SS-SAM : Stochastic Scheduled Sharpness-Aware Minimization for Efficiently Training Deep Neural Networks

Yang Zhao*, Hao Zhang and Xiuyuan Hu

Department of Electronic Engineering, Tsinghua University

March 21, 2022

Abstract

By driving optimizers to converge to flat minima, sharpness-aware minimization (SAM) has shown the power to improve the model generalization. However, SAM requires to perform two forward-backward propagations for one parameter update, which largely burdens the practical computation. In this paper, we propose a novel and efficient training scheme, called Stochastic Scheduled SAM (SS-SAM). Specifically, in SS-SAM, the optimizer is arranged by a predefined scheduling function to perform a random trial at each update step, which would randomly select to perform the SGD optimization or the SAM optimization. In this way, the overall count of propagation pair could be largely reduced. Then, we empirically investigate four typical types of scheduling functions, and demonstrates the computational efficiency and their impact on model performance respectively. We show that with proper scheduling functions, models could be trained to achieve comparable or even better performance with much lower computation cost compared to models trained with only SAM training scheme.

1 Introduction

Deep neural networks (DNNs) have shown great capabilities in solving many real-world complex tasks [8, 14, 2]. However, it is challenging to train a deep neural network to achieve good performance, especially for today's severely overparameterized networks [5, 12]. Although such numerous parameters could improve the expressiveness of DNNs, yet they may complicate the geometry of the loss surface and produce more global and local minima in this huge hypothesis weight space. Moreover, it is demonstrated that optimizers may not converge to the minima with satisfactory model performance when minimizing the conventional empirical loss (such as categorical cross-entropy loss) only [17, 13]. Therefore, effective training schemes are particular necessary for ensuring the optimizers to converge to reliable minima.

Generally, models that converge to flat minima would exhibit better generalization ability. Based on this finding, [7] introduce an efficient training scheme called sharpness-aware minimization (SAM), which encourages the optimizers to converge to flat minima in the hypothesis weight space by minimizing the loss sharpness. Compared to the standard stochastic gradient descent (SGD) training scheme, the SAM training scheme has shown to be more effective in improving model generalization. But on the other hand, the computational cost of the SAM scheme could be almost **twice** that of the standard SGD scheme in practical implementations, since it requires one additional forward-backward propagation for each parameter update during the SAM training process. Regarding this, effective techniques should be proposed to boost the computational efficiency.

In this paper, we present a novel and efficient training scheme, called *Stochastic Scheduled Sharpness-Aware Minimization* (SS-SAM). Compared to the SAM training scheme, the SS-SAM

*Corresponds to: zhao-yan18@mails.tsinghua.edu.cn

training scheme could largely reduce the overall computational overhead and in the meantime achieve better model generalization. Specifically, when training with the SS-SAM scheme, the optimizer would randomly select to perform either the SGD optimization or the SAM optimization with a given probability at any update step. And the probability would be decided by a custom scheduling function predefined before training. So by specifying different scheduling functions, the expected count of forward-backward propagations would be controlled and always stay between the count using the SGD and the SAM training schemes.

We empirically investigate four different types of scheduling functions: constant functions, piecewise functions, linear functions and trigonometric functions. For each type of scheduling functions, we give the corresponding expected propagation count, and present its effect on model performance when parameters in scheduling functions are typical values. We show that by using specific scheduling functions (constant and trigonometric functions), models could reach at comparable results with only 1.5 average propagation count, which is much faster than the SAM scheme. Also, we find with proper scheduling functions, model performance could be also improved compared to training with the SAM scheme.

2 Related Works

In [9], the authors are the first to point out that the flatness of minima could be associated with the model generalization, where models with better generalization should converge to flat minima. And such claim has been supported extensively by both empirical evidences and theoretical demonstrations [10, 4]. In the meantime, researchers are also fascinating by how to implement practical algorithms to force the models to converge to such flat minima. By summarizing this problem to a minimax optimization, [7] introduce the SAM training scheme, which successfully guides optimizers to converge to flat minima. Further, [19] perform gradient descent twice to solve the minimization and maximization respectively in this minimax optimization. In [11], Adaptive SAM training scheme for improving SAM to be able to remain steady when performing weight rescaling operations.In addition to these SAM-related training schemes, [18] seek flat minima by explicitly penalizing the gradient norm of the loss function.

On the other hand, several works introduce techniques to reduce the computational cost when training with the SAM scheme. In particular, instead of using the full batch samples, [1] use only part of batch samples to perform the first forward-backward propagation to make approximations. In [6], the author propose Efficient SAM (ESAM) training scheme, where first randomly selects part of weights for updating and then conditionally selects part of batch samples to perform the second forward-backward propagation. Due to the chain rule, despite part of weights are selected for updating, we still need to compute the gradient for most of the weights in ESAM. Therefore, the main computational efficient comes from reducing the samples. However, unless carefully designed (like ESAM), changing the random sampling policy may introduce unexpected bias to optimization. And more importantly, the count of forward-backward propagation in the SAM training scheme and its variants have not changed essentially.

3 Method

In the standard SGD training scheme, the parameters $\boldsymbol{\theta}$ of DNNs would be updated at each step based on the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ of a given loss function $L(\cdot)$ on batch samples in the training set $\mathcal{D}$. However, merely minimizing the empirical loss would not guarantee that models could converge to minima with satisfactory performance.

3.1 Preliminary: Overview of Sharpness-Aware Minimization

Since flat minima are considered to have better performance, in order to seek the minima where its loss landscape is flatter, [7] propose to optimize the loss,

$$\min_{\boldsymbol{\theta}} \max_{||\boldsymbol{\epsilon}||_2 \leq \rho} L(\boldsymbol{\theta} + \boldsymbol{\epsilon}) \tag{1}$$

where ρ denotes the radius of the neighborhood ball area we would like to optimize. Intuitively, Equation 1 minimizes the maximum in the neighborhood of $\boldsymbol{\theta}$. In this way, the maximum loss within the $\boldsymbol{\theta}$'s neighborhood area could be close to the loss of $\boldsymbol{\theta}$. Therefore, SAM expects to converge to a flatter minimum compared to minimizing the loss $L(\boldsymbol{\theta})$ only.

To solve this minimax optimization, maximization would be resolved first in SAM according to the Cauchy-Schwarz inequality, where $\boldsymbol{\epsilon}$ equals to,

$$\boldsymbol{\epsilon} = \underset{||\boldsymbol{\epsilon}||_2 \leq \rho}{\arg\max} L(\boldsymbol{\theta} + \boldsymbol{\epsilon}) \approx \underset{||\boldsymbol{\epsilon}||_2 \leq \rho}{\arg\max} L(\boldsymbol{\theta}) + \boldsymbol{\epsilon} \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \rho \cdot \frac{\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})}{||\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})||_2} \tag{2}$$

In this way, the minimax optimization could be simplified to $\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta} + \boldsymbol{\epsilon})$ with determinate $\boldsymbol{\epsilon}$. By further approximating $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta} + \boldsymbol{\epsilon})$ to $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ at $\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{\epsilon}$, this minimization could be addressed by the gradient descent framework in practice. In summary, for parameter $\boldsymbol{\theta}_t$ at optimization step t, SAM would implement a two-step calculation strategy to acquire the gradient to update $\boldsymbol{\theta}_t$,

1. In the first step, the first forward-backward propagation would be performed to compute the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ at $\boldsymbol{\theta} = \boldsymbol{\theta}_t$. Then, use Equation 2 to solve $\boldsymbol{\epsilon}$ in the maximization.

2. In the second step, the second forward-backward propagation would be performed to compute the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ at $\boldsymbol{\theta} = \boldsymbol{\theta}_t + \boldsymbol{\epsilon}$ for solving $\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta} + \boldsymbol{\epsilon})$. The parameter $\boldsymbol{\theta}_t$ would be updated based on this gradient.

3.2 Stochastic Scheduled SAM

Apparently, compared to the standard SGD training scheme, the SAM training scheme requires one additional forward-backward propagation at each update step. To reduce the number of forward-backward propagation, we would introduce our efficient SS-SAM training scheme.

