimum that can better generalize to unknown data.

We evaluate the performance of the proposed learning rate adjustment framework using popular benchmark datasets such as CIFAR-10, CIFAR-100, SVHN, and ImageNet. Our experimental results demonstrate that our heuristics and adaptive decay algorithm allow to use an extremely large batch size without a significant loss in validation accuracy. We also show that the validation accuracy can be further improved by harmonizing our framework with other large-batch training techniques. For example, when our framework is used together with LARS, the large-batch $(8,192)$ training of ResNet-20 (He et al. (2016)) on CIFAR-10 achieves $92.66 \pm 0.2\%$ validation accuracy that is almost the same as the small-batch (128) training accuracy $92.83 \pm 0.1\%$.

Our contributions can be summarized as follows.

1. We propose a two-phase learning rate adjustment framework for accurate large-batch training. The framework is independently applicable to many optimization algorithms and learning rate scaling methods.
2. Our empirical study demonstrates that the generalization performance of large-batch training can be remarkably improved by the appropriate learning rate adjustment. We provide a margin distribution analysis to verify our heuristics.
3. We present comprehensive experimental results across several benchmark datasets and neural networks. We compare the proposed framework to the popular large-batch training methods such as linear scaling rule and LARS.

2 BACKGROUND

2.1 SYNCHRONOUS SGD WITH DATA PARALLELISM

In this work, we consider minimization problems of the form

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left[F(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w}, \xi_i) \right], \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^d$ represents the model parameters, N is the number of training samples, ξ_i is the i^{th} training sample, and f is a non-convex loss function. Mini-batch Stochastic Gradient Descent (SGD) iteratively adjusts the model parameters using the gradients computed from a random subset of the training samples (called mini-batch) until the stop condition is satisfied.

Synchronous SGD with data parallelism is the most popular parallelization strategy for mini-batch SGD. Given N training samples, each m random samples (a mini-batch) are evenly distributed to all P workers and processed in parallel. Since the workload is partitioned on the data dimension, this parallelization strategy is called *data parallelism*. Each worker is assigned with $\frac{m}{P}$ training samples and calculates the gradient of loss function f with respect to the model parameters $\mathbf{w}$ using the assigned samples. Then, the local gradient sums are aggregated across all the workers using inter-process communications. Since each worker computes a fixed number of local gradient sums from each mini-batch, the communication cost is independent of m. Given a fixed N, the larger the m, the fewer the communications per epoch. Thus, the scaling efficiency is improved as m increases. This property motivates researchers to employ large batch sizes for data-parallel training.

2.2 Previous Large-Batch Training Methods

Recently, it has been empirically showed that the batch size can be increased to a certain problem-specific threshold without an accuracy drop (Goyal et al. (2017)). Given the best-tuned batch size B_0 and learning rate μ_0, they proposed to use a proportionally increased learning rate for large-batch training such that $\mu = \mu_0 \frac{B}{B_0}$. This technique is called 'linear scaling ruleq. In (Smith et al. (2017)), the authors analyzed the gradient noise scale g in momentum SGD as follows.

$$g \approx \frac{\mu N}{m(1-\kappa)}, \tag{2}$$

where κ is the momentum factor. Given a fixed batch size m, the smaller the learning rate μ, the lower the noise scale. This analysis explains why the linear scaling rule practically works well. AdaScale is an adaptive learning rate scaling method (Johnson et al. (2020)), which scales the learning rate at run-time based on the estimated gradient variance. While it achieves good validation accuracy, it still assumes the conventional hand-tuned learning rate decay schedules. You et al. proposed Layer-wise Adaptive Rate Scaling (LARS) that adjusts the learning rate in a layer-wise way (You et al. (2017)). Several large-scale scientific applications used LARS to scale up their deep learning solutions (Mathuriya et al. (2018); Kurth et al. (2018)). LARS effectively improves the generalization performance of large-batch training by having one extra hyper-parameter (confidence coefficient) and a moderate extra computational cost.

Sharpness-Aware Miminization (SAM) is an optimizer that effectively improves the generalization performance by minimizing a surrogate loss (Foret et al. (2020)). Note that SAM is a general optimizer that is not specifically designed for large-batch training. While it successfully improves the generalization performance, the gradients are calculated twice per iteration having a non-negligible extra computational cost.

Some researchers proposed algorithms that generate inherent noise to improve the generalization performance. SmoothOut (Wen et al. (2018)) perturbs the model parameters using uniform noise every iteration. Such noisy gradients help the model head towards a local minimum that achieves better generalization performance. EXTRAP-SGD (Lin et al. (2020)) is based on a similar principle.

3 Two-Phase Large-Batch Training

We propose an adaptive learning rate adjustment framework for large-batch neural network training. The framework consists of two phases: *exploration* and *convergence* phases. Algorithm 1 shows the framework applied to mini-batch SGD.

