SHARPNESS-AWARE MINIMIZATION IN LARGE-BATCH TRAINING: TRAINING VISION TRANSFORMER IN MINUTES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-batch training is an important direction for distributed machine learning, which can improve the utilization of large-scale clusters and therefore accelerate the training process. However, recent work illustrates that large-batch training is prone to converge to sharp minima and cause a huge generalization gap. Sharpness-Aware Minimization (SAM) tries to narrow the generalization gap by seeking parameters that lie in a flat region. However, it requires two sequential gradient calculations that doubles the computational overhead. In this paper, we propose a novel algorithm LookSAM to significantly reduce its additional training cost. We further propose a layer-wise modification for adapting LookSAM to the large-batch training setting (Look-LayerSAM). Equipped with our enhanced training algorithm, 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 within an hour while maintaining competitive performance.

1 INTRODUCTION

Data parallelism is one of the most commonly used techniques when training large-scale Deep Neural Network (DNN) models on distributed systems. As the number of processors increases, the batch size of the stochastic optimizers (such as SGD or Adam) enlarges accordingly, leading to **large-batch training**. Due to the reduced variance in each update, Keskar et al. (2016) reported that large-batch training often converges to sharp local minima, which hurts the generalization performance of the resulting DNN models. Methods have been proposed to tackle this problem in the past few years, such as strong data augmentation used in (Kumar et al., 2021; Ying et al., 2018) and a set of new layer-wise adaptive optimizers proposed in You et al. (2017; 2019). However, most of the algorithmic improvements have been made on first order methods such as SGD and Adam.

Recently, Foret et al. (2020) proposed an algorithm named Sharpness Aware Minimization (SAM), which explicitly penalizes the sharp minima and biases the convergence to a flat region. Further, Chen et al. (2021) showed that SAM optimizer can improve the validation accuracy of Vision Transformer models on ImageNet-1k by a significant amount (+5.3% when training from scratch). Despite being able to escape from sharp regions, SAM has not been applied to large-batch training. The main challenge is that the update rule of SAM involves two sequential (non-parallelizable) gradient computations at each step, which will double the training time. Furthermore, although plenty of works studying large-batch training on ResNet models (Goyal et al., 2017; Akiba et al., 2017; Jia et al., 2018), to the best of our knowledge none of previous works conduct on ViTs.

In this paper, we aim to develop an efficient version of SAM optimizer that can be applied in the large-batch training setting. In particular, 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, the proposed LookSAM computes it periodically and reuses the direction that leads to flat regions. 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 (You et al., 2017; 2019), 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 $\sim 8\times$ speedup over the training settings in Dosovitskiy et al. (2020) 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 Background and Related Work

In this section, we will describe recent developments on large batch training, with a review of related work on handling the sharp local minima problem.

2.1 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 (Keskar et al., 2016; Hoffer et al., 2017). Keskar et al. illustrates that large-batch training is prone to converge to sharp local minima and cause a huge generalization gap. Traditional methods try to carefully tune the hyper-parameters to narrow the generalization gap, such as learning rate, momentum, and label smoothing (Goyal et al., 2017; Li, 2017; You et al., 2018; Shallue et al., 2018). Goyal et al. propose warmup to better tune the learning rate for training, which tries to increase the learning rate from a small value at the beginning stage and then start to decrease after increased to the target value. Leveraging the warmup training strategy, Goyal et al. can scale up the batch size to 8,192 for ResNet-50 (He et al., 2016) on ImageNet-1k (Deng et al., 2009). However, these heuristic approaches cannot be regarded as a principle solution for large-batch training (Shallue et al., 2018).

Recently, to avoid these hand-tuned methods, adaptive learning rate on large-batch training has gained enormous attention from researchers (Reddi et al., 2018; 2019; Zhang et al., 2019). Many recent works attempt to use adaptive learning rate to scale the batch size for ResNet-50 on ImageNet (Martens & Grosse, 2015; Iandola et al., 2016; Akiba et al., 2017; Smith et al., 2017; Devarakonda et al., 2017; Codreanu et al., 2017; You et al., 2018; Jia et al., 2018; Osawa et al., 2018; You et al., 2019; Yamazaki et al., 2019). More specially, You et al. proposed layer-wise adaptive learning rate algorithm LARS (You et al., 2017) to scale the batch size to 32k for ResNet-50. Based on LARS optimizer, Ying et al. can finish the ResNet-50 training in 2.2 minutes through TPU v3 Pod (Ying et al., 2018). Liu et al. use adversarial learning to further scale the batch size to 96k. In addition, You et al. (2019) propose the LAMB optimizer to scale up the batch size when training BERT, resulting in a 76 minutes training time.

