OPT2022: 14th Annual Workshop on Optimization for Machine Learning

Improved Deep Neural Network Generalization Using m-Sharpness-Aware Minimization

Kayhan Behdin[*] BEHDIN1675@GMAIL.COM
LinkedIn Corporation, USA
(Massachusetts Institute of Technology, USA)

Qingquan Song QSONG@LINKEDIN.COM
Aman Gupta AMAGUPTA@LINKEDIN.COM
David Durfee DDURFEE@LINKEDIN.COM
Ayan Acharya AYACHARYA@LINKEDIN.COM
Sathiya Keerthi KESELVARAJ@LINKEDIN.COM
LinkedIn Corporation, USA

Rahul Mazumder[†] RMAZUMDER@LINKEDIN.COM
LinkedIn Corporation, USA
(Massachusetts Institute of Technology, USA)

Abstract

Modern deep learning models are over-parameterized, where the optimization setup strongly affects the generalization performance. A key element of reliable optimization for these systems is the modification of the loss function. Sharpness-Aware Minimization (SAM) modifies the underlying loss function to guide descent methods towards flatter minima, which arguably have better generalization abilities. In this paper, we focus on a variant of SAM known as mSAM, which, during training, averages the updates generated by adversarial perturbations across several disjoint shards of a mini-batch. Recent work suggests that mSAM can outperform SAM in terms of test accuracy. However, a comprehensive empirical study of mSAM is missing from the literature—previous results have mostly been limited to specific architectures and datasets. To that end, this paper presents a thorough empirical evaluation of mSAM on various tasks and datasets. We provide a flexible implementation of mSAM and compare the generalization performance of mSAM to the performance of SAM and vanilla training on different image classification and natural language processing tasks. We also conduct careful experiments to understand the computational cost of training with mSAM, its sensitivity to hyperparameters and its correlation with the flatness of the loss landscape. Our analysis reveals that mSAM yields superior generalization performance and flatter minima, compared to SAM, across a wide range of tasks without significantly increasing computational costs.

1. Introduction

In recent years, overparameterized deep neural networks (DNNs) have become a staple of modern machine learning advancements and crucial to many domains. In particular, overparameterization has been successful in furthering the state-of-the-art performance for domains like image

[*] This work was done when Kayhan Behdin was an intern at LinkedIn during summer 2022.
[†] This work was done when Rahul Mazumder was a consultant for LinkedIn.

understanding [11, 16, 23], natural language processing (NLP) [4, 17, 24] and recommender systems [9, 20].

Training DNNs requires minimizing complex and non-convex loss functions with an abundance of local minima. Interestingly, different local minima can have varying loss values and generalizability on unseen data. Thus, it is essential to carefully choose an optimization scheme that can seek out minima that yield strong generalization performance. In recent years, a wide range of optimization algorithms has been developed for various domains, e.g. stochastic gradient descent (SGD), heavy-ball momentum [22], Adam [14], LAMB [30] among others. These methods help yield strong generalization performance when coupled with suitable regularization techniques.

Recently, extensive work has been done to study the correlation between the geometry of the loss landscape and generalization performance [6, 10, 13, 21, 28, 29]. The newly proposed Sharpness-Aware Minimization (SAM) algorithm [7] leverages this correlation between the flatness of minima and generalization performance by seeking to obtain flatter solutions during training, leading to superior generalization for a panoply of tasks and domains. In this paper, we focus on a variant of SAM called mSAM [7]. Instead of using a single adversarial perturbation for the entire mini-batch mSAM executes SAM-like training by averaging the updates generated by adversarial perturbations across several disjoint shards of a mini-batch. The name mSAM stems from the usage of m such disjoint shards. SAM represents flatness by looking at what happens in one particular adversarial direction, whereas mSAM leverages several such directions; which may better represent flatness.

Related Work: Although the sharpness of the loss landscape can be calculated using several different measures (for example, the largest eigenvalue [27] or trace [12] of the Hessian of the loss), most of these measures are computationally too expensive for practical purposes. The main idea behind the SAM [7] algorithm is to encourage the network to seek regions where the worst loss value in a local neighbourhood is not too large (see Section 2 for more details). This proxy of sharpness lends itself to easy computation, unlike the measures of sharpness described earlier. SAM has sparked interest in sharpness-aware training, resulting in several variants [5, 19, 32].