3.1 Exploration Phase

A Close Look at Training Progress – When training neural networks, it is common to decay the learning rate when the training loss is flattened. The flattened loss implies that the training cannot be further progressed using the current learning rate due to the high noise scale. However, we find that such loss-based learning rate schedules can significantly harm the generalization performance especially in large-batch training.

To better understand the training progress, we analyze how margin distribution is improved during training. The margin is a distance of the decision boundary to each training data point, which represents the network's generalization performance. We use a margin approximation technique proposed in (Jiang et al. (2018)). The detailed settings are provided in Appendix. Figure 1 a) and b) show the comparison between CIFAR-10 loss curve and margin curve using a batch size of 128 and $8,192$, respectively. Figure 1 c) and d) show the comparison between CIFAR-100 loss curve and margin curve using a batch size of 128 and $8,192$, respectively. The learning rate is fixed to the

Algorithm 1: Two-phase adaptive LR adjustment framework applied to mini-batch SGD.

```
Input: $\mu_0$: initial LR, $w_0$: initial model, $\beta$: LR decay factor, $E$: total epoch budget,
       $\mathcal{D}$: training dataset
1  $\mu \leftarrow \mu_0$ ;
2  Begin the training in exploration phase. ;
3  for $e = 1$ to $E$ do
4      $w_e =$ Mini-batch SGD ($w_{e-1}$, $\mathcal{D}$);
5      if exploration phase then
6          if phase switching condition is met then
7              $h'_{max} \leftarrow$ maximum Hessian Eigenvalue($w_e$);
8              $f'(w_e) \leftarrow$ average training loss obtained in epoch $e$;
9              Switch to convergence phase.;
10     else if convergence phase then
11         $h_{max} \leftarrow$ maximum Hessian Eigenvalue($w_e$);
12         $f(w_e) \leftarrow$ average training loss obtained in epoch $e$;
13         if $h'_{max} > 0$ then
14             if $h'_{max} < h_{max}$ and $f'(w_e) < f(w_e)$ then
15                 $\mu = \beta\mu$;
16                 $h'_{max} = 0$;
17         else
18             $h'_{max} = h_{max}$;
19             $f'(w_e) = f(w_e)$;
```

initial value for the whole training. The vertical bars on the loss curves indicate the epoch in which a plateau is detected (no improvement for 10 epochs). When using a large batch size, the loss is flattened first while the margin is still being increased. In contrast, when using a small batch size, the margin is sharply increased in the early epochs and then the loss is flattened later.

This observation provides a key insight about the training progress. In large-batch training, the training loss plateau does not indicate a sufficient training progress in terms of the generalization performance. Although the training loss is not effectively reduced after the plateau, the underlying margin distance can be further increased. Table 1 shows a CIFAR-10 validation accuracy comparison across different learning rate decay schedules. The later the first learning rate decay step, the higher the final accuracy. This result is well aligned with our margin distribution analysis.

Extra Epochs After Loss Plateau – It is not practical to calculate the margin at run-time due to the expensive computational cost. Instead, based on our analysis, we propose a simple but effective heuristic about the learning rate decay in large-batch training as follows.

- (*Phase switching condition*): If a loss plateau is detected, train the model for α extra epochs before decaying the learning rate.

We call the epochs before the first learning rate decay step *exploration* phase. The conventional loss-based decay schedules have a relatively short *exploration* phase, and it can degrade the generalization performance. The above heuristic suggests increasing the length of *exploration* phase so that the model has sufficient time to not only fit to the training data but also improve the underlying margin distribution. Given this general heuristic, one can easily determine α depending on the allowed epoch budget. In our experiments, when a loss plateau is detected in x epochs, we set α to $x/3$. We found such a simple heuristic considerably improved the final validation accuracy across many representative benchmark datasets. The effectiveness of this heuristic will be verified in Section 4.

3.2 Convergence Phase

Given the model that has been sufficiently trained under a noisy condition in *exploration* phase, our focus now is on making the model rapidly converge to a minimum by adjusting the degree of noise. We call these epochs *convergence* phase.

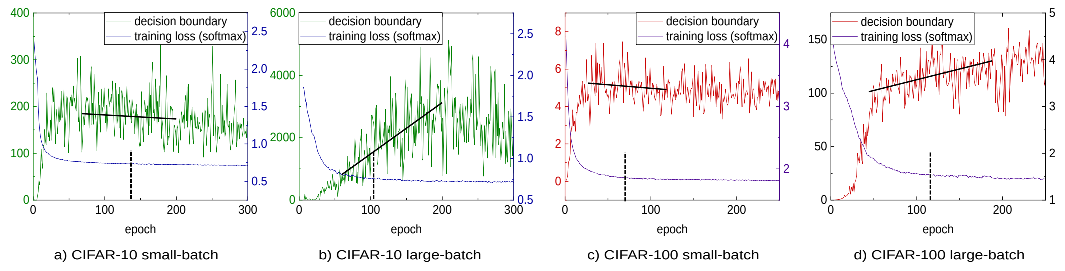

Figure 1: The comparisons of margin distribution and loss curves. CIFAR-10 (ResNet20): a) and b) show the curves for batch size 128 and 8,192. CIFAR-100 (WRN28-10): c) and d) show the curves for batch size 128, and 8,192. The left axis is margin and right axis is loss. The dotted vertical lines indicate the epoch in which the loss plateaus (no improvement for 10 epochs).

