Towards Efficient and Scalable Sharpness-Aware Minimization

Yong Liu[1], Siqi Mai[1], Xiangning Chen[2], Cho-Jui Hsieh[2], Yang You[1]
[1]Department of Computer Science, National University of Singapore
[2]Department of Computer Science, University of California, Los Angeles
{liuyong, siqimai, youy}@comp.nus.edu.sg, {xiangning, chohsieh}@cs.ucla.edu

Abstract

Recently, Sharpness-Aware Minimization (SAM), which connects the geometry of the loss landscape and generalization, has demonstrated a significant performance boost on training large-scale models such as vision transformers. However, the update rule of SAM requires two sequential (non-parallelizable) gradient computations at each step, which can double the computational overhead. In this paper, we propose a novel algorithm LookSAM - that only periodically calculates the inner gradient ascent, to significantly reduce the additional training cost of SAM. The empirical results illustrate that LookSAM achieves similar accuracy gains to SAM while being tremendously faster - it enjoys comparable computational complexity with first-order optimizers such as SGD or Adam. To further evaluate the performance and scalability of LookSAM, we incorporate a layer-wise modification and perform experiments in the large-batch training scenario, which is more prone to converge to sharp local minima. Equipped with the proposed algorithms, we are the first to successfully scale up the batch size when training Vision Transformers (ViTs). With a 64k batch size, we are able to train ViTs from scratch in minutes while maintaining competitive performance. The code is available here: https://github.com/yong-6/LookSAM

1. Introduction

It has been observed that sharp local minima usually leads to significantly dropped generalization performance of deep networks, and many methods have been proposed for mitigating this issue [3, 12, 19, 23, 26, 38]. In particular, Foret et al. [13] recently proposed an algorithm named Sharpness Aware Minimization (SAM), which explicitly penalizes the sharp minima and biases the convergence to a flat region. SAM has been used to achieve state-of-the-art performance in many applications. For instance, Chen et al. [4] showed that SAM optimizer can improve the validation accuracy of Vision Transformer models (ViTs) [10] on ImageNet-1k by a significant amount (+5.3% when training from scratch). However, the update rule of SAM involves two sequential (non-parallelizable) gradient computations at each step, which will double the training time.

In this paper, we aim to improve the efficiency of SAM and apply it to large-scale training problems. Each step of SAM consists of two gradient computations – one for adversarial perturbation to the weights and the other for computing the final update. A naive idea to speedup SAM is to compute the first gradient (adversarial perturbation on weights) only periodically and use standard SGD/Adam updates in between. Unfortunately, this leads to significantly degraded performance, as shown in our experiments. To resolve this issue, we decompose the SAM's update direction into two components — the one that lies parallel to the original SGD direction and the other orthogonal component. Since the second direction captures the differences between SAM's update and SGD's update, we hypothesize that this component can bias learning towards a flat region. Interestingly, we show this second direction tends to remain similar across nearby iterations, both empirically and theoretically. Based on this finding, we develop a novel LookSAM optimizer to reuse this direction across nearby iterations. The resulting LookSAM only needs to periodically calculate the inner gradient ascent and significantly reduce the computational complexity of SAM while maintaining similar generalization performance.

As SAM has become a crucial component for training large-scale Vision Transformer models (ViTs) [4], to further evaluate the performance and scalability of the proposed algorithm, we consider a challenging task — applying LookSAM to conduct large-batch training for ViTs. As pointed out in [45, 48], large-batch training often introduces the non-uniform instability problem across different layers. Hence, we also adopt a layer-wise scaling rule for weight perturbation, namely Look-LayerSAM optimizer. The proposed optimizer can successfully train ViTs with 64K batch size within an hour while maintaining competitive performance.

Our contributions can be summarized in three folds.

- We develop a novel algorithm, called LookSAM, to

speed up the training of SAM. Instead of computing the inner gradient ascent at every step, our method only computes it periodically while being able to approximate the original SAM's direction for every update. The empirical results illustrate that LookSAM achieves similar accuracy gains to SAM while enjoying comparable computational complexity with first-order optimizers such as SGD or Adam.

- Inspired by the successes of layer-wise scaling proposed in large-batch training [46, 48], we develop an algorithm to scale up the batch size of LookSAM by adopting layer-wise scaling rule for weight perturbation (Look-LayerSAM). The proposed Look-LayerSAM can scale up the batch size to 64k, which is a new record for ViT training and is $16\times$ compared with previous training settings.
- Our proposed Look-LayerSAM can achieve $\sim \mathbf{8}\times$ **speedup** over the training settings in [10] with a 4k batch size, and we can finish **the ViT-B-16 training in 0.7 hour**. To the best of our knowledge, this is a new speed record for ViT training.

2. Related Work

Sharp Local Minima. Sharp local minima can largely influence the generalization performance of deep networks [3, 12, 19, 23, 26, 38]. Recently, many studies have carefully analyzed the sharp local minima problem and developed algorithms to address such challenges [3, 9, 13, 15, 21, 26, 28, 40, 43]. For example, Jastrzebski et al. [20] state that three factors - learning rate, batch size, and gradient covariance, can influence the minima found by SGD. Besides, Chaudhari et al. [3] propose a local-entropy-based objective function that favors flat regions during training, to avoid approaching the sharp valleys and bad generalization. Wen et al. [41] introduce the *SmoothOut* framework to smooth out sharp minima and thereby improve generalization. More recently, Sharpness-Aware Minimization (SAM) [13] introduce a novel procedure that can simultaneously minimize loss value and loss sharpness to narrow the generalization gap. It presents rigorous empirical results over a variety of benchmark experiments and achieves state-of-the-art performance. Kwon et al. [26] propose adaptive sharpness-aware minimization, which can adaptively adjusts maximization region with respect to weight scale. Zhuang et al. [52] introduce a novel optimization object to simultaneously minimize the perturbed loss and their defined surrogate gap. Du et al. [11] reduces the computational cost of SAM through selecting a set of weights and performing sharpness-aware data selection for updating. These methods still need to calculate two sequential gradients at each step. Therefore, the main focus of this paper is on improving the efficiency and scalability of SAM.