In this paper, we focus on a less-understood variant of SAM, known as mSAM. While SAM uses a single adversarial perturbation for a mini-batch, mSAM divides the mini-batch into disjoint shards and computes updates using adversarial perturbations applied to each shard. The updates are then averaged, resulting in a single mSAM update. In [7], mSAM is used implicitly to reduce the computational cost of SAM by avoiding synchronization across multiple GPUs (referred to as "accelerators" hereinafter). Thus, each accelerator's shard of data ends up with its own adversarial perturbation. Recently, it has been observed via limited experimentation that mSAM results in better generalization performance [2, 3, 7]. Although the authors of [1] present mathematical expressions for mSAM, their analysis is primarily focused on a particular version of mSAM (see Section 2 for more details). The experiments are also limited to image classification tasks on small architectures. As a result, the current understanding of mSAM is limited, and a comprehensive empirical study of its performance on state-of-the-art architectures and datasets is missing from the literature.

Our Contributions: We conduct a thorough empirical study of the mSAM algorithm and perform the following explorations that extol the value of mSAM. **(i)** Starting from the mathematical description of mSAM, we present an explicit flexible implementation of mSAM that does not rely on accelerator synchronization and is compatible with any single/multi-accelerator setup. **(ii)** We conduct extensive experiments on a wide variety of tasks using images and NLP data, leveraging architectures like Convolutional Neural Networks (CNNs) and transformers. In our experiments,

mSAM consistently outperforms SAM and vanilla training. **(iii)** We also conduct careful experiments to understand the computational cost of training with mSAM, its sensitivity to hyperparameters like m and its correlation with the flatness of the loss landscape. Our thorough empirical investigation reveals that mSAM substantially improves the generalization performance of SAM without significantly increasing the computational cost of training.

2. Algorithm

mSAM is based on the SAM algorithm that aims to obtain flat solutions to the empirical loss function. In particular, SAM tries to find a solution that minimizes the worst-case loss in a ball around the solution. Mathematically, let $\mathcal{S} = \{(x_i, y_i), i \in [n] : x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$ be a dataset of n samples, where $\mathcal{X}$ is the set of features and $\mathcal{Y}$ is the set of outcomes. Moreover, let $\ell : \mathbb{R}^d \times \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}$ be a differentiable loss function, where d is the number of model parameters. The empirical loss over the dataset $\mathcal{S}$ is defined as $\mathcal{L}_{\mathcal{S}}(w) = \sum_{i=1}^{n} \ell(w; x_i, y_i)/n$, where w parameterizes the neural network. With this notation in place, the SAM loss function is defined as [7]:

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

for some $p \geq 1$. In this work, we use $p = 2$. In practice, however, the maximization step in (1) cannot be done in closed-form. Hence, authors in [7] use a first-order approximation to $\mathcal{L}_S$ to simplify (1) as

$$\mathcal{L}_{\mathcal{S}}^{SAM}(w) \approx \max_{\|\epsilon\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(w) + \epsilon^\top \nabla \mathcal{L}_S(w). \tag{2}$$

It is easy to see that the maximum in Problem (2) is achieved for

$$\hat{\epsilon} = \rho \nabla \mathcal{L}_{\mathcal{S}}(w) / \|\nabla \mathcal{L}_{\mathcal{S}}(w)\|_2. \tag{3}$$

As a result, $\mathcal{L}_{\mathcal{S}}^{SAM} \approx \mathcal{L}_{\mathcal{S}}(w + \hat{\epsilon})$. This leads to the gradient

$$\nabla \mathcal{L}_{\mathcal{S}}^{SAM}(w) \approx \nabla_w \left[\mathcal{L}_{\mathcal{S}}(w + \hat{\epsilon})\right] = \frac{\partial (w + \hat{\epsilon})}{\partial w} \nabla \mathcal{L}_{\mathcal{S}}(w + \hat{\epsilon}). \tag{4}$$