Table 1: CIFAR-10 classification performance comparison. The epoch budget is fixed to 300.

model	batch size	Before decay	After 1st decay	After 2nd decay	accuracy
ResNet-20	8192	100	100	100	$87.72 \pm 0.2\%$
		150	75	75	$90.23 \pm 0.1\%$
		200	50	50	$91.43 \pm 0.1\%$
		250	25	25	$91.40 \pm 0.1\%$

Learning Rate for Convergence – Given a loss function $F(w, x)$, the loss surface nearby a local minimum w^* can be approximated using a quadratic approximation as follows.

$$F(w, x) \approx F(w^*, x) + \frac{1}{2}(w - w^*)^\top H^*(w - w^*), \tag{3}$$

where x is the input data sample and H^* is the mean of positive-definite Hessian at the minimum. Note that the first-order derivative is supposed to be zero at the minimum, and thus Equation 3 only has the second-order derivative term. By plugging in the gradient of Equation 3 into the gradient descent equation, $w_{t+1} = w_t - \eta_t \nabla F(w_t)$, we can derive the following condition for convergence (Seong et al. (2018)).

$$w_{t+1} \approx w_t - \eta_t H^*(w_t - w^*) \tag{4}$$

$$w_{t+1} - w^* \approx (I - \eta_t H^*)(w_t - w^*), \tag{5}$$

where η_t is the learning rate at iteration t. So, the convergence can be guaranteed if the learning rate is sufficiently small as below.

$$|1 - \eta_t h_{max}| < 1 \tag{6}$$

$$0 < h_{max} < \frac{2}{\eta_t}, \tag{7}$$

where h_{max} is the maximum Eigenvalue of the Hessian. This analysis shows that the model can converge to a certain stationary point if the learning rate is smaller than $\frac{2}{h_{max}}$. That is, the sharper the loss landscape, the smaller the learning rate should be to guarantee the convergence. Motivated by this observation, we design a Hessian-aware learning rate adjustment strategy as follows.

Hessian-Aware Adaptive Learning Rate Decay – Algorithm 1 adjusts the learning rate by jointly considering the training loss and the Hessian information. Once the training proceeds to the *convergence* phase, the algorithm calculates the largest Hessian Eigenvalue h_{max} at the end of every epoch. Then, it compares h_{max} to that of the previous epoch $h^{'}_{max}$ to estimate the loss landscape. If $h^{'}_{max} < h_{max}$ and $f^{'}(w) < f(w)$, the learning rate is multiplied by a positive constant β smaller than 1. The algorithm resets $h^{'}_{max}$ to 0 at line 16 after decaying the learning rate so that the learning rate is not decayed in two consecutive epochs. This setting enables to fairly compare the Hessian Eigenvalues obtained using the same learning rate only. It also stabilizes the training loss by enforcing the model to be trained using the same learning rate for at least two epochs.

In *convergence* phase, Algorithm 1 focuses on the situation in which the training loss and the principal Hessian Eigenvalue increase together. Figure 2 presents simplified illustrations of such a case.

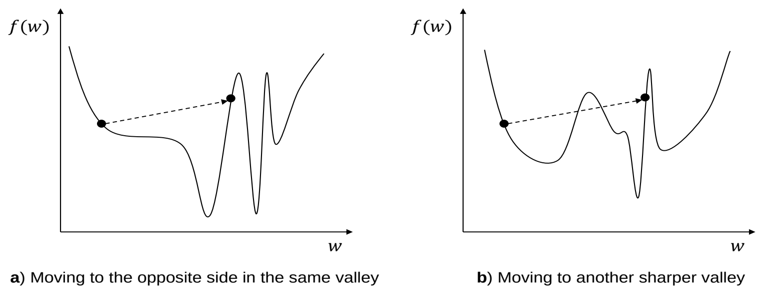
f(w)
w
a) Moving to the opposite side in the same valley
f(w)
w
b) Moving to another sharper valley

Figure 2: Simplified example illustrations of two possible cases in which the training loss increases together with the principal Hessian Eigenvalue: **a**) The model jumps to the opposite side in the same valley or **b**) the model jumps out of the current valley and moves to another sharper valley.

The curve is the loss function $f(w)$ and the black dot is the model with a single parameter w. First, if the learning rate is large, the model can jumps to the opposite side in the same valley (Figure 2.**a**). If the opposite side is sharper, the larger principal Hessian Eigenvalue will be observed. Second, the model can even jump out of the current valley and move to another sharper valley (Figure 2.**a**). In both cases, as shown in Equation 7, the smaller learning rate can encourage the model to rapidly move toward the minimum. Although the loss increases, if the Hessian Eigenvalue is not increased, we consider that the model is moving toward a flatter region and thus do not adjust the learning rate.