Large-Batch Training. Large-batch training is an important direction for distributed machine learning, which can improve the utilization of large-scale clusters and accelerate the training process. However, training with a large batch size incurs additional challenges [17, 24]. Keskar et al. [24] illustrates that large-batch training is prone to converge to sharp local minima and cause a huge generalization gap. The main reason is that the number of interactions will decrease when scaling up the batch size if we fix the number of epochs. Traditional methods try to carefully tune the hyper-parameters to narrow the generalization gap, such as learning rate, momentum, and label smoothing [14, 29, 37, 47]. However, these heuristic approaches cannot be regarded as a principle solution for large-batch training [37].

Recently, to avoid these hand-tuned methods, adaptive learning rate on large-batch training has gained enormous attention from researchers [35, 36, 51]. Many recent works attempt to use adaptive learning rate to scale the batch size for ResNet-50 on ImageNet [1, 5, 8, 18, 22, 32, 34, 39, 42, 47, 48]. In particular, You et al. [46] proposed layer-wise adaptive learning rate algorithm LARS [46] to scale the batch size to 32k for ResNet-50. Based on LARS optimizer, Ying et al. [44] can finish the ResNet-50 training in 2.2 minutes through TPU v3 Pod [44]. Liu et al. [30] use adversarial learning to further scale the batch size to 96k. In addition, You et al. [48] propose the LAMB optimizer to scale up the batch size when training BERT, resulting in a 76 minutes training time.

3. Method

In this section, we will first give an overview of the SAM optimizer and discuss the computational overhead introduced by SAM. The proposed algorithms, including LookSAM and Layer-wise LookSAM will then be introduced in full detail.

3.1. Overview of SAM

Let $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^n$ be the training dataset, where each sample (x_i, y_i) follows the distribution $\mathcal{D}$. Let $f(x; \boldsymbol{w})$ be the neural network model with trainable parameter $\boldsymbol{w} \in \mathbb{R}^p$. The loss function corresponding to an input x_i is given by $l(f(x_i; \boldsymbol{w}), y_i) \in R^+$, shortened to $l(x_i)$ for convenience. The empirical training loss can be defined as $\mathcal{L}_S = \frac{1}{n}\sum_{i=1}^n l(f(x_i; \boldsymbol{w}), y_i)$. In the SAM algorithm [13], we need to find the parameters whose neighbors within the ℓ_p ball have low training loss $\mathcal{L}_S(\boldsymbol{w})$ through the following modified objective function:

$$\mathcal{L}_S^{SAM}(\boldsymbol{w}) = \max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} \mathcal{L}_S(\boldsymbol{w} + \boldsymbol{\epsilon}), \tag{1}$$

where $p \geq 0$ is the radius of the ℓ_p ball. For simplicity, we will ignore p when using 2-norm. As calculating the

optimal solution of inner maximization is infeasible, SAM uses one-step gradient ascent to approximate it:

$$\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \rho \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w}) / \|\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})\| \approx \arg\max_{\|\boldsymbol{\epsilon}\| \leq \rho} \mathcal{L}_S(\boldsymbol{w}+\boldsymbol{\epsilon}). \tag{2}$$

Finally, SAM computes the gradient with respect to perturbed model $\boldsymbol{w} + \hat{\boldsymbol{\epsilon}}$ for the update:

$$\nabla_{\boldsymbol{w}} \mathcal{L}_S^{SAM}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}}. \tag{3}$$

However, this update rule involves two sequential gradient computations at each step, which will double the computational cost.

3.2. LookSAM

The main drawback of SAM lies in its computational overhead. The update rule (Eq 3) demonstrates that each iteration of SAM needs two sequential gradient computations, one for obtaining $\hat{\boldsymbol{\epsilon}}$ and another for computing the gradient descent update (see Figure 3). This will double the computational complexity compared to SGD or Adam optimizers. Further, these two gradient evaluations are not parallelizable, which will be a bottleneck in large-batch training. However, recent work has demonstrated that SAM yields significant accuracy gain when training vision transformer models [4] (e.g., more than 5% accuracy improvement when training ImageNet from scratch), and further, SAM's ability to escape from sharp minima is valuable in large-batch training. In particular, Keskar et al. [24] showed that the main challenge in large-batch training is the convergence to sharp local minima due to insufficient noise in first-order stochastic updates, and SAM is a natural remedy for this problem if it can be conducted efficiently. These motivate our work on improving SAM's computational efficiency.

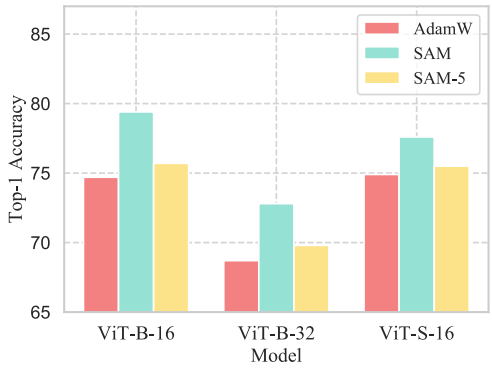

Figure 1. Accuracy of SAM-5, SAM and vanilla ViT on ImageNet-1k. SAM-5 indicates the method that calculating SAM gradients every 5 steps.