However, calculating $\partial(w+\hat{\epsilon})/\partial w$ in (4) involves second order terms that require access to Hessian, which can be computationally inefficient in practice. Thus, by ignoring second order terms in (4), gradient of the SAM loss, is approximated as [7]:

$$\nabla \mathcal{L}_{\mathcal{S}}^{SAM}(w) \approx \nabla \mathcal{L}_{\mathcal{S}}(w + \rho \nabla \mathcal{L}_{\mathcal{S}}(w) / \|\nabla \mathcal{L}_{\mathcal{S}}(w)\|_2) \tag{5}$$

which is used in the SAM algorithm (for example, in conjunction with SGD). We refer to [7] for more details and intuitions about SAM. We call the inner gradient calculations on the right-hand side of (5) as the SAM ascent step and the outer gradient calculations as the gradient step. mSAM [7] is a variation of the SAM algorithm. In general, for mSAM, a minibatch of data $\mathcal{S}$ is further divided into m smaller disjoint shards (aka "micro-batches"), such as $\mathcal{S}_1, \cdots, \mathcal{S}_m$ where $\cup_{i=1}^{m} \mathcal{S}_i = \mathcal{S}$. For simplicity, we assume $|\mathcal{S}_1| = \cdots = |\mathcal{S}_m| = |\mathcal{S}|/m$ although such an assumption is not necessary in general. The mSAM loss is a variation of the SAM loss, defined as:

$$\mathcal{L}_{\mathcal{S}}^{mSAM}(w) = \frac{1}{m} \sum_{i=1}^{m} \max_{\|\epsilon^{(i)}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}_i}(w + \epsilon^{(i)}). \tag{6}$$

Table 1: Comparison of SAM with Different mSAM Implementations

	SAM	mSAM		
Loss function	$\max_{\|\epsilon\|_2 \leq \rho} \sum_{i=1}^{m} \mathcal{L}_{\mathcal{S}_i}(w+\epsilon)/m$	$\sum_{i=1}^{m} \max_{\|\epsilon^{(i)}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}_i}(w+\epsilon^{(i)})/m$		
Ascent step	$\hat{\epsilon} \propto \rho \sum_{i=1}^{m} \nabla \mathcal{L}_{\mathcal{S}_i}(w)/m$	$\hat{\epsilon}^{(i)} \propto \rho \nabla \mathcal{L}_{\mathcal{S}_i}(w),\ i \in [m]$		
Gradient	$g = \sum_{i=1}^{m} \nabla \mathcal{L}_{\mathcal{S}_i}(w+\hat{\epsilon})/m$	$g = \sum_{i=1}^{m} \nabla \mathcal{L}_{\mathcal{S}_i}(w+\hat{\epsilon}^{(i)})/m$		
Implementations	[7]	[7]	[1]	Ours
Possible m values	-	# of accelerators	flexible	flexible
Processor support	Multiple	Multiple	Single	Multiple

Intuitively, mSAM is a version of SAM where the ascent step (or the weight perturbation) of SAM is done independently on each micro-batch using different $\epsilon^{(i)}$, instead of using an average perturbation such as ϵ for all micro-batches. The mSAM gradient can thereby be derived as:

$$\nabla \mathcal{L}_{\mathcal{S}}^{mSAM}(w) = \frac{1}{m} \sum_{i=1}^{m} \nabla \mathcal{L}_{\mathcal{S}_i}(w + \rho \nabla \mathcal{L}_{\mathcal{S}_i}(w)/\|\nabla \mathcal{L}_{\mathcal{S}_i}(w)\|_2), \tag{7}$$

where (7) is a first-order approximation to the gradient of (6). We also note that the loss (6) is related to the mSAM definition of [1]. See Table 1 for a side-by-side comparison of SAM and mSAM, and their different implementations.

An important distinction between our work and prior work is that we treat m as a model hyper-parameter to improve generalization. In particular, in mSAM implementation of [7], the value of m is fixed to the number of hardware accelerators, micro-batch i is the part of the data that is loaded onto accelerator i, and each accelerator uses a separate perturbation, simulating the effect of mSAM. With this implementation, m is an artefact of the hardware setup. On the other hand, the analysis of [1] mostly concerns the value $m = |\mathcal{S}|$. In contrast, we consider a wide range of values for m in our experiments—this offers the flexibility to choose an appropriate value of m that leads to a better generalization performance. Moreover, our implementation supports any single/multi-accelerator setup and allows the user to set an appropriate value of m.