The proposed adaptive learning rate adjustment algorithm is also motivated by the inexact line search algorithms such as backtracking line search (Armijo (1966)). Backtracking line search algorithm iteratively finds the learning rate for every parameter update, that is small enough to effectively reduce the loss (Nocedal & Wright (2006)). The algorithm begins with a large initial learning rate and repeatedly reduces it until Armijo-Goldstein's condition (Armijo (1966)) is satisfied. However, such iterative full gradient computations can be even more expensive than multiple SGD epochs making it impractical. Instead, our algorithm adjusts the learning rate only when it detects the condition where the loss landscape has been sharper and the training is not further progressed.

Hessian Approximation – Computing the Hessian of neural networks is considered as infeasible due to the large dimensions. So, it is essential to efficiently approximate the Hessian during training in order to find good learning rates without significantly increasing the training time. We approximate the principal Hessian Eigenvalue h_{max} in multiple steps as follows.

- Obtain the Hessian information from the largest output-side layer.
- The principal Hessian Eigenvalue is directly approximated using power iteration method.
- Stochastic approximation of Hessian Eigenvalue using one large mini-batch.

First, we calculate the principal Hessian Eigenvalue only at the largest output-side layer instead of the whole model parameters. This can be considered as a diagonal approximation of Hessian with respect to the largest right-bottom corner diagonal block. It has been observed that, for Convolutional Neural Networks, the output layer converges first and then all the other layers converge from the bottom up (Raghu et al. (2017)). Likely, Recurrent Neural Networks are also known to converge from the bottom up (Morcos et al. (2018)). Thus, by adjusting the learning rate based on the Hessian information at the largest output-side layer, the noise scale can be reduced after most of the other layers have been sufficiently trained.

Second, we use the power-iteration method to directly approximate the largest Hessian Eigenvalue without computing Hessian (LeCun et al. (1992)). The Hessian-vector product can be approximated using finite difference method as follows.

$$Hv \approx \frac{\nabla F(W + \alpha v) + \nabla F(W)}{\alpha}, \tag{8}$$

where W is the model parameters, v is a random vector, and α is a small constant. Given any random vector v that is not orthogonal to the principal Eigenvector of H, the power iteration method makes

v converge to the principal Eigenvector.

$$v \leftarrow \frac{Hv}{\|v\|} \tag{9}$$

By applying Equation 8 to 9, the following power iteration-based Hessian Eigenvalue approximation method can be derived.

$$v \leftarrow \frac{\nabla F(W + \alpha v) + \nabla F(W)}{\alpha \|v\|} \tag{10}$$

Third, we approximate the Hessian Eigenvalue using only a single large mini-batch. In the large-batch training, each mini-batch is likely large enough to represent the whole dataset. Thus, h_{max} can be accurately approximated from only one large mini-batch instead of the total dataset. For instance, in our CIFAR-10 classification experiments, we found that the principal Hessian Eigenvalue approximated from a single mini-batch of size $8,192$ was almost the same as the full approximation.

In summary, our framework approximates the principal Hessian Eigenvalue h_{max} at the largest output-side layer using a single large mini-batch based on the power-iteration method.

4 EXPERIMENTS

Experimental Settings – We use TensorFlow 2.4.0 and Horovod 0.19.0 for our experiments. We run all the experiments on a distributed-memory cluster that consists of 4 compute nodes each of which has 2 NVIDIA v100 GPUs. Due to the limited compute resources, we split each large mini-batch to multiple sub-batches and process them sequentially. The model parameters are updated using the gradients averaged across all the sub-batches. This approach allows to run large-batch training exploiting the limited memory space while not affecting the model accuracy.

Hyper-Parameter Settings – For all the benchmark datasets, we borrow the hyper-parameter settings from the reference works and further tune the learning rate. For CIFAR-10 and CIFAR-100, we use the settings reported in (Lin et al. (2020)). We use the ImageNet settings reported in (Goyal et al. (2017)). We tune all the hyper-parameters from the scratch for SVHN. The detailed settings are found in Appendix. We use momentum SGD with a momentum factor of 0.9, and the gradual learning rate warmup (Goyal et al. (2017)) is applied to the first 10 epochs. The loss is considered as flattened if it is not reduced for 5% of the epoch budget.

4.1 CLASSIFICATION PERFORMANCE COMPARISON

We compare the classification performance across five different configurations.

1. *small-batch*: The best-tuned small batch size and learning rate.
2. *large-batch*: The large-batch training based on linear scaling rule (Goyal et al. (2017)).
3. *LARS*: Layer-wise Adaptive Rate Scaling (You et al. (2017)).
4. *two-phase*: The proposed two-phase framework.
5. *two-phase* + *LARS*: The proposed two-phase framework together with LARS.