2.2 Sharp Local Minima

Sharp local minima would largely influence the generalization performance of deep networks (Smith & Le, 2017; Kwon et al., 2021; Dziugaite & Roy, 2017; Chaudhari et al., 2019; Izmailov et al., 2018; Jin et al., 2018). Recently, many studies have attempted to explore the studies of sharp local minima, thus to address the optimization problem (Yi et al., 2019; Tsuzuku et al., 2020; Dinh et al., 2017; Li et al., 2017; Chaudhari et al., 2019; Kwon et al., 2021; He et al., 2019; Foret et al., 2020). For example, Jastrzebski et al. state that three factors - learning rate, batch size and gradient covariance, can influence the minima found by SGD. Besides, Chaudhari et al. propose a local-entropy-based objective function that favors flat regions during training, to avoid approaching the sharp valleys and bad generalization. He et al. observe the loss surfaces and introduce the concept of asymmetric valleys to derive a deeper understanding of flat and sharp minima. By the discovery of Fisher Information Matrix (FIM) as an implicit regularizer of SGD, Jastrzebski et al. try to explicitly penalize the trace of the FIM to solve the problem of sharp minima. Wen et al. introduce the *SmoothOut* framework to smooth out sharp minima and thereby improve generalization. More recently, Sharpness-Aware Minimization (SAM) (Foret et al., 2020) 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. The main focus of this paper is on improving the efficiency and scalability of SAM.

3 PROPOSED 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 details.

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 (Foret et al., 2020), 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:

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

where $p \geq 0$ and ρ is the radius of l^p ball. As calculating the optimal solution of inner maximization is infeasible, SAM uses one-step gradient ascent to approximate it:

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

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, they need to calculate two sequential gradients at each step based on Equation 3, which will double the computational cost for each update.

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. This is also one of the main reasons that SAM hasn't been applied in large-batch training.

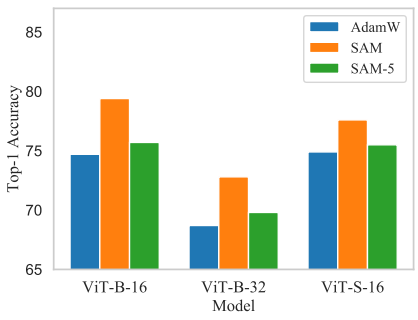


Figure 1: Accuracy of SAM-5, SAM and vanilla ViT on ImageNet-1k. SAM-5 indicates the method that calculating SAM gradients every 5 steps.

However, recent work has demonstrated that SAM yields significant accuracy gain when training vision transformer models (Chen et al., 2021) (e.g., more than 4% 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. (2016) 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 and applying it to large-batch training.

To reduce the computation of the two sequential gradients in SAM, a naive method is to use SAM update only at every k steps, resulting in $\frac{1}{k}\times$ additional calculation 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 maintain its high accuracy.

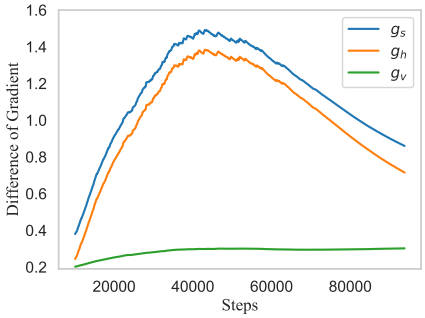


Figure 2: Difference of gradients between every 5 steps for $\boldsymbol{g_s}$, $\boldsymbol{g_h}$, and $\boldsymbol{g_v}$. $\boldsymbol{g_v}$ that leads to a smoother region changes much slower than $\boldsymbol{g_s}$ and $\boldsymbol{g_h}$.

In the following, we propose a novel LookSAM algorithm to address this challenge, where 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 $\boldsymbol{g_s} = \nabla_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}}$ targets to a flatter region (the blue arrow) compared with the SGD gradient (the yellow arrow). 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 on 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 $\boldsymbol{g_v}$:

$$\boldsymbol{g_v} = \nabla_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}} \cdot \sin(\theta), \tag{4}$$

where θ is the angle between the SGD's gradient and SAM's gradient. Empirically, we observe 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 few iterations.

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 steps, we compute SAM's gradient and meanwhile getting the projected component $\boldsymbol{g_v}$ (Equation 4) that will be reused for the subsequent steps. At the following k steps, we only calculate the SGD gradient, armed with projected component to get the approximated SAM gradient. In other words, we train the model and try to mimic the SAM procedure, by sufficiently distilling the information from SAM gradient every k steps. This contributes to the considerable reduction of computation cost, coincident with a smooth convergence that could bias the learning towards a flat region.

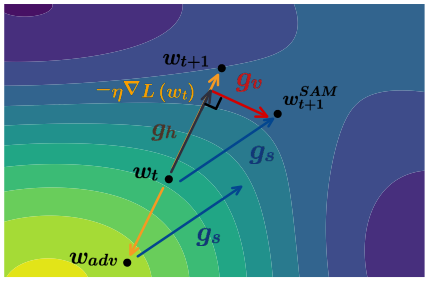


Figure 3: Visualization of LookSAM.

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 a 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.