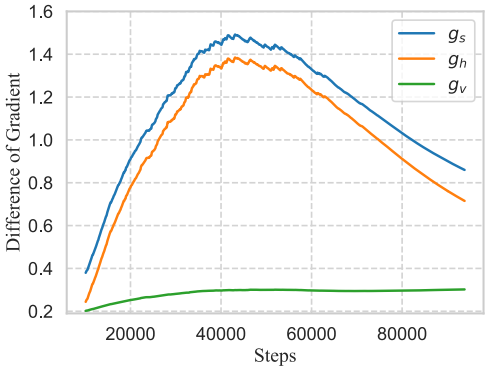

Figure 2. Difference of gradients between every 5 steps for $\boldsymbol{g_s}$, $\boldsymbol{g_h}$, and $\boldsymbol{g_v}$ (i.e., $||g_s^t - g_s^{t+k}||$). $\boldsymbol{g_v}$ that leads to a smoother region changes much slower than $\boldsymbol{g_s}$ and $\boldsymbol{g_h}$.

To reduce the computation of the two sequential gradients in SAM, a naive method is to use SAM update only at every k step, resulting in $\frac{1}{k}\times$ additional calculations on average. We name this method SAM-k, where k indicates the frequency of using SAM. Unfortunately, this naive method does not work well. As shown in Figure 1, we use ViT as the base model and the experimental results illustrate that the accuracy degradation is huge when using SAM-5, although the efficiency is significantly improved. For example, SAM can improve the accuracy from 74.7% to 79.4% for ViT-B-16. However, the accuracy drops to 75.7% when using SAM-5, which significantly degrades the performance of SAM. This motivates us to explore how to effectively improve the efficiency of SAM while maintaining similar generalization performance.

In the following, we propose a novel LookSAM algorithm to address this challenge. The main idea is to study how to reuse information to prevent computing SAM's gradient every time. As shown in Figure 3, the SAM's gradient $\boldsymbol{g_s} = \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}}$ promotes to a flatter region (the blue arrow) compared with the SGD gradient (the yellow arrow). To gain more intuition about this flat region, we rewrite the update of SAM based on Taylor expansion:

$$\begin{aligned}
\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}} &= \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w} + \hat{\boldsymbol{\epsilon}}) \\
&\approx \nabla_{\boldsymbol{w}} [\mathcal{L}_S(\boldsymbol{w}) + \hat{\boldsymbol{\epsilon}} \cdot \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})] \\
&= \nabla_{\boldsymbol{w}} [\mathcal{L}_S(\boldsymbol{w}) + \frac{\rho}{\|\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})\|} \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})^T \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})] \\
&= \nabla_{\boldsymbol{w}} [\mathcal{L}_S(\boldsymbol{w}) + \rho \, \|\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})\|].
\end{aligned} \tag{4}$$

We find that SAM's gradient includes two parts: the original gradient $\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})$ and the gradient of the L2-Norm of original gradient $\|\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})\|$. We think optimizing L2-Norm of gradient can prompt the model converge to flat region as the flat region usually means a low gradient norm

value. Therefore, the update of SAM can be divided into two parts: the first part (denoted as $\boldsymbol{g_h}$) is to decrease the loss value, and the second part (denoted as $\boldsymbol{g_v}$) is to bias the update to a flat region. More specifically, $\boldsymbol{g_h}$ is in the direction of the vanilla SGD's gradient, which needs to be calculated at each step even without SAM. Therefore, the additional computational cost of SAM is mainly induced by the second part $\boldsymbol{g_v}$. Given the SAM's gradient (the red arrow) and the direction of SGD's gradient ($\boldsymbol{g_h}$), we can conduct a projection to obtain $\boldsymbol{g_v}$:

$$\boldsymbol{g_v} = \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}} \cdot \sin(\theta), \tag{5}$$

where θ is the angle between the SGD's gradient and SAM's gradient.

A crucial observation is that $\boldsymbol{g_v}$ **changes much slower than** $\boldsymbol{g_h}$ **and** $\boldsymbol{g_s}$. In Figure 2 we plot the change of these three components between iteration t and iteration $t+5$ throughout the whole training process of SAM, and the results indicate that the difference of $\boldsymbol{g_v}$ (the green line) shows a much more stable pattern than that of $\boldsymbol{g_h}$ (the orange line) and $\boldsymbol{g_s}$ (the blue line). Intuitively, this means the direction pointing to the flat region won't change significantly within a few iterations.

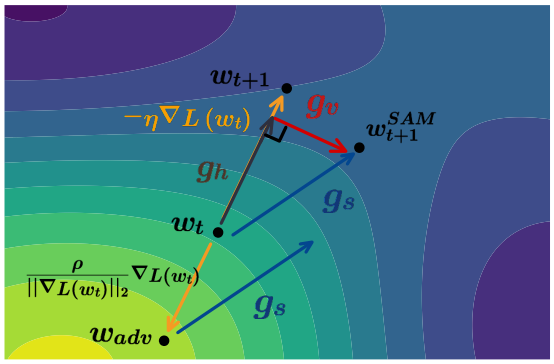

Figure 3. Visualization of LookSAM. The blue arrow $\boldsymbol{g_s}$ is SAM's gradient targeting to a flatter region. The yellow arrow $-\eta \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})$ indicates the SGD gradient. $\boldsymbol{g_h}$ (the brown arrow) and $\boldsymbol{g_v}$ (the red arrow) are the orthogonal gradient components of $\boldsymbol{g_s}$, parallel and vertical to the SGD gradient, respectively.