In the SS-SAM training scheme, the optimizer would perform a Bernoulli trial independently at each update step $t \in [0, T]$ (T denotes the total update steps here). In each Bernoulli trial, the optimizer would perform the standard SGD optimization with a probability $1 - p(t)$ or perform the SAM optimization with probability $p(t)$. Here, $p(t)$ is a predefined function, and we would call it the scheduling function of the SS-SAM scheme. Clearly, the sample space for this Bernoulli trial corresponds to the set $\Omega = \{\text{SGD, SAM}\}$. Then a random variable could be defined on this sample space, $X(t) : \Omega \to \{0, 1\}$, where $X(t) = 0$ represents performing the SGD optimization while $X(t) = 1$ represents the other. So basically, $X(t) \sim \text{Bernoulli}(p(t))$. Algorithm 1 shows the complete implementation when training with the SS-SAM scheme.

Compared to the SAM training scheme, every time the SGD optimization is performed instead of the SAM optimization durning the SS-SAM training scheme, we would save one forward-backward propagation. It would be easily noted that the scheduling function $p(t)$ would directly control how much this propagation could be saved. To assess the computational cost of different scheduling functions, we would investigate the count of forward-backward propagation pair here. For clarity, we would notate the propagation count at update step t as $\eta(t)$ to and notate the average propagation count over the total update step as η,

$$\eta = \frac{\text{total propagation count}}{T} \tag{3}$$

The smaller η is, the lower total computational cost is. Obviously, $\eta(t) = 1$ and $\eta = 1$ if training with the standard SGD scheme while $\eta(t) = 2$ and $\eta = 2$ if training with the SAM scheme.

Since each update step in SS-SAM is a Bernoulli trial, we could compute the expectation of propagation count. So for step t, the expected propagation count $\overline{\eta}(t)$ is,

$$\overline{\eta}(t) = 2 \cdot p(t) + 1 \cdot (1 - p(t)) = 1 + p(t) \tag{4}$$

Correspondingly, the expected average propagation count $\overline{\eta}$ is,

$$\overline{\eta} = \frac{\sum_{t=0}^{T}(1 + p(t))}{T} = 1 + \frac{\sum_{t=0}^{T} p(t)}{T} \tag{5}$$

Algorithm 1 Stochastic Scheduled Sharpness-Aware Minimization (SS-SAM)

Input: Training set $\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=0}^{N}$; loss function $L(\cdot)$; batch size B; learning rate η; total steps T; neighborhood radius of SAM ρ, scheduling function $p(t)$.
Parameter: Model parameters $\boldsymbol{\theta}$.
Output: Model with final optimized weight $\hat{\boldsymbol{\theta}}$.

Parameter initialization $\boldsymbol{\theta}_0$.
Initialize optimizer with scheduling function $p(t)$.
for step $t = 1$ to T **do**
    Get sample batch $\mathcal{B} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=0}^{B}$.
    Perform the Bernoulli trial with probability $p(t)$ and record the result $X(t)$.
    if $X(t) = 0$ **then**
        Perform the SGD optimization: compute the gradient $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$.
    else
        Perform the SAM optimization: compute the gradient $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ at $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} = \rho \cdot \frac{\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})}{||\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})||_2}$.
    end if
    Update parameter $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{g}_t$
end for
return final parameter $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_T$

where $\overline{\eta} \in [1, 2]$. Basically, $\overline{\eta}$ would be larger if performing the SAM optimization with a higher probability.

4 Study of the Scheduling Function $p(t)$

In this section, we would investigate the computational efficiency and the impact on model performance when training with the SS-SAM scheme under different scheduling functions $p(t)$.

4.1 Basic Setting and Baselines

Throughout our investigation in this section, we would use the WideResNet-28-10 architecture [16] as our experimental target. We would train models from scratch to tackle the image classification tasks on Cifar-{10, 100} datasets. For data augmentation, we would follow the basic strategy, where each image would be randomly flipped horizontally, then padded with four extra pixels and finally cropped randomly to 32×32. Expect for the scheduling functions implemented in the SS-SAM scheme, all the involved models are trained for 200 epochs with exactly the same hyperparameters. The detail hyperparameters are reported in the Appendix section. In particular, it should be mentioned that the radius ρ in the SAM optimization would be set to 0.1, unless otherwise specified. For each training case, we would run with five different seeds and report the average mean and standard deviation of these five runs.

Additional results regarding other model architectures and other data augmentation strategy could be found in the Appendix.

The baseline in our investigation is to use only the standard SGD scheme or SAM scheme for training. Before implementing other scheduling functions in SS-SAM, we would like to clarity the baseline first. To give a clear view on the specific optimization implementations during training, we would use the training scheme plots to illustrate the scheduling function $p(t)$ (the red line) and the entire training implementations $X(t)$ (the blue dots), as shown in Figure 4.1. Each blue dot would represent the optimization implementation at a certain update step, which could be either the SGD optimization $X(t) = 0$ or the SAM optimization $X(t) = 1$. As expected, we could see in Figure 4.1 that for the standard SGD scheme, $p(t) = 0$ and $X(t)$ are all zeros while for the SAM scheme, $p(t) = 1$ and $X(t)$ are all ones.

Table 1 shows the results of WideResNet-28-10 models trained with the SGD scheme and the SAM scheme, respectively. Table 1 includes the testing error rate (Error column), the running

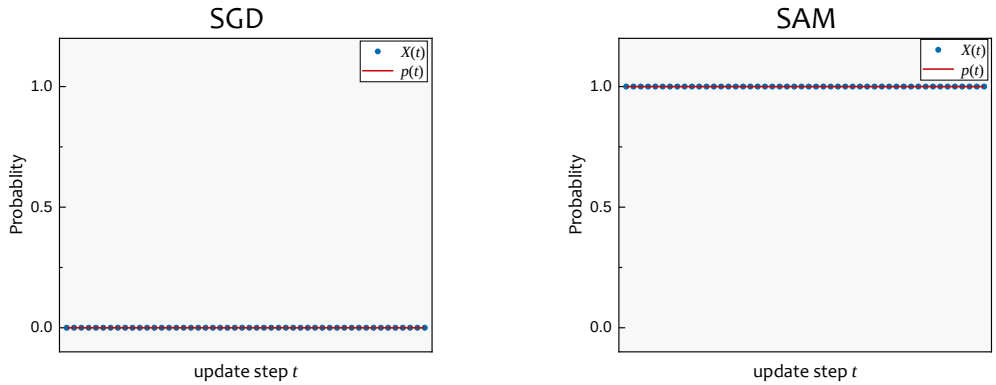

Figure 1: Training scheme plots for the SGD scheme (left) and the SAM scheme (right).

Table 1: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training the SGD scheme and SAM scheme respectively.

	Cifar10			Cifar100		
a_c	Error[%]	Time[m]	$\hat{\eta_c}$	Error[%]	Time[m]	$\hat{\eta_c}$
SGD	$3.53_{\pm 0.10}$	$57.85_{\pm 0.55}$	1.0	$18.69_{\pm 0.12}$	$57.15_{\pm 0.21}$	1.0
SAM	$2.78_{\pm 0.07}$	$103.97_{\pm 0.74}$	2.0	$16.53_{\pm 0.13}$	$103.62_{\pm 0.39}$	2.0

time (Time column) and the expected average propagation count ($\overline{\eta}$ column). For the running time, we would report the total time spent to train for 200 epochs on two A100 Nvidia GPUs. From the table, we could find that since the propagation count of the SAM scheme is twice that of SGD scheme, the practical running time of the SAM scheme would be about 1.79 times that of the SGD scheme ($103.97 \approx 1.79 \times 57.85$).

4.2 Constant Function

First, we would like to investigate the simplest function,

$$p_c(t) = a_c \tag{6}$$

where a_c is a constant and $a_c \in [0, 1]$.