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 $\epsilon(\boldsymbol{w}) = \rho \cdot \text{sign}(\boldsymbol{g})$
    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

To demonstrate the reusing procedure, we theoretically derive the change of $\boldsymbol{g_v}$ within k steps in the following way (full derivation is shown in AppendixA.5),

$$\begin{aligned} \|\boldsymbol{g_{v,t}} - \boldsymbol{g_{v,t+k}}\| \approx \| & \frac{1}{2}\rho \bigtriangledown^2_{\boldsymbol{w_t}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_t}) \frac{\bigtriangledown_{\boldsymbol{w_t}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_t})}{\| \bigtriangledown_{\boldsymbol{w_t}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_t})\|} \\ & - \frac{1}{2}\rho \bigtriangledown^2_{\boldsymbol{w_{t+k}}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_{t+k}}) \frac{\bigtriangledown_{\boldsymbol{w_{t+k}}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_{t+k}})}{\| \bigtriangledown_{\boldsymbol{w_{t+k}}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_{t+k}})\|} \|, \end{aligned} \tag{5}$$

where we ignore the third order terms in Taylor expansion in the derivation. As calculating SAM gradients leads to a relatively flat part of the region, the second order derivative is very small, from which we can infer the change of $\boldsymbol{g_v}$ component is small compared to the gradient. This supports our algorithm in reusing $\boldsymbol{g_v}$ and re-caculating it only periodically.

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 2). You et al. (2017; 2019) 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 You et al. (2019) which applied the scaling to the final update direction of AdamW.

Let $\boldsymbol{\Lambda}$ denote a diagonal $l \times l$ matrix $\boldsymbol{\Lambda} = \text{diag}(\xi^1, \xi^2, ..., \xi^l)$, where ξ^j(j=1,2,...,l) is the layer-wise adaptive rate and can be calculated by $\frac{\|\boldsymbol{w}^j\|}{\|\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})^j\|}$ for each layer. We then adopt this scaling into the inner maximization 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 equation 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 3 (in Appendix A.2) provides us the pseudo-code for the full LayerSAM. Moreover, in order to combine 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 2. Empirically, we show that Layer-LookSAM significantly outperforms LookSAM in large-batch training, as will be demonstrated in Section 4.

Algorithm 2 Look-LayerSAM

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 $\boldsymbol{g} = \nabla_{\boldsymbol{w}} L_B(\boldsymbol{w})$ on minibatch $\mathcal{B}$.
  if t%k = 0 **then**
    Compute $\epsilon(\boldsymbol{w})^{(i)} = \rho \frac{\|\boldsymbol{w}^{(i)}\|}{\|\boldsymbol{g}^{(i)}\|} \cdot \text{sign}(\boldsymbol{g})$
    Compute SAM gradient: $\boldsymbol{g_s} = \nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\epsilon(\boldsymbol{w})}$
    $\boldsymbol{g_v} = \boldsymbol{g_s} - \|\boldsymbol{g_s}\| \cos(\theta) \cdot \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$, 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: $\boldsymbol{w_{t+1}^{(i)}} = \boldsymbol{w_t^{(i)}} - \eta_t^{(i)} \cdot \boldsymbol{g_s^{(i)}}$
end for

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 accelerate 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 larger batch size but also significantly speed up the training. As Vision Transformer (ViT) training has become one of the most important application of SAM (Chen et al., 2021), our experiments will mainly focus on training the ViT model, while we also include some experiments of ResNet and WideResNet on CIFAR10 and CIFAR100 in Appendix A.3 to further evaluate the generality of the proposed methods.

4.1 Setup

Dataset. ImageNet training is the current benchmark for evaluating the performance of large-batch training. In this paper, we use ImageNet-1k with 1.3M images (Deng et al., 2009) to train the ViT models.

Models. To explore the scalability of SAM for ViT models (Dosovitskiy et al., 2020), we select the ViT models with various size to scale up batch size, such as ViT-Base and ViT-Small.

Baselines. Our main baseline is SAM (Foret et al., 2020). 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 use SAM every k steps.

Implementation Details. We implement our algorithm in JAX (Bradbury et al., 2018) and follow the original setting from SAM (Foret et al., 2020). To compare the performance of LookSAM with vanilla SAM, we conduct AdamW (Loshchilov & Hutter, 2017) as the optimizer. Note that the input resolution is 224, which is the official setting for ViT. To scale up the batch size, we use LAMB (You et al., 2019) as our base optimizer for large-batch training and compare our approaches with SAM. We apply learning rate warmup scheme (Goyal et al., 2017) 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 (Cubuk et al., 2020) and Mixup (Zhang et al., 2017) to scale the batch size to 64k. The implementation details can be found in Appendix A.4.

4.2 ImageNet Training from Scratch on Vision Transformer

Following the original setting of ViT, we firstly train ViT with LookSAM and compare it with vanilla ViT and SAM-k. The experimental results are given in Table 1. 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, while SAM-5 can only achieve 75.7%. In addition, our proposed LookSAM algorithm can achieve faster training speed and maintain the similar accuracy compared with SAM. Further, by computing SAM's update only periodically, our method 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 1) and thereby our proposed methods can be adopted in a variety of ViT models.

Table 1: 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 4096.

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

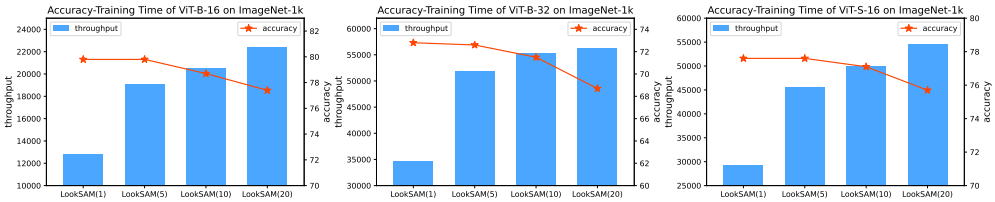
4.3 Large-Batch Training for Vision Transformer

To further 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 2, based on Look-LayerSAM, we can scale the batch size from 4096 to 32768 while keep 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 4096 to 32768 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 32768. In particular, the increment on accuracy are stable from 4096 to 32768: 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 4096. We continue to observe that Look-LayerSAM offer much more considerable benefits on large batch training, including 80.3% accuracy on 4096, as well as 77.1% on batch size 32768, in which SAM and LookSAM achieve 75.1% and 75.3%.