Therefore, we propose to only calculate the exact SAM's gradient every k steps and reuse the projected gradient $\boldsymbol{g_v}$ for the intermediate steps. The pseudocode is shown in Algorithm 1. We calculate the original SGD gradient $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{w})$ based on the sample minibatch $\mathcal{B}$ at every step. For every k step, we compute SAM's gradient and meanwhile get the projected component $\boldsymbol{g_v}$ (Equation 5) that will be reused for the subsequent steps. At the following k steps, we only calculate the SGD gradient, armed with the projected component to get the approximated SAM gradient. In other words, we train the model and try to mimic

Algorithm 1 LookSAM

Input: $x \in \mathbb{R}^d$, learning rate η_t, update frequency k.
for $t \leftarrow 1$ **to** T **do**
  Sample Minibatch $\mathcal{B} = \{(x_i, y_i), \cdots, (x_{|\mathcal{B}|}, y_{|\mathcal{B}|})\}$ from X.
  Compute gradient $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{w})$ on minibatch $\mathcal{B}$.
  if t%k = 0 **then**
    Compute $\epsilon(\boldsymbol{w}) = \rho \cdot \nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w}) / \|\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})\|$
    Compute SAM gradient: $\boldsymbol{g_s} = \nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\boldsymbol{\epsilon}(\boldsymbol{w})}$
    $\boldsymbol{g_v} = \boldsymbol{g_s} - \|\boldsymbol{g_s}\| \cos(\theta) \cdot \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$, where $\cos(\theta) = \frac{\boldsymbol{g} \cdot \boldsymbol{g_s}}{\|\boldsymbol{g}\| \|\boldsymbol{g_s}\|}$
  else
    $\boldsymbol{g_s} = \boldsymbol{g} + \alpha \cdot \frac{\|\boldsymbol{g}\|}{\|\boldsymbol{g_v}\|} \cdot \boldsymbol{g_v}$
  end if
  Update weights: $\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \eta_t \cdot \boldsymbol{g_s}$
end for

the SAM procedure, by sufficiently distilling the information from SAM gradient every k step. This contributes to the considerable reduction of computation cost, coincident with a smooth convergence that could bias the learning towards a flat region.

To reuse $\boldsymbol{g_v}$ in intermediate steps to mimic the SAM's update, we add $\boldsymbol{g_v}$ to the current gradient $\boldsymbol{g}$ (computed on the clean loss). As the empirical analysis in Figure 2 suggests that $\boldsymbol{g_s}$ and $\boldsymbol{g_h}$ are not very stable, we propose an adaptive ratio to combine them. More specifically, we define $\frac{\|\boldsymbol{g}\|}{\|\boldsymbol{g_v}\|}$ as the adaptive ratio to scale α. In this way, we can ensure that the norms of $\boldsymbol{g}$ and $\boldsymbol{g_v}$ are at the same scale.

3.3. Layer-Wise LookSAM

When scaling up the batch size of SAM or LookSAM in large-batch training, we observe degraded performance as shown in the experiments (see Table 4). You et al. [46, 48] showed that the training stability with large batch training varies for each layer and applied a layer-wise adaptive learning rate scaling method to improve AdamW (also known as LAMB) to resolve this issue. We conjecture this also affects the SAM procedure, which motivates the following development of layer-wise SAM (LayerSAM) optimizer. As we are trying to introduce the layer-wise scaling into the inner maximization of SAM, it is different from [48] which applied the scaling to the final update direction of AdamW. Let $\boldsymbol{\Lambda}$ denote a diagonal $d \times d$ matrix $\boldsymbol{\Lambda} = \text{diag}(\text{concat}(\xi^1 \mathbf{1}_{n(1)}, \xi^2 \mathbf{1}_{n(2)}, ..., \xi^l \mathbf{1}_{n(l)}))$, where d, l represents the number of parameters and layers, $n(l)$ is the number of parameters in layer l. $\xi^j (j = 1, 2, ..., l)$ is the layer-wise adaptive rate and can be calculated by $\frac{\|\boldsymbol{w}^j\|}{\|\nabla_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w})^j\|}$ for each layer, from which w^j refers to the weights of layer j.

We then adopt this scaling into the inner maximization

Table 1. Accuracy of Different Models on CIFAR100. We use ResNet-18, ResNet-50 and WideResNet to evaluate the performance of LookSAM, using SGD-Momentum (SGD-M) as the base optimizer. We set the training epoch as 200 and batch size as 128.

Model	SGD-M	SAM-5	LookSAM-5	SAM-10	LookSAM-10	SAM-20	LookSAM-20	SAM
ResNet-18	78.9	80.4	**80.7**	80.0	**80.4**	79.7	**80.0**	80.7
ResNet-50	81.4	82.5	**83.3**	82.3	**82.8**	82.1	**82.4**	83.3
WRN-28-10	81.7	83.8	**84.4**	83.3	**84.3**	82.9	**83.6**	84.4

of SAM as:

$$\tilde{\mathcal{L}}_{\mathcal{S}}(\boldsymbol{w}) = \max_{\|\boldsymbol{\Lambda}\boldsymbol{\epsilon}\|_p \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w} + \boldsymbol{\epsilon}). \tag{6}$$