3. Numerical Experiments

This section compares mSAM to SAM and vanilla optimization methods (i.e. without sharpness-aware modification) on various model architectures and datasets. We report the average and standard deviation of accuracy on the test data over five independent runs. We also note that we use the same values of hyper-parameters for all algorithms, where ρ is chosen based on the best validation performance for SAM and other hyper-parameters are chosen based on the best validation error for vanilla methods. Moreover, although it is possible to use different values of ρ for each micro-batch in mSAM, doing so requires tuning numerous hyper-parameters (as we usually take m to be large) which is computationally infeasible. Therefore, we use the same value of ρ for all micro-batches.

Table 2: Accuracy Results for CNN Architectures

Dataset	Model	Vanilla	SAM	mSAM
CIFAR 10	ResNet50	95.45 ± 0.10	96.09 ± 0.11	96.40 ± 0.06
	WRN-28-10	95.92 ± 0.12	96.90 ± 0.05	96.95 ± 0.04
	ViT-B/16	97.68 ± 0.03	97.75 ± 0.06	98.29 ± 0.08
CIFAR 100	ResNet50	80.68 ± 0.13	81.49 ± 0.18	83.37 ± 0.10
	WRN-28-10	81.01 ± 0.19	82.93 ± 0.13	84.07 ± 0.06
	ViT-B/16	88.02 ± 0.17	88.75 ± 0.07	89.00 ± 0.17

3.1. Image Classification

In our first set of experiments on image classification datasets, we compare the performance of mSAM, SAM and vanilla methods with multiple CNN architectures such as ResNets [11] and WideResNet [31]. We use CIFAR10/100 [15] datasets as our test bed.

The average accuracies for the experiments with CIFAR data are reported in Table 2. These experiments are done on four NVidia V100 GPUs, with an effective batch size of 512 and the value of $m = 32$ for mSAM. Other hyper-parameters used to produce these results can be found in Appendix B. Overall, mSAM leads to better accuracy than SAM and vanilla methods in all cases reported in Table 2. In particular, mSAM achieves the best improvement in CIFAR100 data (about 2% on ResNet50 and about 1% on WRN-28-10) which is a more challenging dataset compared to CIFAR10.

Following the recent results that suggest that sharpness-aware optimization can improve the training quality of ViTs [3], we conduct additional experiments on a ViT architecture. In particular, we use the pre-trained ViT-B/16 checkpoint from [26] and fine-tune the model on CIFAR10/100 data independently. The average accuracy results for ViT fine-tuning are reported in Table 2. Similar to the CNN case, mSAM leads to better results than SAM that improves upon vanilla optimization further. We note that in these experiments, $m = 32$ is fixed for mSAM.

3.2. NLP Fine-tuning

Our next set of experiments is based on NLP analysis. We select four tasks from GLUE benchmark [25]. In particular, we choose COLA and MRPC as two small datasets and SST-2 and QQP as two larger datasets for empirical evaluation. The fine-tuning experiments with the RoBERTa-base model [18] are performed on four NVidia V100 GPUs with an effective batch size of 32. For the ease of reproduction of the results, we tabulate all the hyper-parameters used in Appendix C. For the fine-tuning experiments, we report the average value of Matthews Correlation Coefficient for COLA, and average accuracy for other datasets in Table 3. Overall, mSAM performs better than the baseline methods on these datasets. However, the variance among different runs is comparably high for smaller datasets such as COLA and MRPC. On the other hand, the results on larger data such as SST-2 and QQP are expectedly more robust across different runs.

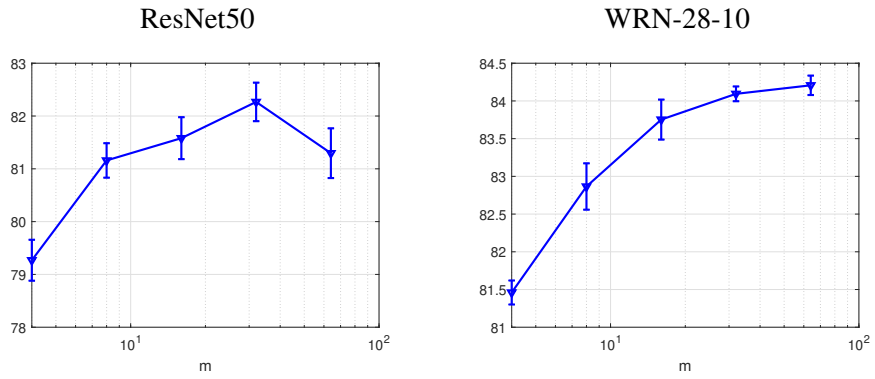


Figure 1: Effect of varying m

Table 3: Accuracy Results for GLUE Tasks

Task	Vanilla	SAM	mSAM ($m = 8$)
COLA	63.66 ± 2.46	64.30 ± 0.49	64.57 ± 0.66
MRPC	89.79 ± 0.05	90.37 ± 0.13	90.92 ± 0.16
SST-2	94.27 ± 0.18	95.21 ± 0.12	95.38 ± 0.10
QQP	91.70 ± 0.11	92.13 ± 0.02	92.18 ± 0.03