Table 2, 3, 4, and 5 show the accuracy comparisons for CIFAR-10, CIFAR-100, SVHN, and ImageNet, respectively. The full learning curves can be found in Appendix. In the decay epochs column, adaptive(k) means that the learning rate is decayed after k epochs. Once the first loss plateau is detected in α epochs, we switch the phase to *convergence* phase after $\alpha/3$ extra epochs. All the reported results are the average accuracy across 5 separate runs. In all the experiments, the proposed *two-phase* framework shows a significantly improved validation accuracy compared to the popular linear scaling rule-based large-batch training (*large-batch*). These results demonstrate that the proposed heuristics and the adaptive learning rate decay algorithm effectively improve the generalization performance in large-batch training. In addition, the accuracy is further improved when our framework is applied to *LARS*. That is, our learning rate adjustment improves the generalization performance independently of the layer-wise learning rate scaling method.

In many previous works, ResNet50 is trained on ImageNet for 90 epochs. However, we observed a non-negligible amount of accuracy improvement when the epoch budget was increased to 100.

Table 2: CIFAR-10 classification comparison (ResNet20 training for 300 epochs).

Settings	Batch size	LR	LR decay policy	Decay epochs	LARS coefficient	Validation acc.
small-batch	128	0.1	step-wise ($\beta = 0.1$)	150 / 225	-	$92.83 \pm 0.1\%$
large-batch	$8,192$	6.4	step-wise ($\beta = 0.1$)	150 / 225	-	$90.40 \pm 0.2\%$
LARS		3.5	step-wise ($\beta = 0.1$)	150 / 225	0.01	$92.10 \pm 0.1\%$
two-phase		3.8	adaptive ($\beta = 0.5$)	adaptive (141 ± 13)	-	$92.48 \pm 0.2\%$
two-phase + LARS		3.5	adaptive ($\beta = 0.5$)	adaptive (153 ± 9)	0.01	**92.66** $\pm$ 0.1%

Table 3: CIFAR-100 classification comparison (WideResNet28-10 training for 250 epochs).

Settings	Batch size	LR	LR decay policy	Decay epochs	LARS coefficient	Validation acc.
small-batch	128	0.1	step-wise ($\beta = 0.1$)	125 / 185	-	$81.09 \pm 0.1\%$
large-batch	$8,192$	6.4	step-wise ($\beta = 0.1$)	125 / 185	-	$76.89 \pm 0.2\%$
LARS		6	step-wise ($\beta = 0.1$)	125 / 185	0.01	$78.88 \pm 0.2\%$
two-phase		3	adaptive ($\beta = 0.5$)	adaptive (153 ± 15)	-	$78.90 \pm 0.2\%$
two-phase + LARS		6	adaptive ($\beta = 0.5$)	adaptive (141 ± 19)	0.01	**80.34** $\pm$ 0.2%

Table 4: SVHN classification comparison (WideResNet16-8 training for 160 epochs).

Settings	Batch size	LR	LR decay policy	Decay epochs	LARS coefficient	Validation acc.
small-batch	256	0.01	step-wise ($\beta = 0.1$)	80 / 120	-	$98.55 \pm 0.1\%$
large-batch	$4,096$	0.32	step-wise ($\beta = 0.1$)	80 / 120	-	$98.11 \pm 0.2\%$
LARS		0.32	step-wise ($\beta = 0.1$)	80 / 120	0.001	$98.32 \pm 0.1\%$
two-phase		0.32	adaptive ($\beta = 0.5$)	adaptive (90 ± 13)	-	$98.29 \pm 0.1\%$
two-phase + LARS		0.32	adaptive ($\beta = 0.5$)	adaptive (93 ± 19)	0.001	**98.48** $\pm$ 0.2%

Table 5: ImageNet classification comparison (ResNet50 training for 100 epochs).

Settings	Batch size	LR	LR decay policy	Decay epochs	LARS coefficient	Validation acc.
small-batch	256	0.1	step-wise ($\beta = 0.1$)	30 / 60 / 80	-	$76.79 \pm 0.2\%$
large-batch	$16,384$	6.4	step-wise ($\beta = 0.1$)	30 / 60 / 80	-	$74.91 \pm 0.2\%$
LARS		32	step-wise ($\beta = 0.1$)	30 / 60 / 80	0.001	$76.15 \pm 0.2\%$
two-phase		6.4	adaptive ($\beta = 0.5$)	adaptive (42 ± 4)	-	$76.17 \pm 0.1\%$
two-phase + LARS		32	adaptive ($\beta = 0.5$)	adaptive (39 ± 3)	0.001	**76.67** $\pm$ 0.2%

Within 90 epochs, our algorithm does not sufficiently detect the decay conditions to reduce the learning rate. When we increase the epoch budget to 100, the algorithm decays the learning rate once or twice more, and it results in improving the accuracy from 75.98% to 76.17%. This observation gives us an interesting insight into finding the proper epoch budget for accurate large-batch training. We suggest finding the proper epoch budget that allows the model to sufficiently improve the underlying margin distribution.