Here the main idea is to scale each dimension of the perturbation vector according $\boldsymbol{\Lambda}$. Similar to SAM, the weight perturbation in LayerSAM is the solution of the first-order approximation of (6). With the added $\boldsymbol{\Lambda}$, the approximate inner solution can be written as

$$\tilde{\boldsymbol{\epsilon}} = \rho \, \text{sign}(\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \boldsymbol{\Lambda} \frac{|\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})|^{q-1}}{(\|\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})\|_q^q)^{\frac{1}{p}}}, \tag{7}$$

where $\frac{1}{p} + \frac{1}{q} = 1$. Equation 7 gives us the layer-wise calculation of $\tilde{\boldsymbol{\epsilon}}$ to scale up the batch size when using LookSAM. Algorithm 2 (in Appendix A.1) provides the pseudo-code for the full LayerSAM algorithm. Moreover, to combine the advantages of both LookSAM and LayerSAM in large batch training, we further propose Look Layer-wise SAM (Look-LayerSAM) algorithm. The pseudo-code is given in Algorithm 3. Empirically, we show that Look-LayerSAM significantly outperforms LookSAM in large-batch training, as will be demonstrated in Section 4.

4. Experimental Results

In this section, we evaluate the performance of our proposed LookSAM, LayerSAM, and Look-LayerSAM. First, we empirically illustrate that LookSAM can obtain similar accuracy to vanilla SAM while accelerating the training process. Next, we show that LayerSAM has better generalization for large-batch training on ImageNet-1k compared with vanilla SAM. In addition, we observe Look-LayerSAM can not only scale up to a larger batch size but also significantly speed up the training. As Vision Transformer (ViT) training has become one of the most important applications of SAM [4], our experiments will mainly focus on ViT training, while we also include some experiments of ResNet and WideResNet on CIFAR100 to further evaluate the generality of the proposed methods.

4.1. Setup

Datasets. To evaluate the efficiency of Look-SAM, we conduct the experiments on CIFAR-100 [25] and ImageNet-1k [7] datasets. In addition, ImageNet training is the current benchmark for evaluating the performance of large-batch training [33]. In this paper, we also use ImageNet-1k to train the ViT models.

Models. We firstly use ResNet-18, ResNet-50 [16] and WideResNet [49] to evaluate the performance of LookSAM on CIFAR-100. To explore the scalability of LookSAM, we use ViT [10] models to train ImageNet-1k based on the proposed LookSAM optimizer. Finally, we test the performance of our proposed Look-LayerSAM for large-batch training. More specially, we select the ViT models with various sizes to scale up the batch size, such as ViT-Base and ViT-Small for 300 epochs.

Baselines. Our main baseline is SAM [13]. To better assess the performance of LookSAM, we propose the algorithm SAM-k as the baseline for comparison. More specially, SAM-k can be seen as the method that directly uses SAM every k step.

Implementation Details. We implement our algorithm in JAX [2] and follow the original setting from SAM [13]. To compare the performance of LookSAM with vanilla SAM, we adopt AdamW [31] as the base optimizer. Note that the input resolution is 224, which is the official setting for ViT. To scale up the batch size, we use LAMB [48] as our base optimizer for large-batch training and compare our approaches with SAM. We apply learning rate warmup scheme [14] to avoid the divergence due to the large learning rate, where training starts with a smaller learning rate η and gradually increases to the large learning rate η for 300 epochs. In addition, to further improve the performance of large-batch training, we use RandAug [6] and Mixup [50] to scale the batch size to 64k. The implementation details can be found in Appendix A.2.

4.2. CIFAR Training on ResNet and WideResNet

In this section, we conduct experiments for training ResNet and WideResNet on CIFAR-100 to evaluate the performance of our proposed algorithms. The experimental results are shown in Table 1. We can find that LookSAM-k can achieve a similar accuracy compared with SAM which is much better than SAM-k. As shown in Table 1, LookSAM-5 achieves the same accuracy as SAM did

Table 2. Top-1 accuracy and training time in per epoch (accuracy/time) of ViTs trained from scratch on ImageNet-1k. We use warmup scheme coupled with a cosine scaling rule for 300 epochs. Following the original setting of ViT, we set batch size as 4,096.

Model	AdamW	SAM-5	LookSAM-5	SAM-10	LookSAM-10	SAM
ViT-B-16	74.7/59.7s	75.7/68.6s	**79.8**/70.5s	75.1/63.7s	**78.7**/67.1s	79.8/103.1s
ViT-B-32	68.7/21.8s	69.8/24.7s	**72.6**/26.3s	69.0/23.4s	**71.5**/24.4s	72.8/38.5s
ViT-S-16	74.9/24.1s	75.5/28.3s	**77.6**/30.1s	74.9/25.4s	**77.1**/27.6s	77.6/44.9s
ViT-S-32	68.1/18.2s	68.7/18.5s	**68.8**/19.8s	68.1/18.5s	**68.7**/19.5s	68.9/25.7s

(80.7%, 83.3%, 84.4%) but with much less training time based on the performance of all the three models. Additionally, LookSAM-k shows a remarkable improvement over the performance on SAM-k that likewise takes the comparable training time. Specifically, LookSAM-5 can obtain noticeably higher accuracy (80.7%, 83.3%, 84.4%) compared with SAM-5 (80.4%, 82.5%, 83.8%) on ResNet-18, ResNet-50 and WRN-28-10 respectively. When increasing k, although the performance of LookSAM-k degrades, it still outperforms SAM-k with the same k. For instance, according to the experiment on WRN-28-10, the improvement of LookSAM-k over SAM-k is desirable, with an increment of 0.6%, 1.0% and 0.7% for $k = 5, 10, 20$.