Table 2: 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**

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 Layer-LookSAM 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%.



(a) ViT-B-16 (b) ViT-B-32 (c) ViT-S-16

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.

Table 3: Accuracy of ViT-B-16 on ImageNet-1k when using RandAug and Mixup

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

To further evaluate the performance of LookSAM about accelerating the training of SAM, we analyze their training time when scaling batch size from 4096 to 32768. Note that we use TPU v3 128 chips, TPU v3 256 chips, TPU v3 512 chips and TPU v3 1024 chips to report the speed of ViT-B-16 on batch size 4096, 8192, 16384 and 32768. Besides, we use warmup schedule coupled with linear learning rate decay for 300 epochs. The experimental results are shown in Table 4, which illustrates that LayerSAM will cause about $1.7\times$ training time compared with vanilla ViT. However, Look-LayerSAM can significantly reduce the training time and achieve $1.5\times$ speed compared with LayerSAM when $k = 5$. In particular, 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.

Table 4: Training Time of ViT-B-16 on ImageNet-1k

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**

4.4 Accuracy and Efficiency Tradeoff

The reuse frequency k controls the trade-off between the 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 begin to drop while the training speed is accelerated. When k is larger than 15, we notice that the speed is converged (almost identical to plain AdamW training). 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 achieve almost identical test accuracy as SAM.

4.5 SENSITIVITY ANALYSIS ABOUT HYPER-PARAMETERS

In this section, we will analyse the sensitivity of our proposed algorithms for hyper-parameters, including the intensity of perturbation ρ and gradient reuse weight α.

4.5.1 SENSITIVITY ANALYSIS OF α

We study the effect of gradient reuse weight α has on the performance of training ImageNet-1k. As for batch size, we select 16384 and 32768 to analyze. That is because larger batch size is always more sensitive to the hyperparameters. The experiments are conducted on ViT-B-16 using Look-LayerSAM, with LAMB as optimizer. In the experiment, we set ρ as 1.0. We report the validation accuracy for different α (0.5,0.7,1.0) in Table 5. When $\alpha = 0.7$, Look-LayerSAM achieves the best accuracy 78.4% on batch size 16384 and 77.1% on batch size 32768. Further, even if α is not well tuned, Look-LayerSAM is able to obtain a good performance, including above 77% accuracy on 16384 and approx. 76% on 32768.

Table 5: Sensitivity Analysis of α

Model	**Method**	**Optimizer**	**Batch Size**	α **= 0.5**	α **= 0.7**	α **= 1.0**
ViT-B-16	Look-LayerSAM	LAMB	16384	77.7	**78.4**	78.2
ViT-B-16	Look-LayerSAM	LAMB	32768	76.5	**77.1**	75.9

4.5.2 SENSITIVITY ANALYSIS OF ρ

Finally, we also have a sensitivity analysis for different value of ρ, the intensity of perturbation on Look-LayerSAM algorithm. We evaluate the accuracy of ViT-B-16 on batch size 16384 and 32768. We set $\alpha = 0.7$, the best value in our analysis from Section 4.5.1. The experimental results regarding ρ (0.5,0.8,1.0,1.2) are shown in Table 6. We report when $\rho = 1.0$, Look-LayerSAM achieves the highest accuracy on both batch size 16384 (78.4%) and 32768 (77.1%). Additionally, we observe the overall robustness from the analysis of ρ, which gives us 77% accuracy on 16384 and more than 75% accuracy on 32768 without finetuning.

Table 6: Sensitivity Analysis of ρ

Model	**Method**	**Optimizer**	**Batch Size**	ρ **= 0.5**	ρ **= 0.8**	ρ **= 1.0**	ρ **= 1.2**
ViT-B-16	Look-LayerSAM	LAMB	16384	77.0	77.8	**78.4**	77.9
ViT-B-16	Look-LayerSAM	LAMB	32768	75.2	76.4	**77.1**	76.7

5 CONCLUSION

We propose a novel algorithm LookSAM to reduce the additional computation from SAM and speed up the training process. To further evaluate the performance in large-batch training, we propose Look-LayerSAM, which use a layer-wise schedule to scale the weight perturbation of LookSAM. Finally, we evaluate our proposed algorithms on Vision Transformer. Experimental results illustrate that we can scale the batch size to 64k and obtain the accuracy above 75%. Further, we can achieve about $8\times$ speedup over the training settings in Dosovitskiy et al. (2020) with a 4k batch size and finish the ViT training in 0.7 hour. To the best of our knowledge, this is a new speed record for ViT training.

6 Ethics Statement

We are not aware of any potential ethical issues in our work.