Impact of Two-Phase Adjustment on Hessian – Figure 3 shows the principal Hessian Eigenvalue comparisons between *large-batch* and *two-phase*. The Eigenvalues are collected from a) CIFAR-10 training and b) ImageNet training that correspond to Table 2 and 5, respectively. The learning rate for *large-batch* is decayed after 50% and 75% of the epoch budget by a factor of 10, following the reference works (Lin et al. (2020); Goyal et al. (2017)). We see that the longer *exploration* phase and the adaptive decay in *convergence* phase effectively suppress the magnitude of the principal Hessian Eigenvalues. It is known that the narrow Hessian Eigenvalue spectrum represents better generalization performance Keskar et al. (2016).

4.2 Sensitivity on Hyper-Parameters

Length of Exploration Phase – We compare the accuracy across different lengths of *exploration* phase. This experiment verifies the effectiveness of the heuristic discussed in Section 3.1. Table 6 shows the experimental results. The extra epoch column shows the number of epochs after a loss plateau is detected. Note that, we fix the epoch budget to 300 and 250 for CIFAR-10 and CIFAR-100, respectively. Thus, if *exploration* phase is increased too much, it may harm the final accuracy due to the insufficient training epochs under the low noise condition.

When the length of *exploration* phase is increased by 20%, the accuracy is slightly lower than 30% and 40% settings. This implies that the underlying margin has not been sufficiently improved before decaying the learning rate. As expected, 50% setting slightly drops the accuracy due to the

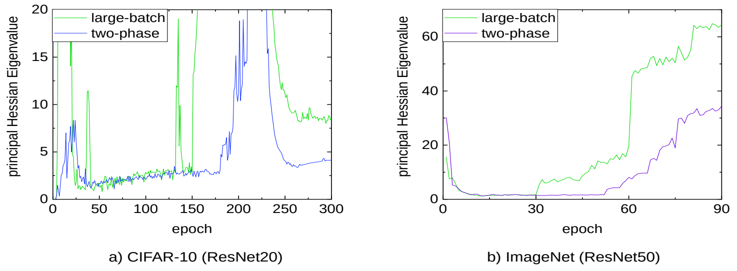

Figure 3: A principle Hessian Eigenvalue comparison between *large-batch* and *two-phase*.

Table 6: CIFAR-10 / CIFAR-100 classification results with different lengths of *exploration* phase.

Extra epochs	Batch size	LR	CIFAR-10	CIFAR-100
20%	8, 192	CIFAR-10: 3.5 CIFAR-100: 6	$92.03 \pm 0.2\%$	$78.45 \pm 0.1\%$
30%			$92.66 \pm 0.1\%$	$80.34 \pm 0.2\%$
40%			$92.54 \pm 0.2\%$	$80.29 \pm 0.2\%$
50%			$92.03 \pm 0.2\%$	$78.84 \pm 0.3\%$
50% (increased epoch budget)			$92.86 \pm 0.1\%$	$80.56 \pm 0.1\%$

Table 7: CIFAR-10 / CIFAR-100 classification results with different β settings.

Decay factor	Batch size	LR	CIFAR-10	CIFAR-100
$\beta = 0.3$	8, 192	CIFAR-10: 3.5 CIFAR-100: 6	$92.31 \pm 0.1\%$	$79.71 \pm 0.1\%$
$\beta = 0.4$			$92.63 \pm 0.1\%$	$80.03 \pm 0.1\%$
$\beta = 0.5$			$92.66 \pm 0.1\%$	$80.34 \pm 0.2\%$
$\beta = 0.6$			$92.18 \pm 0.1\%$	$79.64 \pm 0.1\%$
$\beta = 0.7$			$91.46 \pm 0.1\%$	$77.23 \pm 0.2\%$

insufficient training under the low noise condition. When the epoch budget is increased by 10%, as shown on the bottom row, the loss is well minimized and it achieves remarkably higher accuracy.

Decay Factor β – In Algorithm 1, the learning rate is decayed by a factor of β to enforce the convergence. Table 7 shows the CIFAR-10 and CIFAR-100 accuracy across different β settings. We fix the total number of epochs to 300 and 250 for CIFAR-10 and CIFAR-100, respectively. We see that β around 0.5 generally works well achieving a higher accuracy than the other β settings. Depending on the allowed epoch budget, however, a large β value, such as $\beta \geq 0.6$, can potentially achieve better accuracy while increasing the total training time.

Extra Computational Cost – The Hessian approximation causes an extra computational cost at the end of every epoch. While the extra computational cost depends on the model architecture, we found that the proposed algorithm generally increases the epoch time by $3\% \sim 5\%$. For instance, estimating the h_{max} at the largest output-side layer of ResNet20 takes 0.15 ± 0.1 seconds while the average epoch time is 5.7 ± 0.5 seconds. When measuring the timings, we run 8 processes on four machines each of which has 2 NVIDIA v100 GPUs. The local batch size is 1024 and the power iteration method loops over the same batch 10 times. The timings are averaged across 100 epochs. Overall, the proposed Hessian approximation method does not cause an expensive extra computational cost while effectively obtaining the second order information.