The empirical results in Table 1 also demonstrate that the performance gap between LookSAM and SAM enlarges as model size increases. For example, we can observe an obvious increment of the average improvement of LookSAM-k over SAM-k's when comparing the experiments of ResNet-18 with those of ResNet-50 and WRN-28-10, from 0.37% to 0.53% and 0.77%. Therefore, to further evaluate the performance and scalability of LookSAM, we present the experiment of ImageNet training from scratch on ViT with LookSAM in Section 4.3.

4.3. ImageNet Training from Scratch on Vision Transformer

Following the original setting of ViT, we train ViT with LookSAM and compare it with vanilla ViT and SAM-k. The experimental results are given in Table 2. It shows that LookSAM achieves similar accuracy with vanilla SAM and obtains much better performance than SAM-k. Specifically, compared with the minimal improvement of SAM-k over vanilla AdamW, LookSAM yields considerable improvements, such as the top-1 accuracy improvement from 74.7% to 79.8% on LookSAM-5 (↑ 5.1%), while SAM-5 can only achieve 75.7%. There is a remarkable improvement (↑ 4.1%) of LookSAM-5 in test accuracy (79.8%) in comparison to SAM-5 (75.7%). Further, by computing SAM's update only periodically, our methods significantly improve the time cost over SAM while keeping similar predictive performance. For instance, LookSAM-5 enables a competitive reduction of training time by 2/3 for ViT-B-16 (from 103.1s to 68.6s) without any loss in test accuracy (79.8%). Moreover, this advantage is widely reflected in different settings (shown in Table 2) and thereby our proposed methods can be adopted in a variety of ViT models.

4.4. Large-Batch Training for Vision Transformer

In addition to standard training tasks, we further apply the proposed methods to the challenging large-batch distributed training. It has been observed that large-batch training usually converges to sharp local minima with degraded generalization performance [14, 24]. This is due to insufficient noise in gradient estimation and the decreased number of updates. Therefore, scaling an algorithm to large-batch training is a challenging task.

As mentioned in Section 3.3, we extended LookSAM to Look-LayerSAM to overcome the training instability problem in large-batch training. To evaluate the performance of our proposed algorithms for large-batch training, we use Look-LayerSAM to scale the batch size for ViT training on ImageNet-1k. As shown in Table 4, based on Look-LayerSAM, we can scale the batch size from 4,096 to 32,768 while keeping the accuracy above 77%. Note that although vanilla SAM can improve the performance of ViT while scaling up, the improvement is weakened as batch size increases. For instance, the improvements are 4%, 4%, 3.2%, 2.7% from batch size 4,096 to 32,768 over LAMB (which is a standard optimizer for large batch training). In contrast, our proposed Look-LayerSAM can consistently achieve a higher improvement even if scaling up the batch size to 32,768. In particular, the increments on accuracy are stable from 4,096 to 32,768: 5.6%, 5.8%, 4.4%, and 5.5% over the LAMB optimizer. Moreover, LookSAM is able to achieve the performance on par with the vanilla SAM, while enjoying similar computational cost as LAMB. For example, top-1 accuracy of SAM and LookSAM are 78.6% and 78.9%, respectively, when batch size is 4,096. We continue to observe that Look-LayerSAM offers much more considerable benefits on large batch training, including 80.3% accuracy on 4,096, as well as 77.1% on batch size 32,768, in which SAM and LookSAM achieve 75.1% and 75.3%.

Table 3. Accuracy of ViT-B-16 on ImageNet-1k for 300 epoch when using RandAug and Mixup. Look-LayerSAM can obtain above 75% accuracy when we scale up the batch size to 64k.

Model	Algorithm	RandAug	Mixup	Optimizer	32k	64k
ViT-B-16	**Vanilla ViT**			LAMB	72.4	68.1
ViT-B-16	**Look-LayerSAM**			LAMB	77.1	72.0
ViT-B-16	**Look-LayerSAM**	✓		LAMB	79.2	74.9
ViT-B-16	**Look-LayerSAM**	✓	✓	LAMB	**79.7**	**75.6**

Table 4. Large-batch training accuracy of ViT-B-16 on ImageNet-1k. We use warmup scheme coupled with linear rule to scale the learning rate for 300 epochs. Look-LayerSAM achieves consistent higher accuracy than SAM from 4k to 32k.

Algorithm	4k	8k	16k	32k
LAMB	74.6	74.3	74.4	72.4
LAMB + SAM	78.6	78.3	77.6	75.1
LAMB + Look-SAM	78.9	78.4	77.1	75.3
LAMB + Look-LayerSAM	**80.3**	**79.5**	**78.4**	**77.1**

Table 5. Training Time of ViT-B-16 on ImageNet-1k. We set LAMB as the base optimizer and 300 as the training epoch. We can finish the ViT training within 1 hour.

Algorithm	4k	8k	16k	32k
LAMB	**4.8h**	**2.4h**	**1.2h**	/
LAMB + LayerSAM	8.4h	4.3h	2.2h	1.1h
LAMB + Look-LayerSAM	5.6h	2.8h	1.4h	**0.7h**

In addition, related work has shown that data augmentation can improve the performance of large-batch training. Therefore, we try to further scale the batch size to 64k based on RandAug and Mixup. The experimental results are shown in Table 3, which illustrates that our proposed LayerLookSAM can work together with data augmentation and improve the performance of large-batch training. For instance, Look-LayerSAM can also achieve 74.9% when applying RandAug and Mixup at 64k. After using Mixup, the accuracy improves to 75.6%.