Table 4: $\lambda_{\max}$ for CNNs

Model	Vanilla	SAM	mSAM
ResNet50	26 ± 2	21 ± 3	18 ± 1
WRN-28-10	92 ± 4	30 ± 2	17 ± 1

4. A Deeper Investigation of mSAM

To further understand the mSAM algorithm, we design and report some experiments in this section. Additional experimental results are moved to the Appendix A.

Effect of varying m: In our experiments, we have observed that a larger value of m often leads to better test accuracy. We recover SAM by setting $m = 1$, which produces inferior results. To test this hypothesis, we set up the experiments with the CIFAR100 dataset on two CNNs, ResNet50 and WRN-28-10 in the same setup as in Section 3.1. We run mSAM for different values of $m \in \{4, 8, 16, 32, 64\}$. The accuracy results for these experiments are shown in Figure 1.

Increasing m improves the performance up to $m \approx 32$. However, a value of m larger than this threshold either leads to worse performance or marginal improvements, so increasing m does not necessarily result in better generalization. Intuitively speaking, when the micro-batch is too small, the perturbation derived according to the micro-batch might not be a good estimate of the actual SAM perturbation, leading to worse performance. We leave the theoretical analysis of such a phenomenon an interesting direction for future research. We also note that understanding how the optimal value of m and batch size interact is an open question left for future work.

Are mSAM solutions flat? The SAM algorithm hypothesizes that flat solutions generalize better. Since mSAM consistently outperforms SAM, it is worth investigating if mSAM settles for even flatter solutions. To that end and to quantify sharpness, we calculate the largest eigenvalue of the Hessian of the loss function $\mathcal{L}_{\mathcal{S}}(w)$ at the final solution, denoted as $\lambda_{\max}$. This metric is widely accepted to be a good indicator of the sharpness/flatness of a solution [27]. We calculate $\lambda_{\max}$ over the full train data using the power iteration, as implemented by [8]. We use the ResNet50 and WRN-28-10 models trained on CIFAR100 (see Section 3.1) to calculate $\lambda_{\max}$. The average results

for these experiments are reported in Table 4. We see that mSAM leads to solutions with smaller $\lambda_{\max}$ than SAM and vanilla SGD, confirming our conjecture.

mSAM runtime: A general misconception about mSAM is that it is computationally inefficient, as the total number of forward-backward passes in the network is multiplied by m [2]. However, note that these passes are performed on micro-batches, which are m times smaller than the actual minibatch. Hence, the overall computational cost gets amortized and is never as high as m times the cost of SAM. In practice, on large networks, the runtime of mSAM is only 1.2-1.3 times more compared to SAM. In Appendix A, we present numerical evidence for mSAM efficiency and discuss a few hybrid algorithms to reduce the computational cost of mSAM even further.

5. Discussion

The empirical study presented in this paper shows that mSAM outperforms SAM on different datasets (image classification and NLP tasks) and model architectures (CNNs and transformers); the exact performance gap is primarily a function of the data and the architecture. Moreover, our experiments suggest that mSAM does not necessarily incur a substantially higher computational cost than SAM, making it amenable for large-scale problems. An exciting avenue of work is the theoretical justification of better generalization abilities of mSAM. Our analysis suggests that mSAM leads to flatter solutions than SAM, which may explain the better generalization performance. However, the formal proof regarding why mSAM promotes a flatter minimum is left for future work.

6. Acknowledgements

Kayhan Behdin contributed to this work while he was an intern at LinkedIn during summer 2022. This work is not a part of his MIT research. Rahul Mazumder contributed to this work while he was a consultant for LinkedIn (in compliance with MIT's outside professional activities policies). This work is not a part of his MIT research.