7 Reproducibility Statement

We provide detailed experimental setting of all experiments in Section 4 and Appendix to ensure the reproducibility of this paper. More specially, all the model architecture can be found in Appendix A.4 (Table 9). As for the hyperparameters of ViTs, we also introduce them in Appendix A.4 (Table 10 and Table 11), which includes the learning rate, optimizer, weight decay, perturbation value ρ and so on.

References

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.

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.

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.

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.

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.

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*, pp. 702–703, 2020.

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*, pp. 248–255. Ieee, 2009.

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

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

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.

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.

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.

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

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.

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.

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.

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.

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.

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*, pp. 4772–4784. PMLR, 2021.

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.

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.

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.

Sameer Kumar, Yu Wang, Cliff Young, James Bradbury, Naveen Kumar, Dehao Chen, and Andy Swing. Exploring the limits of concurrency in ml training on google tpus. *Machine Learning and Systems*, 2021.

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.

Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pp. 1302–1338, 2000.

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

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

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.

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

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

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.

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.

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

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.

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

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.

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*, pp. 9636–9647. PMLR, 2020.

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.

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.

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.

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

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

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

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.

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

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.

A APPENDIX

A.1 THEORETICAL ANALYSIS OF PROJECTED GRADIENT

For SAM loss function $\mathcal{L}_S(\boldsymbol{w}+\hat{\boldsymbol{\epsilon}})$, based on Taylor Expansion, we can obtain:

$$\mathcal{L}_S(\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}) \approx \mathcal{L}_S(\boldsymbol{w}) + \hat{\boldsymbol{\epsilon}}\nabla_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) \tag{8}$$

Therefore, we can rewrite Equation 3 as follows:

$$\begin{aligned}\nabla_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}} &= \nabla_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}}\mathcal{L}_S(\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}) \\ &\approx \nabla_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}}[\mathcal{L}_S(\boldsymbol{w}) + \hat{\boldsymbol{\epsilon}}\nabla_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w})] \\ &= \nabla_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) + \hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w})\end{aligned} \tag{9}$$

In this section, we will analyse the distance of $\boldsymbol{g_v}$ within several steps k, which is given by,

$$\boldsymbol{g_v} = \hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \tag{10}$$

Note that $\boldsymbol{g_v}$ is vertical to the SGD gradient on original weight $\boldsymbol{w}$, which gives us the following constraint:

$$(\hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) = 0 \tag{11}$$

We can use Lagrange function to describe and solve this problem. Let's introduce a new variable λ and define the Lagrangian L with all the parameters $\boldsymbol{w}$, λ_0 and λ as variable as follows,

$$\begin{aligned}L(\boldsymbol{w},\lambda_0,\lambda) =& \hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \\ &+ \lambda(\hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})\end{aligned} \tag{12}$$

The partial derivative of L with respect to $\boldsymbol{w}$, λ_0 and λ are as follows,

$$\begin{aligned}L_{\boldsymbol{w}}(\boldsymbol{w},\lambda_0,\lambda) =& \frac{\partial\hat{\boldsymbol{\epsilon}}}{\partial\boldsymbol{w}} \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) + \hat{\boldsymbol{\epsilon}} \bigtriangledown^3_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) - \lambda_0 \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \\ &+ \lambda(\frac{\partial\hat{\boldsymbol{\epsilon}}}{\partial\boldsymbol{w}} \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) + \hat{\boldsymbol{\epsilon}} \bigtriangledown^3_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) - \lambda_0 \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \\ &+ \lambda(\hat{\boldsymbol{\epsilon}} \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})\end{aligned} \tag{13}$$

$$L_{\lambda_0}(\boldsymbol{w},\lambda_0,\lambda) = - \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \tag{14}$$

$$L_{\lambda}(\boldsymbol{w},\lambda_0,\lambda) = (\hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \tag{15}$$

we omit some high order terms that are close to 0 in the calculation and set all of the partial derivatives equal to 0. This gives,

$$\begin{aligned}&L_{\boldsymbol{w}}(\boldsymbol{w},\lambda_0,\lambda) \approx \frac{\partial\hat{\boldsymbol{\epsilon}}}{\partial\boldsymbol{w}} \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) - \lambda_0 \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) = 0 \\ &L_{\lambda_0}(\boldsymbol{w},\lambda_0,\lambda) = - \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) = 0 \\ &L_{\lambda}(\boldsymbol{w},\lambda_0,\lambda) = (\hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) = 0\end{aligned} \tag{16}$$

Since $\frac{\partial\hat{\boldsymbol{\epsilon}}}{\partial\boldsymbol{w}} = \rho(\frac{\nabla^2_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{||\nabla_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} - \frac{\nabla_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})\nabla^2_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})\nabla_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{||\nabla_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||^3})$ and from Equation 16 we have $\frac{\partial\hat{\boldsymbol{\epsilon}}}{\partial\boldsymbol{w}} = \lambda_0$, then

$$\lambda_0 = \rho(\frac{\bigtriangledown^2_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{||\bigtriangledown_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} - \frac{\bigtriangledown_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{||\bigtriangledown_{\boldsymbol{w}}\mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||^3}) \tag{17}$$