To further evaluate the performance of LookSAM on accelerating the training of SAM, we analyze their training time when scaling batch size from 4,096 to 32,768. Note that we use 128, 256, 512 and 1024 TPU-v3 chips to report the speed of ViT-B-16 on batch size 4,096, 8,192, 16,384, and 32,768. Besides, we use warmup schedule coupled with linear learning rate decay for 300 epochs. The experimental results are shown in Table 5, which illustrates that LayerSAM will cause about $1.7\times$ training time compared with vanilla LAMB. However, Look-LayerSAM can significantly reduce the training time and achieve $1.5\times$ speed compared with LayerSAM when $k = 5$. In particular, the training time of ViT-B-16 on ImageNet-1k can be reduced to 0.7 hour.

To sum up, with Look-LayerSAM, we are able to train Vision Transformer in 0.7 hour and achieve 77.1% top-1 accuracy on ImageNet-1k with 32K batch size, outperforming existing optimizers such as LAMB and SAM.

4.5. Accuracy and Efficiency Tradeoff

The reuse frequency k controls the trade-off between accuracy and speed. In this section, we try to conduct an analysis on the performance of LookSAM with different values of k. The experimental results in Figure 4 indicates that LookSAM can achieve the similar accuracy as vanilla SAM when $k \leq 5$. With reuse frequency k getting larger, the accuracy begins to drop while the training speed is accelerated. For example, as shown in Figure 4, the accuracy of LookSAM-5 on ViT-B-16 is 79.8%, which is the same as the original SAM. In the meantime, the throughput increases from 12,800 (SAM) to 19,051 (LookSAM-5). In addition, when the value k increases to 10, the accuracy drops to 78.7% (improves by 4% compared with AdamW) but the throughput increases to 20,480.

When k is larger than 10, we notice that the speed is converged (almost identical to plain AdamW). Therefore, in practice, we can determine the k value based on the desired trade-off, and we recommend $k = 5$ for general applications since it will significantly improve the efficiency while still achieving almost equivalent test accuracy as SAM. In addition, our proposed LookSAM also provides more selections for deep learning researchers. If the application scenario requires a higher training speed, we can try increasing the frequency k. Otherwise, the frequency k can be reduced.

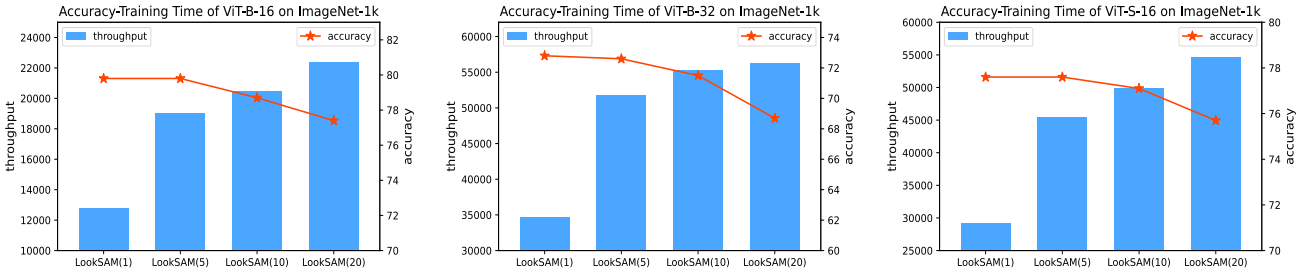

Figure 4. Accuracy-Training Time of different models for LookSAM-k on ImageNet-1k. With the growth of k value, the throughput is increasing but the accuracy starts to drop. There is a trade-off between the accuracy and training speed. Note that LookSAM-1 is the same as the original SAM.

4.6. Sensitivity Analysis about Hyper-Parameters

4.6.1 Sensitivity Analysis of α

We study the effect of gradient reuse weight α has on the performance of training ImageNet-1k. We conduct this experiment with batch size 16,384 and 32,768 since large-batch training is usually more sensitive to hyperparameters. The experiments are conducted on ViT-B-16 using Look-LayerSAM, with LAMB as optimizer, and we set ρ as 1.0. We report the validation accuracy for different α (0.5, 0.7, 1.0) in Table 6. When $\alpha = 0.7$, Look-LayerSAM achieves the best accuracy 78.4% on batch size 16,384 and 77.1% on batch size 32,768. Further, even if α is not well-tuned, Look-LayerSAM is able to obtain a good performance, including above 77% accuracy on 16,384 batch size and $\sim$ 76% accuracy on 32,768 batch size.

Table 6. Sensitivity Analysis of α. We select ViT-B-16 as our base model and the optimizer is Look-LayerSAM (based on LAMB).

Batch Size	α **= 0.5**	α **= 0.7**	α **= 1.0**
16384	77.7	**78.4**	78.2
32768	76.5	**77.1**	75.9

4.6.2 Sensitivity Analysis of ρ

Finally, we conduct a sensitivity analysis for different values of ρ, the intensity of perturbation in SAM and LookSAM. We evaluate the accuracy of ViT-B-16 on batch size 16,384 and 32,768. We set $\alpha = 0.7$, the best value in our analysis from Section 4.6.1. The experimental results regarding ρ (0.5, 0.8, 1.0, 1.2) are shown in Table 7. We report when $\rho = 1.0$, Look-LayerSAM achieves the highest accuracy on both batch size 16,384 (78.4%) and 32,768 (77.1%). Additionally, we observe the overall robustness from the analysis of ρ, which gives us 77% accuracy on 16,384 batch size and more than 75% accuracy on 32,768 batch size without finetuning.

Table 7. Sensitivity Analysis of ρ. We select ViT-B-16 as our base model and the optimizer is Look-LayerSAM (based on LAMB).