When training with constant scheduling functions, the expected average propagation count $\hat{\eta}_c$ is,

$$\hat{\eta}_c = 1 + \frac{\sum_{t=0}^{T} p_c(t)}{T} = 1 + a_c \tag{7}$$

In this training schedule, for any update step t, optimizers would perform the SAM optimization with a fixed probability a_c and the SGD optimization with $1 - a_c$. Figure 4.2 shows the training schedule plots where a_c equals to 0.2, 0.5, 0.8 respectively. We could clearly see that the larger this constant a_c is, the higher probability the optimizer chooses to perform the SAM optimization. Besides, it could be easily found that the standard SGD scheme and the SAM scheme are two extreme cases when training is arranged by constant scheduling functions.

Then, we would investigate the model performance and computational cost when WideResNet models are trained with constant function schedule. For the constant a_c, we would take values from 0.1 to 0.9 with an interval of 0.1. Table 2 shows the final results.

Based on the Equation 7, $\hat{\eta}$ would increase from 1.1 to 1.9 as a_c increases. Correspondingly, we could see in the table that we would gradually spend more time training the models in practice due to the growth of expected average propagation count.

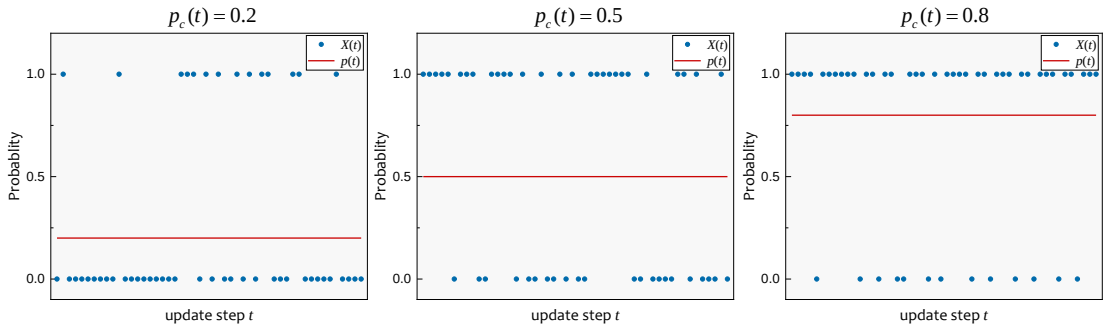

Figure 2: Training scheme plots for constant scheduling functions.

Table 2: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training with constant scheduling functions.

	Cifar10			Cifar100		
a_c	Error[%]	Time[m]	$\hat{\eta}_c$	Error[%]	Time[m]	$\hat{\eta}_c$
SGD	$3.53_{\pm 0.10}$	$57.85_{\pm 0.55}$	1.0	$18.69_{\pm 0.12}$	$57.15_{\pm 0.21}$	1.0
SAM	$2.78_{\pm 0.07}$	$103.97_{\pm 0.74}$	2.0	$16.53_{\pm 0.13}$	$103.62_{\pm 0.39}$	2.0
$a_c = 0.1$	$3.34_{\pm 0.08}$	$62.47_{\pm 0.43}$	1.1	$17.57_{\pm 0.05}$	$61.93_{\pm 0.34}$	1.1
$a_c = 0.2$	$3.16_{\pm 0.09}$	$67.13_{\pm 0.67}$	1.2	$17.19_{\pm 0.08}$	$66.36_{\pm 0.87}$	1.2
$a_c = 0.3$	$3.08_{\pm 0.09}$	$72.68_{\pm 0.30}$	1.3	$16.90_{\pm 0.11}$	$72.83_{\pm 1.03}$	1.3
$a_c = 0.4$	$2.94_{\pm 0.07}$	$77.29_{\pm 1.13}$	1.4	$16.76_{\pm 0.21}$	$76.75_{\pm 1.36}$	1.4
$a_c = 0.5$	$2.87_{\pm 0.05}$	$81.94_{\pm 1.25}$	1.5	$16.51_{\pm 0.25}$	$80.97_{\pm 1.64}$	1.5
$a_c = 0.6$	**$2.75_{\pm 0.04}$**	**$85.41_{\pm 0.38}$**	**1.6**	**$16.49_{\pm 0.13}$**	**$85.26_{\pm 0.91}$**	**1.6**
$a_c = 0.7$	$2.74_{\pm 0.03}$	$89.69_{\pm 0.34}$	1.7	$16.21_{\pm 0.12}$	$88.64_{\pm 0.54}$	1.7
$a_c = 0.8$	**$2.71_{\pm 0.08}$**	**$95.02_{\pm 0.68}$**	**1.8**	**$16.17_{\pm 0.08}$**	**$95.31_{\pm 0.63}$**	**1.8**
$a_c = 0.9$	$2.79_{\pm 0.06}$	$100.10_{\pm 1.55}$	1.9	$16.44_{\pm 0.04}$	$99.72_{\pm 0.74}$	1.9

As for the model performance, we could find that as long as SAM optimization is involved during training, the testing error rate could be reduced compared to that trained only with the SGD scheme. However, the model performance could not be improved continuously with more implementations of the SAM optimization steps. For $a_c \geq 0.6$, the testing error rates could already be lower than that of the SAM scheme. In particular, when $a_c = 0.8$, models would achieve the best performance, and now the running time is 1.64 times that of SGD scheme, which is much shorter than that of the SAM scheme. Additionally, we could see from the standard deviation that despite the randomness introduced when training with the SS-SAM scheme, the error rates would stay relatively stable over the five runs.

4.3 Piecewise Function

Then, we would like to investigate piecewise scheduling functions,

$$p_p(t) = \begin{cases} a_p, & t \leq b_p T \\ 1 - a_p, & t > b_p T \end{cases} \tag{8}$$

where a_p and b_p are constants and $a_p \in [0, 1]$ and $b_p \in [0, 1]$.

When training with such a piecewise scheduling function, the expected average propagation count $\hat{\eta}_c$ is,

$$\hat{\eta}_p = 1 + \frac{\sum_{t=0}^{T} p(t)}{T} = 1 + \frac{a_p b_p T + (1 - a_p)(T - b_p T)}{T} = 2 + 2a_p b_p - b_p - a_p \tag{9}$$

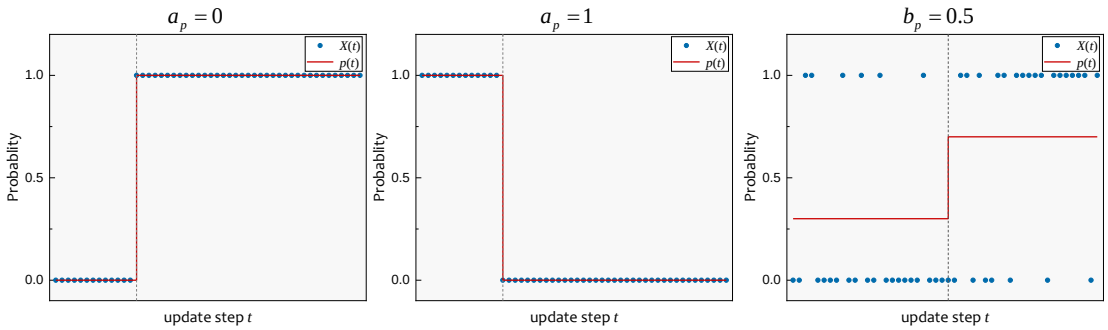

Figure 3: Training scheme plots for typical piecewise scheduling functions.

Table 3: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training with the first group of piecewise scheduling functions.