In addition, from Equation 16, we have,

$$(\hat{\boldsymbol{\epsilon}}\nabla^2_{\boldsymbol{w}}\mathcal{L}_S(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) = 0 \tag{18}$$

Then λ_0 can be written as:

$$\lambda_0 = \rho \frac{\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||^3} \tag{19}$$

With Equation 14 and 19,

$$\rho(\frac{\bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} - \frac{\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||^3}) = \rho \frac{\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||^3}$$

$$\frac{1}{2}\rho \frac{\bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} = \rho \frac{\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||^3} \tag{20}$$

Using the relationship in Equation 20, we can write λ_0 in this way:

$$\lambda_0 = \frac{1}{2}\rho \frac{\bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} \tag{21}$$

Substituting the value of λ_0 back to Equation 12 gives us the maximum of the Lagrangian.

$$\begin{aligned} \tilde{L}(\boldsymbol{w}, \lambda_0, \lambda) &= \hat{\boldsymbol{\epsilon}} \nabla^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) - \lambda_0 \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \\ &= \rho \frac{\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) - \frac{1}{2}\rho \frac{\bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \end{aligned} \tag{22}$$

Using this relationship in Equation 22 gives us:

$$\boldsymbol{g_v} = \tilde{L}(\boldsymbol{w}, \lambda_0, \lambda) = \frac{1}{2}\rho \bigtriangledown^2_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}) \frac{\bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})}{|| \bigtriangledown_{\boldsymbol{w}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w})||} \tag{23}$$

We can use Equation 23 to derive the distance of $\boldsymbol{g_v}$ within k steps,

$$\begin{aligned} ||\boldsymbol{g_{v,t}} - \boldsymbol{g_{v,t+k}}|| &= ||\tilde{L}_t(\boldsymbol{w_t}, \lambda_0, \lambda) - \tilde{L}_{t+k}(\boldsymbol{w_{t+k}}, \lambda_0, \lambda)|| \\ &\approx ||\frac{1}{2}\rho \bigtriangledown^2_{\boldsymbol{w_t}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_t}) \frac{\bigtriangledown_{\boldsymbol{w_t}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_t})}{|| \bigtriangledown_{\boldsymbol{w_t}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_t})||} \\ &\quad - \frac{1}{2}\rho \bigtriangledown^2_{\boldsymbol{w_{t+k}}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_{t+k}}) \frac{\bigtriangledown_{\boldsymbol{w_{t+k}}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_{t+k}})}{|| \bigtriangledown_{\boldsymbol{w_{t+k}}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w_{t+k}})||}|| \end{aligned} \tag{24}$$

A.2 LAYERSAM

Algorithm 3 Layer-wise SAM (LayerSAM)

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 $g = \nabla_{\boldsymbol{w}} L_B(\boldsymbol{w})$ on minibatch $\mathcal{B}$.
  Compute $\boldsymbol{\epsilon^{(i)}} = \rho \frac{||\boldsymbol{w}^{(i)}||}{||\boldsymbol{g}^{(i)}||} \cdot \text{sign}(\boldsymbol{g})$
  Compute gradient approximation for the SAM objective: $\boldsymbol{g_s} = \nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\boldsymbol{\epsilon}}$
  Update weights: $\boldsymbol{w^{(i)}_{t+1}} = \boldsymbol{w^{(i)}_t} - \eta^{(i)}_t \cdot \boldsymbol{g^{(i)}_s}$
end for

A.3 RESNET AND WIDERESNET

In this section, we try to conduct some experiments for training ResNet and WideResNet on CIFAR-10 and CIFAR-100 to evaluate the performance of our proposed algorithms. The experimental results are shown in Table 7 and 8. We can find that LookSAM can obtain a better accuracy than SAM-5 and meanwhile achieve a similar accuracy compared with SAM.

Table 7: Accuracy of different Models on CIFAR10

Model	AdamW	SAM-5	SAM-10	SAM-20	LookSAM-5	LookSAM-10	LookSAM-20	SAM
WRN-28-10	96.5	97.2	97.0	96.8	97.3	97.1	97.0	97.3
ResNet-18	95.6	96.2	96.0	96.0	96.2	96.1	96.1	96.4
ResNet-50	95.7	96.6	96.5	96.2	96.8	96.6	96.4	96.9

Table 8: Accuracy of Different Models on CIFAR100

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

A.4 PARAMETER SETTINGS

In this section, we will introduce the architectures of ViTs in this paper (Table 9). Next, we provide the hyperparameters in Table 10 for ViT training, including learning rate, warmup, optimizer, gradient clipping, epoch, etc. In addition, Table 11 gives us the parameter settings of ViT for large-batch training in this paper.