5 CONCLUSION

Considering ever-increasing size of available data and models in this Big Data era, effective scaling of deep learning solutions is critical to tackle large-scale real-world problems. We believe accurate large-batch training techniques can provide researchers with unprecedented opportunities to solve large-scale problems. This paper provides practical insights to deep learning users regarding how to schedule the learning rate to achieve the scalability of the large-batch training together with the high accuracy of the small-batch training. The framework is readily applicable to any deep learning solutions independently of the optimization algorithms, model architectures, and hyper-parameter settings. Harmonizing the proposed framework with other generalization-focused optimization techniques can be a promising future work.

6 Code of Ethics

Our work does not deliver potentially harmful insights or conflicts of interests. We also do not find any potential inappropriate application or privacy/security issues. The datasets we used in our study are all public benchmark datasets, and our source code will be opened once the paper is accepted.

7 Reproducibility Statement

The software versions, implementation details, hyper-parameter settings can be found in the first two paragraphs of Section 4. The data preprocessing details can be found in Appendix A.2. The details about margin distribution approximation method we used is explained in Appendix A.1. The entire source code used in our experiments will be published as an open source once the paper is accepted. We believe one can exactly reproduce our experimental results following the provided descriptions.

References

Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 558–567, 2019.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pp. 1731–1741, 2017.

Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. *arXiv preprint arXiv:1810.00113*, 2018.

Tyler Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. Adascale sgd: A user-friendly algorithm for distributed training. In *International Conference on Machine Learning*, pp. 4911–4920. PMLR, 2020.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, et al. Exascale deep learning for climate analytics. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 649–660. IEEE, 2018.

Yann LeCun, Patrice Simard, and Barak Pearlmutter. Automatic learning rate maximization by online estimation of the hessian's eigenvectors. *Advances in neural information processing systems*, 5:156–163, 1992.

Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems*, pp. 11674–11685, 2019.

Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Extrapolation for large-batch training in deep learning. *arXiv preprint arXiv:2006.05720*, 2020.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Amrita Mathuriya, Deborah Bard, Peter Mendygral, Lawrence Meadows, James Arnemann, Lei Shao, Siyu He, Tuomas Kärnä, Diana Moise, Simon J Pennycook, et al. Cosmoflow: Using deep learning to learn the universe at scale. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 819–829. IEEE, 2018.

Ari S Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. *arXiv preprint arXiv:1806.05759*, 2018.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. *arXiv preprint arXiv:1706.05806*, 2017.

Sihyeon Seong, Yegang Lee, Youngwook Kee, Dongyoon Han, and Junmo Kim. Towards flatter loss surface via nonmonotonic learning rate scheduling. In *UAI*, pp. 1020–1030, 2018.

Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

Wei Wen, Yandan Wang, Feng Yan, Cong Xu, Chunpeng Wu, Yiran Chen, and Hai Li. Smoothout: Smoothing out sharp minima to improve generalization in deep learning. *arXiv preprint arXiv:1805.07898*, 2018.

Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 6, 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

A APPENDIX

A.1 MARGIN DISTRIBUTION ANALYSIS

We analyze how the margin distribution evolves during training to better understand the impact of the length of *exploration* phase on the generalization performance. The decision boundary between two classes i and j is defined as follows.

$$d_{f,x,(i,j)} := \min_{\delta} \|\delta\|_2 \quad s.t. \quad f_i(x+\delta) = f_j(x+\delta) \tag{11}$$

Because the above 'exact' distance for a deep neural network is intractable, we employ an approximation technique proposed in (Jiang et al. (2018)).

$$d_{f,x,(i,j)} = \frac{f_i(x) - f_j(x)}{\|\nabla f_i(x) - \nabla f_j(x)\|_2}, \tag{12}$$

where $f_i(x)$ is the output of the network logit i given the model parameters x, and $\nabla f_i(x)$ is the gradients with respect to the model parameters of logit i. We use the normalized margin $\hat{d}_{f,x,(i,j)}$ that is $d_{f,x,(i,j)}$ divided by the square root of total variation of the input x. Similarly to the reference work (Jiang et al. (2018)), we ignore the margins smaller than the first quartile and larger than the third quartile. We collect the margins across all training samples and compare the accumulated margin to the training loss curve. The margin curves are shown in Figure 1.

A.2 DATA AUGMENTATIONS AND DETAILED SETTINGS

To enable reproduction, we summarize the detailed settings including data preprocessing and augmentation settings.

CIFAR-10 / CIFAR-100 – For each training image, all individual pixels are subtracted by the mean values and divided by the standard deviation values. We mostly follow the hyper-parameter settings used in (Lin et al. (2018; 2020)). The CIFAR-10 training is performed for 300 epochs. The CIFAR-100 training is performed for 250 epochs.