	Cifar10			Cifar100		
b_p	Error[%]	Time[m]	$\hat{\eta_p}$	Error[%]	Time[m]	$\hat{\eta_p}$
SGD	$3.53_{\pm 0.10}$	$57.85_{\pm 0.55}$	1.0	$18.69_{\pm 0.12}$	$57.15_{\pm 0.21}$	1.0
SAM	$2.78_{\pm 0.07}$	$103.97_{\pm 0.74}$	2.0	$16.53_{\pm 0.13}$	$103.62_{\pm 0.39}$	2.0
$b_p = 0.1$	$2.69_{\pm 0.07}$	$99.29_{\pm 0.64}$	1.9	$16.31_{\pm 0.21}$	$99.65_{\pm 0.81}$	1.9
$b_p = 0.2$	$2.75_{\pm 0.04}$	$95.64_{\pm 1.08}$	1.8	$16.65_{\pm 0.09}$	$95.86_{\pm 1.09}$	1.8
$b_p = 0.3$	$2.76_{\pm 0.09}$	$90.14_{\pm 1.05}$	1.7	$16.87_{\pm 0.16}$	$89.19_{\pm 1.21}$	1.7
$b_p = 0.4$	$2.86_{\pm 0.04}$	$85.78_{\pm 1.17}$	1.6	$16.90_{\pm 0.18}$	$85.11_{\pm 1.39}$	1.6
$b_p = 0.5$	$2.87_{\pm 0.07}$	$82.31_{\pm 1.19}$	1.5	$16.79_{\pm 0.15}$	$83.04_{\pm 1.13}$	1.5
$b_p = 0.6$	$2.96_{\pm 0.06}$	$77.75_{\pm 0.80}$	1.4	$16.91_{\pm 0.22}$	$78.06_{\pm 0.98}$	1.4
$b_p = 0.7$	$2.95_{\pm 0.04}$	$71.70_{\pm 0.42}$	1.3	$16.83_{\pm 0.17}$	$71.57_{\pm 0.65}$	1.3
$b_p = 0.8$	$3.07_{\pm 0.02}$	$68.26_{\pm 0.97}$	1.2	$17.05_{\pm 0.09}$	$68.64_{\pm 0.41}$	1.2
$b_p = 0.9$	$3.32_{\pm 0.05}$	$64.59_{\pm 0.27}$	1.1	$17.77_{\pm 0.14}$	$64.31_{\pm 0.49}$	1.1

Following this schedule, training would be divided into two stages at the update step b_pT. In the first stage, the beginning b_pT update steps would be arranged by a constant scheduling function $p(t) = a_p$. Then in the second stage, the rest update steps would be arranged by another constant scheduling function $p(t) = 1-a_p$. In other words, based on the previous demonstration, the optimizer could perform the SAM optimization with a probability of a_p in the first stage while this probability would change to $1 - a_p$ in the second stage.

In our experiments, we would consider three typical groups of piecewise scheduling functions, as shown in Figure 4.3. Specifically, in the first group, we would set $a_p = 0$ and change b_p from 0.1 to 0.9 with an interval of 0.1. Now, the optimizer actually behaves in a deterministic manner, which performs the SGD optimization for all the steps in the first stage and then the SAM optimization for all the rest steps. The larger b_p is, the longer the SGD optimization would be performed. Table 3 shows the final results. As expected, the running time would progressively decrease as b_p increases since more SGD optimization would be implemented during training. From the table, we could also find that for both Cifar10 and Cifar100, as more rest steps are implemented with SAM optimization, the model performance would get better and better. Notably, the model would achieve the best performance when $b_p = 0.1$, which is also better than the model trained only with SAM scheme.

Next, in the second group, we would set $a_p = 1$ and change b_p similarly from 0.1 to 0.9 with an interval of 0.1. This time, training would be arranged in the exact opposite way as when $a_p = 0$. The optimizer would perform the SAM optimization in the first stage and then perform the SGD optimization for the rest steps. Table 4 shows the final results. Opposite to

Table 4: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training with the second group of piecewise scheduling functions.

	Cifar10			Cifar100		
b_p	Error[%]	Time[m]	$\hat{\eta_p}$	Error[%]	Time[m]	$\hat{\eta_p}$
SGD	$3.53_{\pm 0.10}$	$57.85_{\pm 0.55}$	1.0	$18.69_{\pm 0.12}$	$57.15_{\pm 0.21}$	1.0
SAM	$2.78_{\pm 0.07}$	$103.97_{\pm 0.74}$	2.0	$16.53_{\pm 0.13}$	$103.62_{\pm 0.39}$	2.0
$b_p = 0.1$	$3.56_{\pm 0.06}$	$62.91_{\pm 0.71}$	1.1	$18.53_{\pm 0.15}$	$62.08_{\pm 0.20}$	1.1
$b_p = 0.2$	$3.52_{\pm 0.08}$	$67.81_{\pm 0.57}$	1.2	$18.74_{\pm 0.21}$	$66.82_{\pm 0.65}$	1.2
$b_p = 0.3$	$3.50_{\pm 0.04}$	$71.99_{\pm 0.44}$	1.3	$18.36_{\pm 0.13}$	$72.21_{\pm 0.47}$	1.3
$b_p = 0.4$	$3.54_{\pm 0.08}$	$76.74_{\pm 1.28}$	1.4	$18.21_{\pm 0.17}$	$76.18_{\pm 1.01}$	1.4
$b_p = 0.5$	$3.53_{\pm 0.11}$	$82.06_{\pm 0.65}$	1.5	$18.23_{\pm 0.24}$	$81.55_{\pm 1.12}$	1.5
$b_p = 0.6$	$3.41_{\pm 0.09}$	$85.58_{\pm 1.04}$	1.4	$18.06_{\pm 0.11}$	$85.60_{\pm 0.83}$	1.4
$b_p = 0.7$	$3.02_{\pm 0.06}$	$90.42_{\pm 0.87}$	1.7	$17.62_{\pm 0.14}$	$90.01_{\pm 1.70}$	1.7
$b_p = 0.8$	$2.82_{\pm 0.04}$	$94.06_{\pm 0.32}$	1.8	$16.99_{\pm 0.13}$	$94.71_{\pm 0.93}$	1.8
$b_p = 0.9$	$2.67_{\pm 0.07}$	$99.31_{\pm 0.35}$	1.9	$16.50_{\pm 0.11}$	$98.72_{\pm 0.58}$	1.9

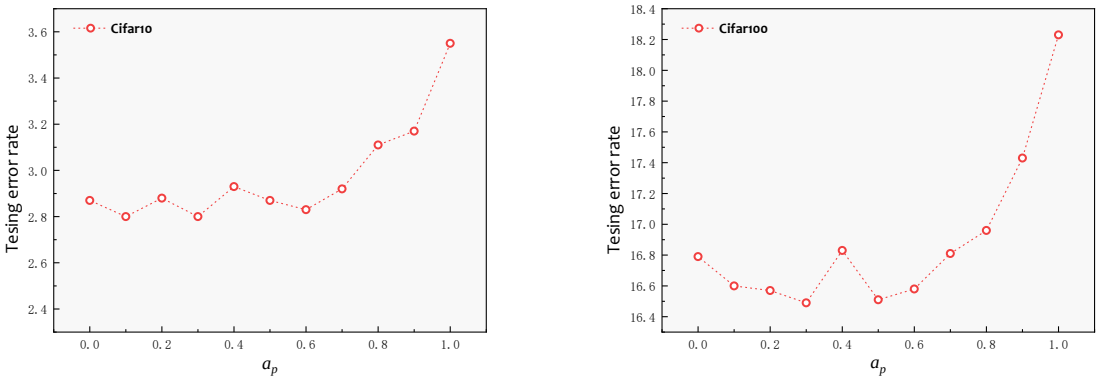

Figure 4: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training with the third group of piecewise scheduling functions.

$a_p = 0$, when $a_p = 1$, more running time is required as b_p increases. And we could find that for this training scheme, the model performances are rather close when $b_p \leq 0.6$. Considering that the optimizer would perform SAM optimization first and SGD optimization last under this schedule, if SAM optimization would not accumulate to some degree, the SGD optimization would dominate the training. It should be also noted that models could also achieve comparable performance when the SGD optimization is performed in only last few steps. In particular, when $b_p = 0, 9$, it even outperforms the previous best cases.