Batch Size	ρ **= 0.5**	ρ **= 0.8**	ρ **= 1.0**	ρ **= 1.2**
16384	77.0	77.8	**78.4**	77.9
32768	75.2	76.4	**77.1**	76.7

5. Conclusions

We propose a novel algorithm LookSAM which is able to obtain solutions with similar generalization performance as SAM while having time complexity almost identical to standard stochastic optimizers such as SGD and Adam. The effectiveness and efficiency of LookSAM are verified on multiple datasets and architectures (ViT and ResNet). To further evaluate the performance in large-batch training, we propose Look-LayerSAM, which uses a layer-wise schedule to scale the weight perturbation of LookSAM. By using Look-LayerSAM, we are able to scale the batch size of ViT to 32k and finish the ViT training in 0.7 hour, which is 8$\times$ faster than the original training setting in [10] with a 4k batch size. To the best of our knowledge, this is a new speed record for ViT training.

6. Acknowledgements

We thank Google TFRC for supporting us to get access to the Cloud TPUs. We thank CSCS (Swiss National Supercomputing Centre) for supporting us to get access to the Piz Daint supercomputer. We thank TACC (Texas Advanced Computing Center) for supporting us to get access to the Longhorn supercomputer and the Frontera supercomputer. We thank LuxProvide (Luxembourg national supercomputer HPC organization) for supporting us to get access to the MeluXina supercomputer. CJH and XC are partially supported by NSF under IIS-2008173 and IIS-2048280.

References

[1] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017. 2

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax. 5

[3] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019. 1, 2

[4] Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. When vision transformers outperform resnets without pretraining or strong data augmentations. *arXiv preprint arXiv:2106.01548*, 2021. 1, 3, 5

[5] Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. Scale out for large minibatch sgd: Residual network training on imagenet-1k with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017. 2

[6] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020. 5

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[8] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*, 2017. 2

[9] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. PMLR, 2017. 2

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1, 2, 5, 8

[11] Jiawei Du, Hanshu Yan, Jiashi Feng, Joey Tianyi Zhou, Liangli Zhen, Rick Siow Mong Goh, and Vincent Tan. Efficient sharpness-aware minimization for improved training of neural networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=n0OeTdNRG0Q. 2

[12] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017. 1, 2

[13] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020. 1, 2, 5

[14] 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. 2, 5, 6

[15] Haowei He, Gao Huang, and Yang Yuan. Asymmetric valleys: Beyond sharp and flat local minima. *arXiv preprint arXiv:1902.00744*, 2019. 2

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 5

[17] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv preprint arXiv:1705.08741*, 2017. 2

[18] Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, and Kurt Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 2

[19] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018. 1, 2

[20] Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017. 2

[21] Stanislaw Jastrzebski, Devansh Arpit, Oliver Astrand, Giancarlo B Kerg, Huan Wang, Caiming Xiong, Richard Socher, Kyunghyun Cho, and Krzysztof J Geras. Catastrophic fisher explosion: Early phase fisher matrix impacts generalization. In *International Conference on Machine Learning*, pages 4772–4784. PMLR, 2021. 2

[22] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018. 2

[23] Chi Jin, Lydia T Liu, Rong Ge, and Michael I Jordan. On the local minima of the empirical risk. *arXiv preprint arXiv:1803.09357*, 2018. 1, 2

[24] 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. 2, 3, 6

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[26] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. *arXiv preprint arXiv:2102.11600*, 2021. 1, 2

[27] Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pages 1302–1338, 2000. 2

[28] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017. 2

[29] Mu Li. *Scaling distributed machine learning with system and algorithm co-design*. PhD thesis, PhD thesis, Intel, 2017. 2

[30] Yong Liu, Xiangning Chen, Minhao Cheng, Cho-Jui Hsieh, and Yang You. Concurrent adversarial learning for large-batch training. *arXiv preprint arXiv:2106.00221*, 2021. 2

[31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 5

[32] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*. PMLR, 2015. 2

[33] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500*, 2019. 5

[34] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs. *arXiv preprint arXiv:1811.12019*, 2018. 2

[35] S Reddi, Manzil Zaheer, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *Proceeding of 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018. 2

[36] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019. 2

[37] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018. 2

[38] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*, 2017. 1, 2

[39] 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. 2

[40] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Normalized flat minima: Exploring scale invariant definition of flat minima for neural networks using pac-bayesian analysis. In *International Conference on Machine Learning*, pages 9636–9647. PMLR, 2020. 2

[41] 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. 2

[42] Masafumi Yamazaki, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, and Kohta Nakashima. Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds. *arXiv preprint arXiv:1903.12650*, 2019. 2

[43] Mingyang Yi, Qi Meng, Wei Chen, Zhi-ming Ma, and Tie-Yan Liu. Positively scale-invariant flatness of relu neural networks. *arXiv preprint arXiv:1903.02237*, 2019. 2

[44] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992*, 2018. 2

[45] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017. 1

[46] Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017. 2, 4

[47] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *International Conference on Parallel Processing*, 2018. 2

[48] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019. 1, 2, 4, 5

[49] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 5

[50] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 5

[51] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J Reddi, Sanjiv Kumar, and Suvrit Sra. Why adam beats sgd for attention models. 2019. 2

[52] Juntang Zhuang, Boqing Gong, Liangzhe Yuan, Yin Cui, Hartwig Adam, Nicha Dvornek, Sekhar Tatikonda, James Duncan, and Ting Liu. Surrogate gap minimization improves sharpness-aware training. *arXiv preprint arXiv:2203.08065*, 2022. 2