SVHN – SVHN consists of $73K$ training samples, $26K$ validation samples, and $531K$ extra training samples. We use both sets of training samples when training the model and use the validation samples for evaluating the trained model. Likely to CIFAR datasets, each training image is normalized using the pixel-wise statistics. The training is performed for 160 epochs following the settings in (Zagoruyko & Komodakis (2016)). The drop out factor is set to 0.5.

ImageNet – We adopt several data preprocessing methods used in (He et al. (2019)). First, each training sample is normalized in a channel-wise way. Each image is resized to 256×384 pixels keeping the original aspect ratio, and 224×224 image is randomly cropped from the resized image. The cropped image is horizontally flipped with a probability of 0.5. The brightness is randomly adjusted using a maximum delta value of $32/255$. Then, the image color is randomly adjusted in HSV color space using the maximum delta value of 0.1. The saturation is also augmented between 0.6 and 1.4. Finally, the contrast is also augmented between 0.6 and 1.4. We slightly modify ResNet50 model such that the last batch normalization *gamma* parameters are initialized to 0 in all the residual blocks. The bias parameters at the last fully-connected layer are regularized like all the other weight parameters.

The learning rate decay schedule for *small-batch*, *large-batch*, and *LARS* follows the hand-tuned setting used in Goyal et al. (2017). Although many previous works train ResNet50 for 90 epochs only, we observed that the accuracy was still improved after 90 epochs. So, we perform the training for 100 epochs in total.

A.3 LEARNING CURVES

Here, we show the full training loss curves and the validation curves. To clearly show the performance of *two-phase*, we show the learning curves without using LARS.

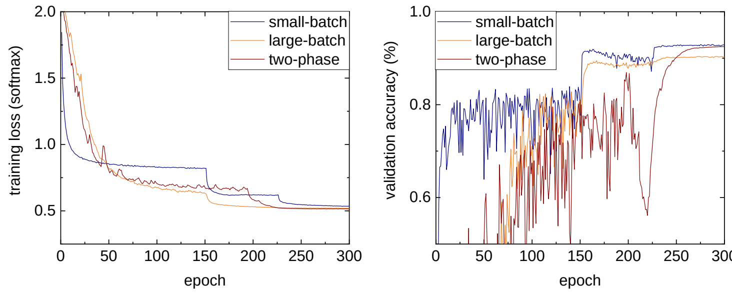

Figure 4: CIFAR-10 (ResNet20) learning curves comparison. The hyper-parameters are shown in Table 2.

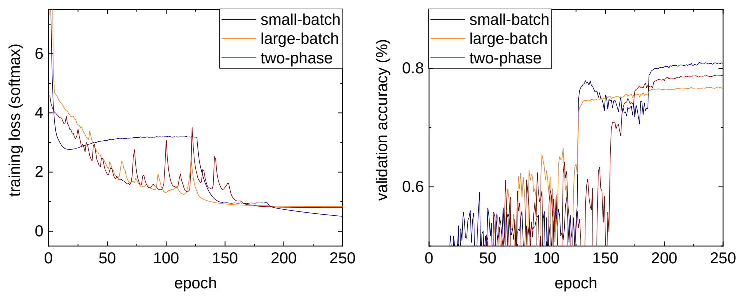

Figure 5: CIFAR-100 (WideResNet28-10) learning curves comparison. The hyper-parameters are shown in Table 3.

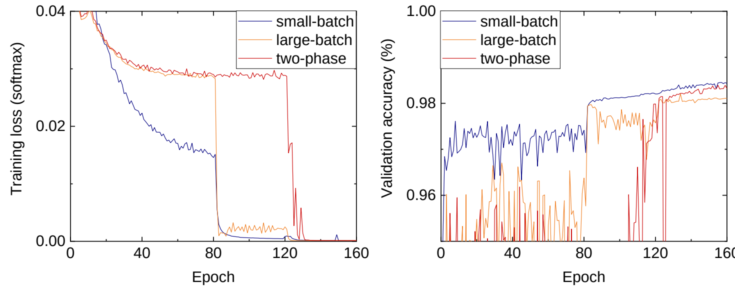

Figure 6: SVHN (WideResNet16-8) learning curves comparison. The hyper-parameters are shown in Table 4.

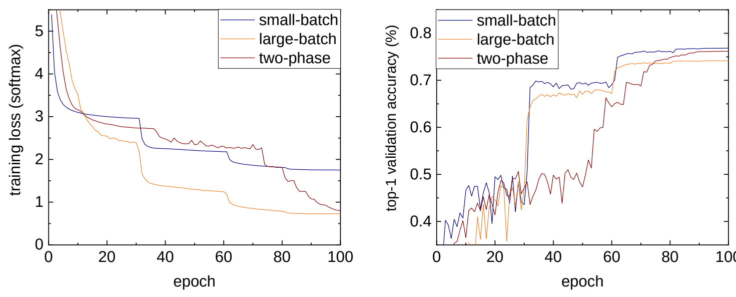

Figure 7: ImageNet (ResNet50) learning curves comparison. The hyper-parameters are shown in Table 5.