Lastly, in the third group, we would set $b_p = 0.5$ and change a_p from 0 to 1 with an interval of 0.1. Since b_p is fixed to 0.5, the optimizer would follow the scheduling function $p(t) = a_p$ in the first half update steps and change to $p(t) = 1 - a_p$ in the last half update steps. According to Equation 9, the expected average propagation count $\overline{\eta}$ of this group would be 1.5 for any valid a_p, which means that the corresponding running time would be very close ($81.83_{\pm 1.14}$[m] in practical implementation). Figure 4.3 shows the model performance of this group. From Figure 4.3, we could find that for $a_p \leq 0.7$, the model performances are very close. But when $a_p \geq 0.8$, the testing error rates would be increased rapidly as a_p increases. This is mainly because that most of the SAM optimization would be performed in the first stage. Besides, for this group of scheduling functions, $a_p = 0$ is actually the case where $b_p = 0.5$ in the first group;

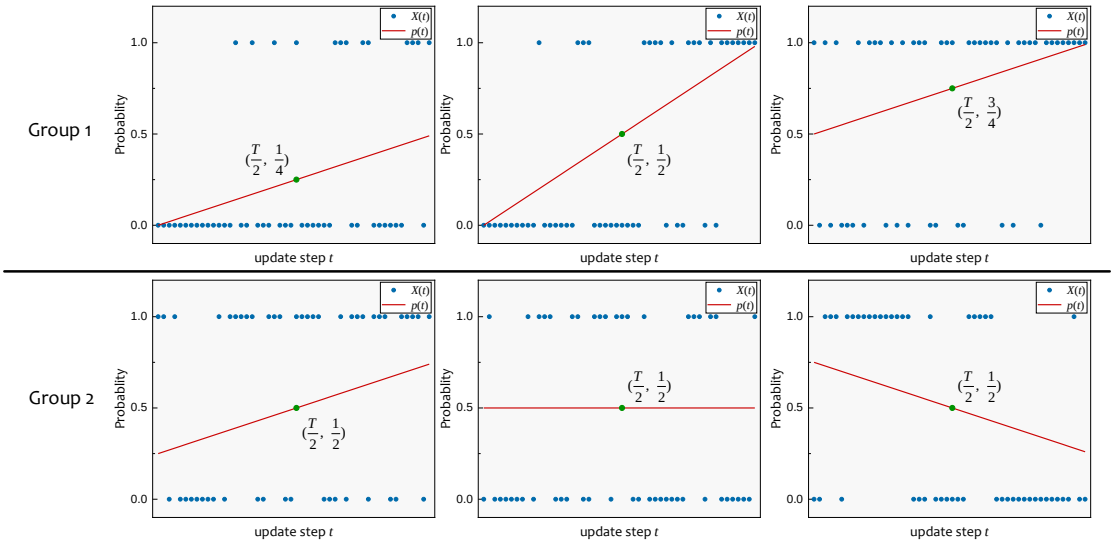

Figure 5: Training scheme plots for typical linear scheduling functions.

$a_p = 0.5$ is actually the constant scheduling function where $a_c = 0.5$; $a_p = 1$ is actually the case where $b_p = 0.5$ in the second group.

4.4 Linear Function

Then, we would like to investigate linear scheduling functions,

$$p_l(t) = a_l t + b_l \tag{10}$$

where $0 \leq p_l(t) \leq 1$ for $t \in [0, T]$.

When training with linear scheduling functions, the expected average propagation count $\overline{\eta}_c$ is,

$$\hat{\eta} = 1 + \frac{\sum_{t=0}^{T} p_l(t)}{T} = 1 + \frac{a_l}{2}(T+1) + b_l \approx 1 + \frac{a_l T}{2} + b_l = 1 + p_l(\frac{T}{2}) \tag{11}$$

Here $T + 1 \approx T$ since T is generally much greater than 1. It should be mentioned that for such linear scheduling functions, the computational cost is actually decided by the probability value at half of the total update steps.

Following the linear scheduling functions, the probability of that the optimizer chooses to perform the SGD optimization or the SAM optimization would change monotonously and gradually during the training process. Here, we would focus on investigating two typical groups of linear scheduling functions, as shown in Figure 4.4.

In the first group, the linear scheduling functions would pass through the two points, where the first point is $(\frac{T}{2}, m)$ and the second point is either $(0, 0)$[1] or $(1, 1)$. Here, the parameter m denotes the probability at the update step $\frac{T}{2}$, and actually m controls the whole scheduling function. The optimizer would perform more SAM optimization steps during training as m increases. In our experiments, we would set it from 0.1 to 0.9 with an interval of 0.1. Table 5 shows the final results. From the Equation 11, we could know that for every time m increases 0.1, $\overline{\eta}$ would increase by 0.1. Also, we could find in the table that as performing more SAM optimization, the model performance would be more and more better.

In the second group, the linear scheduling functions would pass through a fixed point $(\frac{T}{2}, \frac{1}{2})$ and $(0, b_l)$. Based on Equation 11, we could know that $\overline{\eta} = 1.5$ for all the cases. Figure 4.4 shows

[1]When $m > 0.5$, this linear scheduling function would be invalid for $(0, 0)$ since $p_l(T)$ would be greater than 1. The same situation would happen for $m < 0.5$ and $(1, 1)$.

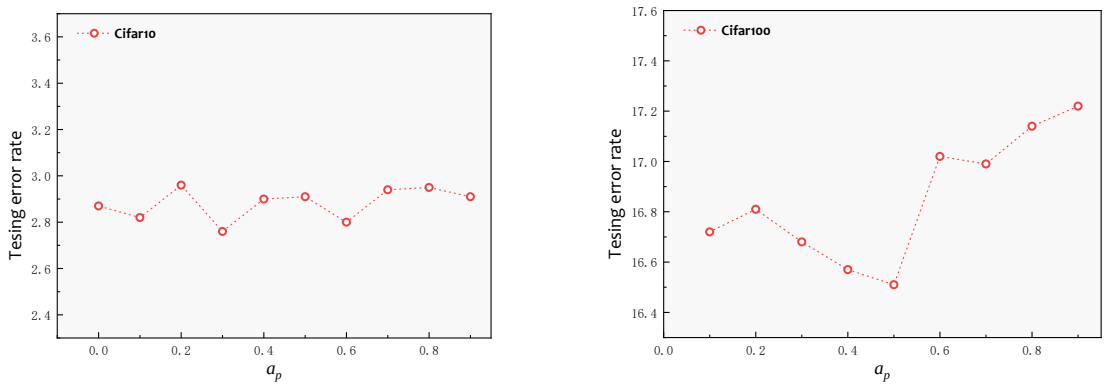

Figure 6: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training with the second group of linear scheduling functions.

Table 5: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training with the first group of linear scheduling functions.

	Cifar10			Cifar100		
$p(\frac{T}{2})$	Error[%]	Time[m]	$\hat{\eta_p}$	Error[%]	Time[m]	$\hat{\eta_p}$
SGD	$3.53_{\pm 0.10}$	$57.85_{\pm 0.55}$	1.0	$18.69_{\pm 0.12}$	$57.15_{\pm 0.21}$	1.0
SAM	$2.78_{\pm 0.07}$	$103.97_{\pm 0.74}$	2.0	$16.53_{\pm 0.13}$	$103.62_{\pm 0.39}$	2.0
$p(\frac{T}{2}) = 0.1$	$3.22_{\pm 0.08}$	$62.73_{\pm 1.11}$	1.1	$17.88_{\pm 0.18}$	$62.87_{\pm 0.99}$	1.1
$p(\frac{T}{2}) = 0.2$	$2.97_{\pm 0.05}$	$67.56_{\pm 1.06}$	1.2	$17.49_{\pm 0.11}$	$67.72_{\pm 1.50}$	1.2
$p(\frac{T}{2}) = 0.3$	$2.92_{\pm 0.05}$	$72.74_{\pm 0.36}$	1.3	$17.18_{\pm 0.13}$	$72.57_{\pm 0.47}$	1.3
$p(\frac{T}{2}) = 0.4$	$2.89_{\pm 0.08}$	$76.05_{\pm 0.46}$	1.4	$17.25_{\pm 0.22}$	$76.78_{\pm 0.92}$	1.4
$p(\frac{T}{2}) = 0.5$	$2.91_{\pm 0.04}$	$81.53_{\pm 0.79}$	1.5	$17.11_{\pm 0.21}$	$81.30_{\pm 1.01}$	1.5
$p(\frac{T}{2}) = 0.6$	$2.85_{\pm 0.03}$	$85.82_{\pm 0.94}$	1.4	$16.97_{\pm 0.19}$	$86.04_{\pm 0.86}$	1.4
$p(\frac{T}{2}) = 0.7$	$2.77_{\pm 0.04}$	$90.39_{\pm 1.07}$	1.7	$16.87_{\pm 0.13}$	$90.77_{\pm 1.80}$	1.7
$p(\frac{T}{2}) = 0.8$	$2.78_{\pm 0.06}$	$95.43_{\pm 0.98}$	1.8	$16.82_{\pm 0.19}$	$95.10_{\pm 0.78}$	1.8
$p(\frac{T}{2}) = 0.9$	$2.69_{\pm 0.05}$	$99.75_{\pm 1.09}$	1.9	$16.19_{\pm 0.14}$	$100.16_{\pm 1.64}$	1.9

the results. We could find in the figure that models could acquire comparable performance, since the scheduling functions here are actually quite close for all the b_l here.