Table 9: Architectures of ViTs

Model	Params	Patch Resolution	Sequence Length	Hidden Size	Heads	Layers
ViT-B-16	87M	16×16	196	768	12	12
ViT-B-32	88M	32×32	49	768	12	12
ViT-S-16	22M	16×16	196	384	6	12
ViT-S-32	23M	32×32	49	384	6	12

Table 10: Parameter Settings of ViT for Vanilla Training

Model	Input Resolution	Batch Size	Epoch	Warmup Steps	Peak LR	LR Decay	Optimizer	ρ	Weight Decay	Gradient Clipping
ViT-B-16	224	4096	200	10000	3e-3	cosine	AdamW	/	0.3	1.0
ViT-B-32	224	4096	200	10000	3e-3	cosine	AdamW	/	0.3	1.0
ViT-S-16	224	4096	200	10000	3e-3	cosine	AdamW	/	0.3	1.0
ViT-S-32	224	4096	200	10000	3e-3	cosine	AdamW	/	0.3	1.0
ViT-B-16 + SAM	224	4096	200	10000	3e-3	cosine	AdamW	0.18	0.3	1.0
ViT-B-32 + SAM	224	4096	200	10000	3e-3	cosine	AdamW	0.15	0.3	1.0
ViT-S-16 + SAM	224	4096	200	10000	3e-3	cosine	AdamW	0.1	0.3	1.0
ViT-S-32 + SAM	224	4096	200	10000	3e-3	cosine	AdamW	0.05	0.3	1.0
ViT-B-16 + LookSAM	224	4096	200	10000	3e-3	cosine	AdamW	0.18	0.3	1.0
ViT-B-32 + LookSAM	224	4096	200	10000	3e-3	cosine	AdamW	0.15	0.3	1.0
ViT-S-16 + LookSAM	224	4096	200	10000	3e-3	cosine	AdamW	0.1	0.3	1.0
ViT-S-32 + LookSAM	224	4096	200	10000	3e-3	cosine	AdamW	0.05	0.3	1.0

Table 11: Parameter Settings of ViT for Large-Batch Training

Model	Batch Size	Epoch	Warmup Steps	Peak LR	LR Decay	Optimizer	ρ	α	Weight Decay	Gradient Clipping
ViT-B-16 + SAM	4096	200	10000	1e-2	linear	LAMB	0.18	/	0.1	1.0
ViT-B-16 + SAM	8192	200	10000	1.7e-2	linear	LAMB	0.18	/	0.1	1.0
ViT-B-16 + SAM	16834	200	7000	1.8e-2	linear	LAMB	0.18	/	0.1	1.0
ViT-B-16 + SAM	32768	200	6000	1.8e-2	linear	LAMB	0.18	/	0.1	1.0
ViT-B-16 + LayerSAM	4096	200	10000	1e-2	linear	LAMB	1.0	/	0.1	1.0
ViT-B-16 + LayerSAM	8192	200	10000	1.7e-2	linear	LAMB	1.0	/	0.1	1.0
ViT-B-16 + LayerSAM	16384	200	7000	1.8e-2	linear	LAMB	1.0	/	0.1	1.0
ViT-B-16 + LayerSAM	32768	200	6000	1.8e-2	linear	LAMB	1.0	/	0.1	1.0
ViT-B-16 + LayerSAM	65536	200	3500	2e-2	linear	LAMB	1.0	/	0.2	1.0
ViT-B-16 + Look-LayerSAM	4096	200	10000	1e-2	linear	LAMB	1.0	0.7	0.1	1.0
ViT-B-16 + Look-LayerSAM	8192	200	10000	1.7e-2	linear	LAMB	1.0	0.7	0.1	1.0
ViT-B-16 + Look-LayerSAM	16384	200	7000	1.8e-2	linear	LAMB	1.0	0.7	0.1	1.0
ViT-B-16 + Look-LayerSAM	32768	200	6000	1.8e-2	linear	LAMB	1.0	0.7	0.1	1.0
ViT-B-16 + Look-LayerSAM	65536	200	3500	2e-2	linear	LAMB	1.0	0.7	0.2	1.0

A.5 GENERALIZATION BOUND

We firstly introduce Theorem 1 regarding generalization bound based on sharpness of LookSAM and then give a proof for it. Note that a similar bound was also established in the original SAM paper Foret et al. (2020).

Theorem 1. *With probability 1 - δ over the choice the training set $\mathcal{S} \sim \mathcal{D}$, we have*

$$\mathcal{L}_D(\boldsymbol{w}) \leq \max_{||\boldsymbol{\epsilon'}||_p \leq \rho'} \mathcal{L}_S(\boldsymbol{w} + \boldsymbol{\epsilon'}) + \sqrt{\frac{k \log(1 + \frac{||\boldsymbol{w}||_2^2}{\rho'^2}(1 + \sqrt{\frac{\log(n)}{k}})^2) + 4 \log \frac{n}{\delta} + \tilde{O}(1)}{n-1}} \tag{25}$$

where $n = |\mathcal{S}|$ and $\rho'^2 = \rho^2 + \rho_0^2$.