References

[1] Maksym Andriushchenko and Nicolas Flammarion. Towards understanding sharpness-aware minimization. In International Conference on Machine Learning, pages 639–668. PMLR, 2022.

[2] Dara Bahri, Hossein Mobahi, and Yi Tay. Sharpness-aware minimization improves language model generalization. arXiv preprint arXiv:2110.08529, 2021.

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

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

[5] Jiawei Du, Daquan Zhou, Jiashi Feng, Vincent YF Tan, and Joey Tianyi Zhou. Sharpness-aware training for free. arXiv preprint arXiv:2205.14083, 2022.

[6] 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.

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

[8] Noah Golmant, Zhewei Yao, Amir Gholami, Michael Mahoney, and Joseph Gonzalez. pytorch-hessian-eigenthings: efficient pytorch hessian eigendecomposition, October 2018.

[9] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. arXiv preprint arXiv:1703.04247, 2017.

[10] Jeff Z. HaoChen, Colin Wei, Jason Lee, and Tengyu Ma. Shape matters: Understanding the implicit bias of the noise covariance. In Proceedings of Thirty Fourth Conference on Learning Theory, volume 134, pages 2315–2357, 2021.

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

[12] Hikaru Ibayashi, Takuo Hamaguchi, and Masaaki Imaizumi. Minimum sharpness: Scale-invariant parameter-robustness of neural networks. arXiv preprint arXiv:2106.12612, 2021.

[13] 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.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

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

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6):84–90, 2017.

[17] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.

[19] Yong Liu, Siqi Mai, Xiangning Chen, Cho-Jui Hsieh, and Yang You. Towards efficient and scalable sharpness-aware minimization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12360–12370, 2022.

[20] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. arXiv preprint arXiv:1906.00091, 2019.

[21] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. In International Conference on Learning Representations, 2018.

[22] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In International conference on machine learning, pages 1139–1147. PMLR, 2013.

[23] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning, pages 6105–6114. PMLR, 2019.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[25] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[26] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision, 2020.

[27] Lei Wu, Chao Ma, and Weinan E. How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective. In Advances in Neural Information Processing Systems, volume 31, 2018.

[28] Lei Wu, Mingze Wang, and Weijie Su. When does sgd favor flat minima? a quantitative characterization via linear stability, 2022. URL https://arxiv.org/abs/2207.02628.

[29] Zeke Xie, Issei Sato, and Masashi Sugiyama. A diffusion theory for deep learning dynamics: Stochastic gradient descent exponentially favors flat minima. In International Conference on Learning Representations, 2021.

[30] 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.

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

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

Appendix A. mSAM and Computational Efficiency

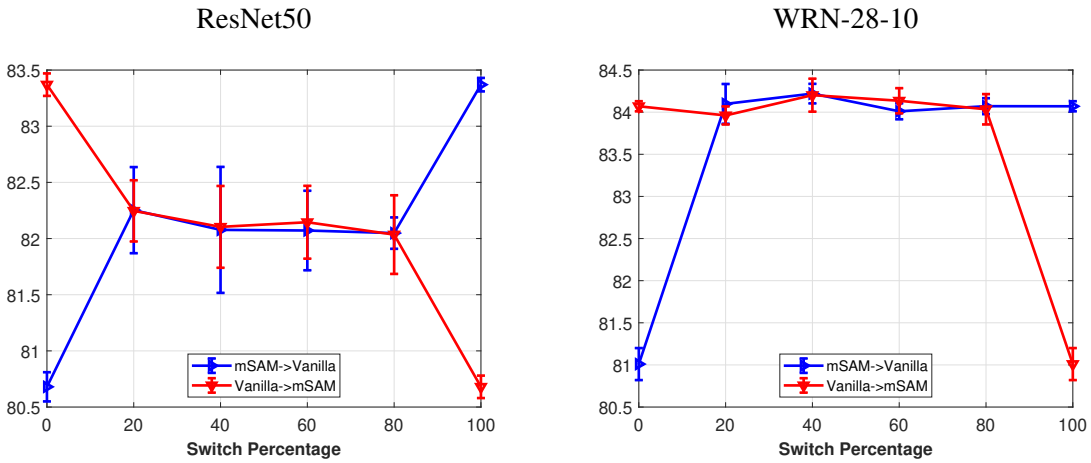
Since SAM requires two forward-backwards passes for each batch of data, SAM is almost twice as slow as vanilla training. In our experiments, mSAM appears to be slower than SAM, although not m times slower, as suggested by, for example [2]. To demonstrate that, we report the average runtime of our experiments from Section 3.1 on CIFAR100 data in Table A.1. Expectedly, SAM is twice as slow as the vanilla method. Interestingly, in the worst case, mSAM is only twice as slow as SAM, and in the best case, the computational penalty is only about 10% of SAM. The primary reason is that the overall computation complexity of forward-backwards passes for mSAM and SAM is not too different. In each epoch, mSAM performs m times more passes on micro-batches that are m times smaller. Encouragingly, mSAM appears more efficient on larger architectures such as ViT-B/16 and WRN-28-10.