4.5 Trigonometric Function

Finally, we would like to investigate the scheduling functions which are trigonometric functions $p_{tr}(t)$. Here, we would confine the trigonometric functions to only sinusoidal functions and cosine functions. And more specifically, we focus on investigating four scheduling functions,

$$\begin{cases} p_{cos1}(t) & = \frac{1}{2} + \frac{1}{2}\cos\frac{t}{T}\pi \\ p_{cos2}(t) & = 1 - p_{cos1}(t) = \frac{1}{2} - \frac{1}{2}\cos\frac{t}{T}\pi \\ p_{sin1}(t) & = \sin\frac{t}{T}\pi \\ p_{sin2}(t) & = 1 - p_{sin1}(t) = 1 - \sin\frac{t}{T}\pi \end{cases} \tag{12}$$

Note that all these functions are in the range between 0 and 1.

When training with these trigonometric scheduling functions, the expected average propagation count $\overline{\eta}_c$ equals to $\frac{3}{2}$, where we would provide detailed demonstration in the Appendix

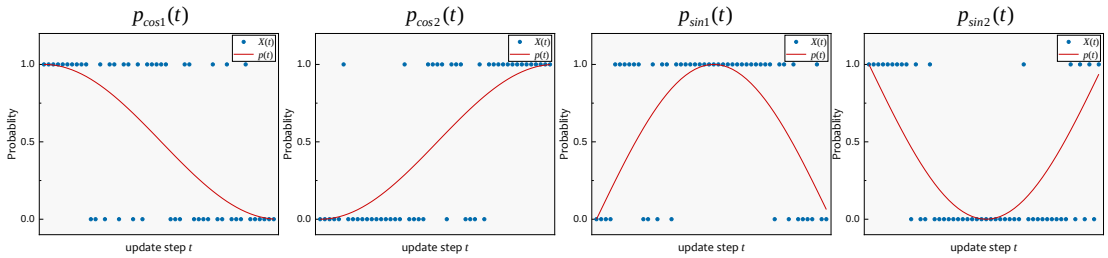

Figure 7: Training scheme plots for the four trigonometric scheduling functions.

Table 6: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets when training with the four trigonometric scheduling functions.

	Cifar10			Cifar100		
$p(\frac{T}{2})$	Error[%]	Time[m]	$\hat{\eta_p}$	Error[%]	Time[m]	$\hat{\eta_p}$
SGD	$3.53_{\pm 0.10}$	$57.85_{\pm 0.55}$	1.0	$18.69_{\pm 0.12}$	$57.15_{\pm 0.21}$	1.0
SAM	$2.78_{\pm 0.07}$	$103.97_{\pm 0.74}$	2.0	$16.53_{\pm 0.13}$	$103.62_{\pm 0.39}$	2.0
$p_{cos1}(t)$	$3.16_{\pm 0.09}$	$81.87_{\pm 1.19}$	1.5	$17.08_{\pm 0.12}$	$81.41_{\pm 0.88}$	1.5
$p_{cos2}(t)$	$2.86_{\pm 0.10}$	$81.04_{\pm 1.03}$	1.5	$16.77_{\pm 0.18}$	$82.21_{\pm 1.47}$	1.5
$p_{sin1}(t)$	$2.81_{\pm 0.06}$	$88.88_{\pm 1.16}$	1.5	$16.69_{\pm 0.15}$	$89.55_{\pm 1.22}$	1.5
$p_{sin2}(t)$	$3.21_{\pm 0.13}$	$89.19_{\pm 0.77}$	1.5	$17.15_{\pm 0.10}$	$89.49_{\pm 0.89}$	1.5

section.

Figure 4.5 shows the training scheme plots of the four functions and Table 6 shows the final results. From the figure and the table, we could find that for $p_{cos1}(t)$, the optimizer would perform the SAM optimization with a high probability in the beginning and this probability would gradually drop to 0 at the end of training. Since most of the SAM optimization would be performed in the beginning, we would not acquire relatively good model performance. On the contrary, for $p_{cos2}(t)$, the testing error rates would be much lower since the optimizer would perform the SAM optimization with a high probability near the end of training.

As for $p_{sin1}(t)$, the optimizer would focus on performing the SAM optimization in the middle of the training. Models would acquire the best performance in this case than other cases. However, opposite to $p_{sin1}(t)$, the testing error rate would be much higher for $p_{sin2}(t)$.

5 Summary

We propose a novel and efficient training scheme, called Stochastic Scheduled SAM (SS-SAM), for reducing the computational overhead in the SAM training scheme. In the SS-SAM scheme, optimizers would perform a Bernoulli trial with a scheduling function at each step. This trial would decide to perform either the SGD optimization or the SAM optimization for this update step. In this way, SS-SAM would expect to perform less number of forward-backward propagation for each step. Then, we empirically investigate four typical types of scheduling functions, and demonstrate the computational efficiency and their impact on model performance respectively. We show that with proper scheduling functions, models could be trained to achieve comparable or even better performance with much lower computation cost compared to models trained with only SAM training scheme. Further works would focus on exploiting more appropriate scheduling functions that could boost the computational efficiency and improve the model generalization.

References

[1] Bahri, D., Mobahi, H., Tay, Y.: Sharpness-aware minimization improves language model generalization. arXiv preprint arXiv:2110.08529 (2021)

[2] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[3] Devries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXivPreprint **abs/1708.04552** (2017), http://arxiv.org/abs/1708.04552

[4] Dinh, L., Pascanu, R., Bengio, S., Bengio, Y.: Sharp minima can generalize for deep nets. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017. vol. 70, pp. 1019–1028 (2017)

[5] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: 9th International Conference on Learning Representations, ICLR 2021 (2021)

[6] Du, J., Yan, H., Feng, J., Zhou, J.T., Zhen, L., Goh, R.S.M., Tan, V.Y.F.: Efficient sharpness-aware minimization for improved training of neural networks. arXivPreprint **abs/2110.03141** (2021)

[7] Foret, P., Kleiner, A., Mobahi, H., Neyshabur, B.: Sharpness-aware minimization for efficiently improving generalization. In: 9th International Conference on Learning Representations, ICLR 2021 (2021)

[8] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016. pp. 770–778 (2016)

[9] Hochreiter, S., Schmidhuber, J.: Flat minima. Neural Comput. **9**(1), 1–42 (1997)

[10] Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. In: 5th International Conference on Learning Representations, ICLR 2017 (2017)

[11] Kwon, J., Kim, J., Park, H., Choi, I.K.: ASAM: adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In: Proceedings of the 38th International Conference on Machine Learning, ICML 2021. Proceedings of Machine Learning Research, vol. 139, pp. 5905–5914 (2021)

[12] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10012–10022 (2021)

[13] Neyshabur, B., Bhojanapalli, S., McAllester, D., Srebro, N.: Exploring generalization in deep learning. In: Advances in Neural Information Processing Systems. pp. 5947–5956 (2017)