Proof. We start by illustrating the PAC-Bayesian Generation Bound theorem, which gives a bound on the generalization error of any posterior distribution $\mathcal{Q}$ on parameters that can be achieved using

a selected prior distribution $\mathcal{P}$ over parameters training with data set $\mathcal{S}$. Let $KL(\mathcal{Q}||\mathcal{P})$ denote the KL divergence between two Bernoulli distributions $\mathcal{P}$ and $\mathcal{Q}$, we have:

$$\mathbb{E}_{\boldsymbol{w}\sim\mathcal{L}}[L_{\mathcal{D}}(\boldsymbol{w})] \leq \mathbb{E}_{\boldsymbol{w}\sim\mathcal{L}}[L_{\mathcal{S}}(\boldsymbol{w})] + \sqrt{\frac{KL(\mathcal{Q}||\mathcal{P}) + \log\frac{n}{\delta}}{2(n-1)}} \tag{26}$$

In order to accelerate the training process, LookSAM calculate the SAM gradient only at every k step and try to reuse the projected components to imitate the weight perturbations introduced from SAM procedure in the subsequent steps. We use $\boldsymbol{\epsilon^0}$ to indicate the difference between our imitated weight perturbation, $\boldsymbol{\epsilon'}$, from LookSAM and the real weight perturbation, $\boldsymbol{\epsilon}$, from SAM. As the optimization is in fact regarding the distribution of $\boldsymbol{\epsilon'}$, we assume that $\mathcal{L}_{\mathcal{D}}(\boldsymbol{w}) \leq \mathbb{E}_{\epsilon'_i\sim\mathcal{N}(0,\rho')}[L_{\mathcal{D}}(\boldsymbol{w}+\boldsymbol{\epsilon'})]$, which indicates adding Gaussian perturbation should not decrease the test error(Foret et al., 2020). Following Foret et al., the generalization bound can be written as follows:

$$\begin{aligned}\mathbb{E}_{\epsilon'_i\sim\mathcal{N}(0,\sigma')}[L_{\mathcal{D}}(\boldsymbol{w}+\boldsymbol{\epsilon'})] &\leq \mathbb{E}_{\epsilon'_i\sim\mathcal{N}(0,\sigma')}[L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon'})] \\ &+ \sqrt{\frac{\frac{1}{4}k\log(1+\frac{||\boldsymbol{w}||_2^2}{k\sigma'^2}) + \frac{1}{4} + log\frac{n}{\delta} + 2log(6n+3k)}{n-1}},\end{aligned} \tag{27}$$

where $\epsilon'_i = \epsilon_i + \epsilon_i^0$

In Equation (27), we assume that ϵ_i and ϵ_i^0 are independent normal variables with mean 0, and corresponding variance σ and σ_0 respectively. Let $\{\epsilon'_i\}$, where $\epsilon'_i = \epsilon_i + \epsilon_i^0$, be the independent normal variable with mean 0 and variance $\sigma'^2 = \sigma^2 + \sigma_0^2$. In particular, at the time when LookSAM can perfectly imitate the SAM procedure by reusing the projected gradient, σ_0^2 becomes zero and σ'^2 equals to σ^2. As $||\boldsymbol{\epsilon'}||_2^2$ has chi-square distribution in this case and based on concentration inequality from Lemma 1 in (Laurent & Massart, 2000), we obtain the following for any positive x:

$$P(||\boldsymbol{\epsilon}+\boldsymbol{\epsilon_0}||_2^2 - k(\sigma^2+\sigma_0^2) \geq 2(\sigma^2+\sigma_0^2)\sqrt{kx} + 2x(\sigma^2+\sigma_0^2)) \leq exp(-x) \tag{28}$$

Let $x = \ln\sqrt{n}$, then we have that

$$P(||\boldsymbol{\epsilon}+\boldsymbol{\epsilon_0}||_2^2 \geq (\sigma^2+\sigma_0^2)(k+2\sqrt{k\ln\sqrt{n}} + 2\ln\sqrt{n})) \leq \frac{1}{\sqrt{n}} \tag{29}$$

With probability of $(1-\frac{1}{\sqrt{n}})$, we have,

$$||\boldsymbol{\epsilon'}||_2^2 = ||\boldsymbol{\epsilon}+\boldsymbol{\epsilon_0}||_2^2 \leq (\sigma^2+\sigma_0^2)(k+2\sqrt{k\ln\sqrt{n}}+2\ln\sqrt{n}) \leq (\sigma^2+\sigma_0^2)k(1+\sqrt{\frac{\ln n}{k}})^2 \leq \rho^2+\rho_0^2, \tag{30}$$

where $\rho_0^2 = \sigma_0^2 k(1+\sqrt{\frac{\ln n}{k}})^2$.

After substituting the value for σ' back to Equation (27), we can generate the following bounds:

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}(\boldsymbol{w}) &\leq (1-\frac{1}{\sqrt{n}}) \max_{||\boldsymbol{\epsilon'}||_p\leq\rho'} \mathcal{L}_S(\boldsymbol{w}+\boldsymbol{\epsilon'}) + \frac{1}{\sqrt{n}} \\ &+ \sqrt{\frac{\frac{1}{4}k\log(1+\frac{||\boldsymbol{w}||_2^2}{\rho^2}(1+\sqrt{\frac{\log(n)}{k}}))^2 + \log\frac{n}{\delta} + 2\log(6n+3k))}{n-1}} \\ &\leq \max_{||\boldsymbol{\epsilon'}||_p\leq\rho'} \mathcal{L}_S(\boldsymbol{w}+\boldsymbol{\epsilon'}) \\ &+ \sqrt{\frac{k\log(1+\frac{||\boldsymbol{w}||_2^2}{\rho'^2}(1+\sqrt{\frac{\log(n)}{k}})^2) + 4\log\frac{n}{\delta} + 8\log(6n+3k))}{n-1}}\end{aligned} \tag{31}$$

where $\rho'^2 = \rho^2 + \rho_0^2$.