Although mSAM does not appear to be computationally prohibitive in our experiments, it is still not as efficient as vanilla training, leaving room for further improving its efficiency. To that end, we conduct the following set of experiments. Building on our CIFAR100 experiments from Section 3.1, we start the training either with mSAM or vanilla training and then switch to the other training algorithm at some point. We keep all other training parameters fixed. The accuracy results for this setup for ResNet50 and WRN-28-10 are reported in Figure A.1. In this figure, the switch percent is the threshold in training when we transition from one algorithm to the other. For example, for the switch percent of 20, if we start with mSAM, we use mSAM for the first 20% of epochs and vanilla updates for the rest. If mSAM is used for the initial and/or final part of training, the accuracy is always better than vanilla training. In fact, in the WRN-28-10 case, as long as we partially use mSAM, the accuracy is almost the same as training with mSAM for the entire duration. For ResNet50, not using mSAM for the whole training leads to a drop in performance; however, even in this case, the accuracy of the hybrid training is better than the SAM training. These observations suggest that it is possible to enjoy the superior performance of mSAM, at least to some degree, while not having to deal with the computational complexity of mSAM for the entire training. A better theoretical and empirical understanding of the hybrid training method can be an exciting avenue for future work.

Table A.1: Runtime of different methods and architectures on CIFAR100 data

Model	Vanilla	SAM	mSAM
ResNet50	4497 ± 11	7440 ± 9	16196 ± 77
WRN	10675 ± 18	17483 ± 40	22261 ± 24
ViT-B/16	4349 ± 21	7007 ± 43	8163 ± 14

Figure A.1: Effect of switching training algorithm



Appendix B. Hyper-parameters for Image Classification Experiments

As mentioned, our experiments on CIFAR data in this section are done on 4 Nvidia V100 GPUs, with effective batch size of 512. For mSAM, we have used the micro-batch size of 16 which corresponds to $m = 32$. Rest of the hyper-parameters are chosen as in Table B.1 for CNNs, and as in Table B.2 for ViT experiments.

Appendix C. Hyper-parameters for GLUE Experiments

Similarly, our experiments in this section are done on 4 Nvidia V100 GPUs, with effective batch size of 32. For mSAM, we have used the microbatch size of 4 which corresponds to $m = 8$. The rest of task-specific hyper-parameters can be found in Table C.1.

Table B.1: Hyper-parameters for CNN experiments

Model	ResNet50	WRN-28-10
Optimizer	SGD	
Peak Learning Rate	0.5	0.75
Number of epochs	200	
Momentum	0.9	
Weight Decay	5×10^{-4}	
Label Smoothing	0.1	
Learning Rate Schedule	One cycle with 5% warm-up	
ρ (SAM/mSAM)	0.2	

Table B.2: Hyper-parameters for ViT experiments

Model	ViT-B/16
Optimizer	AdamW
Peak Learning Rate	10^{-3}
Number of epochs	20
Weight Decay	0.3
Learning Rate Schedule	One cycle with 5% warm-up
Gradient Clipping	norm=1
ρ (SAM/mSAM)	0.3

Table C.1: Hyper-parameters for NLP experiments

Task	COLA	MRPC	SST-2	QQP
Optimizer	AdamW			
Learning Rate	10^{-5}	10^{-5}	5×10^{-6}	2×10^{-5}
Learning Rate Schedule	One cycle with 6% warm-up			
Number of Epochs	60	60	20	15
Weight Decay	0.01			
ρ (SAM/mSAM)	0.01	0.01	0.05	0.05