[14] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)

[15] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015 (2015)

[16] Zagoruyko, S., Komodakis, N.: Wide residual networks. In: Proceedings of the British Machine Vision Conference 2016, BMVC 2016 (2016)

[17] Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: 5th International Conference on Learning Representations, ICLR 2017 (2017)

[18] Zhao, Y., Zhang, H., Hu, X.: Penalizing gradient norm for efficiently improving generalization in deep learning. arXiv preprint arXiv:2202.03599 (2022)

[19] Zheng, Y., Zhang, R., Mao, Y.: Regularizing neural networks via adversarial model perturbation. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021. pp. 8156–8165 (2021)

A Calculation of Expected Count of Propagation Pair for Trigonometric Scheduling Functions

The expected average propagation count $\overline{\eta}$ is,

$$\hat{\eta} = 1 + \frac{\sum_{t=0}^{T} p(t)}{T} \tag{13}$$

We would calculate the core part in trigonometric scheduling functions first,

$$\sum_{t=0}^{T} g(\frac{t}{T}\pi) \tag{14}$$

where $g \in \{\sin, \cos\}$. We would use the Euler's identity,

$$e^{ix} = \cos x + i \sin x \tag{15}$$

Therefore, we have $\cos x = \Re\{e^{ix}\}$ and $\sin x = \Im\{e^{ix}\}$, where $\Re\{\cdot\}$ and $\Im\{\cdot\}$ denote the real part and imaginary part.

In this way, for $g = \cos$, Equation 14 would be,

$$\begin{aligned}
\sum_{t=0}^{T} \cos(\frac{t}{T}\pi) &= \sum_{t=0}^{T} \Re\{e^{i\frac{t}{T}\pi}\} \\
&= \Re\{\sum_{t=0}^{T} e^{i\frac{t}{T}\pi}\} \\
&= \Re\{\frac{e^0(1 - e^{i\frac{T+1}{T}\pi})}{1 - e^{i\frac{1}{T}\pi}}\} \\
&= \Re\{\frac{e^{i\frac{T+1}{2T}\pi} \cdot (e^{-i\frac{T+1}{2T}\pi} - e^{i\frac{T+1}{2T}\pi})}{e^{i\frac{1}{2T}\pi} \cdot (e^{-i\frac{1}{2T}\pi} - e^{i\frac{1}{2T}\pi})}\} \\
&= \Re\{e^{i\frac{T}{2T}\pi}\frac{\sin(\frac{T+1}{2T}\pi)}{\sin(\frac{1}{2T})\pi}\} \\
&= \cos(\frac{T}{2T}\pi)\frac{\sin(\frac{T+1}{2T}\pi)}{\sin(\frac{1}{2T})\pi} \\
&= 0 \quad (\cos\frac{\pi}{2} = 0)
\end{aligned} \tag{16}$$

Therefore, for $p_{cos1}(t) = \frac{1}{2} + \frac{1}{2}\cos\frac{t}{T}\pi$ and $p_{cos2}(t) = 1 - p_{cos1}(t) = \frac{1}{2} - \frac{1}{2}\cos\frac{t}{T}\pi$, their corresponding expected count of propagation pair would be,

$$\hat{\eta} = 1 + \frac{\sum_{t=0}^{T} p(t)}{T} = 1 + \frac{1}{2} = 1.5 \tag{17}$$

B Experiment Results when using Cutout Regularization

In addition to the basic data augmentation strategy used in the previous section, we would also investigate the effect when using the Cutout Regularization [3]. Here, the training hyperparameters are the same as them used in the previous sections.

The tables below show the final results, where trainings are scheduled by constant scheduling functions (Table 7), the first group of piecewise scheduling functions (Table 8) and the first group of linear scheduling functions (Table 9) and trigonometric scheduling functions (Table 10), respectively. We could see that when implementing the Cutout regularization, with proper scheduling functions, the computational cost could be reduced, while models could acquire comparable or even better generalization with the SAM training scheme.

Table 7: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets with Cutout regularization when training with constant scheduling functions.

	Cifar10			Cifar100		
a_c	Error[%]	Time[m]	$\hat{\eta_c}$	Error[%]	Time[m]	$\hat{\eta_c}$
SGD	$2.81_{\pm 0.07}$	$57.33_{\pm 0.71}$	1.0	$16.91_{\pm 0.10}$	$57.46_{\pm 0.77}$	1.0
SAM	$2.43_{\pm 0.13}$	$105.17_{\pm 0.94}$	2.0	$14.87_{\pm 0.16}$	$103.14_{\pm 1.12}$	2.0
$a_c = 0.1$	$2.67_{\pm 0.05}$	$61.82_{\pm 0.17}$	1.1	$16.14_{\pm 0.18}$	$62.46_{\pm 0.85}$	1.1
$a_c = 0.2$	$2.53_{\pm 0.06}$	$67.52_{\pm 1.12}$	1.2	$15.97_{\pm 0.16}$	$67.85_{\pm 0.96}$	1.2
$a_c = 0.3$	$2.46_{\pm 0.06}$	$72.33_{\pm 1.23}$	1.3	$15.56_{\pm 0.14}$	$72.29_{\pm 0.99}$	1.3
$a_c = 0.4$	$2.40_{\pm 0.06}$	$76.31_{\pm 1.31}$	1.4	$15.17_{\pm 0.22}$	$76.04_{\pm 1.04}$	1.4
$a_c = 0.5$	$2.32_{\pm 0.06}$	$80.81_{\pm 0.96}$	1.5	$15.10_{\pm 0.11}$	$80.37_{\pm 0.78}$	1.5
$a_c = 0.6$	$2.31_{\pm 0.07}$	$85.60_{\pm 1.09}$	1.6	$14.96_{\pm 0.08}$	$84.98_{\pm 1.06}$	1.6
$a_c = 0.7$	$2.25_{\pm 0.08}$	$89.66_{\pm 1.19}$	1.7	$14.94_{\pm 0.09}$	$89.84_{\pm 1.53}$	1.7
$a_c = 0.8$	$2.23_{\pm 0.03}$	$95.05_{\pm 1.10}$	1.8	$14.81_{\pm 0.09}$	$94.42_{\pm 1.35}$	1.8
$a_c = 0.9$	$2.31_{\pm 0.06}$	$98.18_{\pm 0.20}$	1.9	$14.71_{\pm 0.03}$	$97.79_{\pm 1.01}$	1.9

Table 8: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets with Cutout regularization when training with the first group of piecewise scheduling functions.

	Cifar10			Cifar100		
b_p	Error[%]	Time[m]	$\hat{\eta_p}$	Error[%]	Time[m]	$\hat{\eta_p}$
SGD	$2.81_{\pm 0.07}$	$57.33_{\pm 0.71}$	1.0	$16.91_{\pm 0.10}$	$57.46_{\pm 0.77}$	1.0
SAM	$2.43_{\pm 0.13}$	$105.17_{\pm 0.94}$	2.0	$14.87_{\pm 0.16}$	$103.14_{\pm 1.12}$	2.0
$b_p = 0.1$	$2.19_{\pm 0.05}$	$101.47_{\pm 1.54}$	1.9	$14.75_{\pm 0.24}$	$100.44_{\pm 1.47}$	1.9
$b_p = 0.2$	$2.30_{\pm 0.05}$	$95.70_{\pm 1.39}$	1.8	$14.84_{\pm 0.09}$	$94.89_{\pm 0.70}$	1.8
$b_p = 0.3$	$2.34_{\pm 0.03}$	$90.84_{\pm 1.20}$	1.7	$14.99_{\pm 0.26}$	$90.27_{\pm 0.51}$	1.7
$b_p = 0.4$	$2.34_{\pm 0.02}$	$85.50_{\pm 0.18}$	1.6	$14.98_{\pm 0.28}$	$86.82_{\pm 1.19}$	1.6
$b_p = 0.5$	$2.46_{\pm 0.02}$	$81.46_{\pm 0.81}$	1.5	$15.24_{\pm 0.22}$	$81.55_{\pm 1.20}$	1.5
$b_p = 0.6$	$2.44_{\pm 0.04}$	$77.59_{\pm 1.02}$	1.4	$15.24_{\pm 0.35}$	$76.74_{\pm 0.62}$	1.4
$b_p = 0.7$	$2.47_{\pm 0.05}$	$72.27_{\pm 0.61}$	1.3	$15.38_{\pm 0.33}$	$71.83_{\pm 0.04}$	1.3
$b_p = 0.8$	$2.51_{\pm 0.07}$	$67.80_{\pm 0.64}$	1.2	$15.37_{\pm 0.18}$	$67.82_{\pm 0.71}$	1.2
$b_p = 0.9$	$2.69_{\pm 0.04}$	$64.08_{\pm 0.87}$	1.1	$15.70_{\pm 0.21}$	$63.77_{\pm 0.82}$	1.1

Table 9: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets with Cutout regularization when training with the first group of linear scheduling functions.

	Cifar10			Cifar100		
$p(\frac{T}{2})$	Error[%]	Time[m]	$\hat{\eta_p}$	Error[%]	Time[m]	$\hat{\eta_p}$
SGD	$2.81_{\pm 0.07}$	$57.33_{\pm 0.71}$	1.0	$16.91_{\pm 0.10}$	$57.46_{\pm 0.77}$	1.0
SAM	$2.43_{\pm 0.13}$	$105.17_{\pm 0.94}$	2.0	$14.87_{\pm 0.16}$	$103.14_{\pm 1.12}$	2.0
$p(\frac{T}{2}) = 0.1$	$2.63_{\pm 0.02}$	$63.08_{\pm 0.15}$	1.1	$15.83_{\pm 0.27}$	$62.97_{\pm 0.46}$	1.1
$p(\frac{T}{2}) = 0.2$	$2.58_{\pm 0.06}$	$67.32_{\pm 0.09}$	1.2	$15.67_{\pm 0.16}$	$67.53_{\pm 0.18}$	1.2
$p(\frac{T}{2}) = 0.3$	$2.45_{\pm 0.05}$	$72.11_{\pm 0.01}$	1.3	$15.33_{\pm 0.17}$	$72.09_{\pm 0.09}$	1.3
$p(\frac{T}{2}) = 0.4$	$2.37_{\pm 0.07}$	$77.01_{\pm 0.87}$	1.4	$15.30_{\pm 0.19}$	$76.91_{\pm 0.38}$	1.4
$p(\frac{T}{2}) = 0.5$	$2.38_{\pm 0.02}$	$80.73_{\pm 0.11}$	1.5	$15.33_{\pm 0.11}$	$81.19_{\pm 0.69}$	1.5
$p(\frac{T}{2}) = 0.6$	$2.27_{\pm 0.10}$	$86.65_{\pm 1.41}$	1.4	$14.97_{\pm 0.02}$	$86.93_{\pm 0.42}$	1.4
$p(\frac{T}{2}) = 0.7$	$2.25_{\pm 0.03}$	$91.20_{\pm 1.62}$	1.7	$14.95_{\pm 0.23}$	$91.78_{\pm 0.32}$	1.7
$p(\frac{T}{2}) = 0.8$	$2.22_{\pm 0.03}$	$94.25_{\pm 0.12}$	1.8	$14.68_{\pm 0.14}$	$94.29_{\pm 0.53}$	1.8
$p(\frac{T}{2}) = 0.9$	$2.23_{\pm 0.10}$	$99.69_{\pm 0.59}$	1.9	$14.79_{\pm 0.04}$	$100.57_{\pm 0.72}$	1.9

Table 10: Testing error rate of WideResNet28-10 models on Cifar10 and Cifar100 datasets with Cutout regularization when training with two trigonometric scheduling functions.

	Cifar10			Cifar100		
$p(\frac{T}{2})$	Error[%]	Time[m]	$\hat{\eta_p}$	Error[%]	Time[m]	$\hat{\eta_p}$
SGD	$2.81_{\pm 0.07}$	$57.33_{\pm 0.71}$	1.0	$16.91_{\pm 0.10}$	$57.46_{\pm 0.77}$	1.0
SAM	$2.43_{\pm 0.13}$	$105.17_{\pm 0.94}$	2.0	$14.87_{\pm 0.16}$	$103.14_{\pm 1.12}$	2.0
$p_{cos2}(t)$	$2.35_{\pm 0.03}$	$81.49_{\pm 0.39}$	1.5	$15.02_{\pm 0.19}$	$80.83_{\pm 0.55}$	1.5
$p_{sin1}(t)$	$2.27_{\pm 0.04}$	$80.39_{\pm 1.53}$	1.5	$14.85_{\pm 0.17}$	$81.15_{\pm 0.53}$	1.5

C Other Experiment Results for VGG Models

Other than WideResNet28-10, we would also investigate another model architecture, namely the VGG16 [15] with batch normalization. From the previous results, we could see that the constant scheduling functions would already provide representative results. So here we would only investigate the results when trained with constant scheduling functions. Table 11 shows the results, where the running time is reported on one A100 Nvidia GPU. We could see in the table that SS-SAM again could boost the computational efficiency and in the meantime acquire better model generalization compared to that trained using the SAM scheme.

Table 11: Testing error rate of VGG16-BN models on Cifar10 and Cifar100 datasets when training with constant scheduling functions.

	Cifar10			Cifar100		
a_c	Error[%]	Time[m]	$\hat{\eta_c}$	Error[%]	Time[m]	$\hat{\eta_c}$
SGD	$5.74_{\pm 0.09}$	$12.18_{\pm 0.21}$	1.0	$25.22_{\pm 0.31}$	$12.07_{\pm 0.37}$	1.0
SAM	$5.24_{\pm 0.08}$	$20.41_{\pm 0.57}$	2.0	$24.23_{\pm 0.29}$	$20.33_{\pm 0.68}$	2.0
$a_c = 0.1$	$5.49_{\pm 0.08}$	$13.04_{\pm 0.21}$	1.1	$24.90_{\pm 0.21}$	$12.93_{\pm 0.19}$	1.1
$a_c = 0.2$	$5.25_{\pm 0.07}$	$13.66_{\pm 0.14}$	1.2	$24.83_{\pm 0.17}$	$13.77_{\pm 0.39}$	1.2
$a_c = 0.3$	$5.19_{\pm 0.07}$	$14.40_{\pm 0.30}$	1.3	$24.08_{\pm 0.12}$	$14.53_{\pm 0.17}$	1.3
$a_c = 0.4$	$5.29_{\pm 0.03}$	$15.25_{\pm 0.42}$	1.4	$24.23_{\pm 0.20}$	$15.41_{\pm 0.23}$	1.4
$a_c = 0.5$	$5.05_{\pm 0.05}$	$16.11_{\pm 0.37}$	1.5	$23.83_{\pm 0.17}$	$16.18_{\pm 0.32}$	1.5
$a_c = 0.6$	$5.15_{\pm 0.02}$	$17.14_{\pm 0.52}$	1.6	$24.12_{\pm 0.11}$	$17.16_{\pm 0.28}$	1.6
$a_c = 0.7$	$4.97_{\pm 0.04}$	$17.72_{\pm 0.22}$	1.7	$23.86_{\pm 0.18}$	$18.03_{\pm 0.47}$	1.7
$a_c = 0.8$	$4.89_{\pm 0.05}$	$18.56_{\pm 0.56}$	1.8	$23.97_{\pm 0.14}$	$18.72_{\pm 0.37}$	1.8
$a_c = 0.9$	$4.91_{\pm 0.06}$	$19.66_{\pm 0.31}$	1.9	$23.81_{\pm 0.15}$	$19.62_{\pm 0.29}$	1.9

D Training Details

The basic training hyperparameters are deployed as below,

Table 12: The basic hyperparameters for training.

	Cifar10 & Cifar100
Epoch	200
Batch size	256
Basic learning rate	0.1
Learning rate schedule	cosine
Weight decay	0.001
ρ in SAM and SS-SAM	0.1