

---

# Penalizing Gradient Norm for Efficiently Improving Generalization in Deep Learning

---

Yang Zhao<sup>1</sup> Hao Zhang<sup>1</sup> Xiuyuan Hu<sup>1</sup>

## Abstract

How to train deep neural networks (DNNs) to generalize well is a central concern in deep learning, especially for severely overparameterized networks nowadays. In this paper, we propose an effective method to improve the model generalization by additionally penalizing the gradient norm of loss function during optimization. We demonstrate that confining the gradient norm of loss function could help lead the optimizers towards finding flat minima. We leverage the first-order approximation to efficiently implement the corresponding gradient to fit well in the gradient descent framework. In our experiments, we confirm that when using our methods, generalization performance of various models could be improved on different datasets. Also, we show that the recent sharpness-aware minimization method (Foret et al., 2021) is a special, but not the best, case of our method, where the best case of our method could give new state-of-art performance on these tasks. Code is available at <https://github.com/zhaoyang-0204/gnp>.

## 1. Introduction

Today’s powerful computation hardwares make it possible for training large-scale deep neural networks (DNNs) (Goyal et al., 2017; Han et al., 2017; Dosovitskiy et al., 2021). These DNNs typically have millions or even billions of parameters, completely far exceeding the amount of training samples. Due to such heavy parametrization, they are capable to provide larger hypothesis space with normally better solutions. But in the meantime, such a huge hypothesis space is also full of more minima with diverse generalization ability (Neysshabur et al., 2017). This makes

---

<sup>1</sup>Department of Electronic Engineering, Tsinghua University. Correspondence to: Hao Zhang <haozhang@tsinghua.edu.cn>, Yang Zhao <zhaoyan18@mails.tsinghua.edu.cn>.

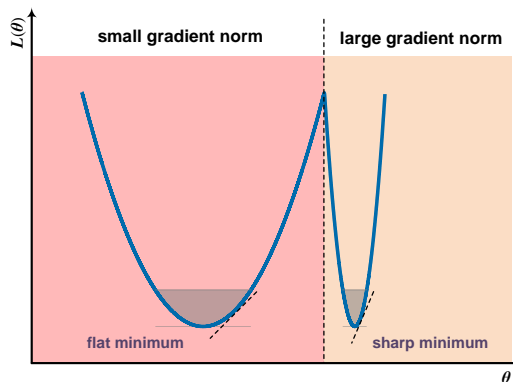


Figure 1. Toy example illustrating connections between the gradient norm of a function and the flatness of the function landscape.

it more challenging to train them to converge to optimal minima at which models would generalize better. Therefore, how to guide optimizers to find such optimal minima becomes a more salient concern than ever.

Generally, even if the given training datasets have been fully utilized, minimizing only the training loss gauging the gap between the true labels and predicted labels still could not ensure convergence to satisfactory minima. Regarding this, implementing regularization would play a critical role in modern training paradigm (Ioffe & Szegedy, 2015; Srivastava et al., 2014). Regularization techniques may contribute in various ways beyond datasets. In particular, regularizing models to have certain “good” properties could be one of the most commonly used techniques, typically implemented through penalty function methods (Smith et al., 1997).

In this paper, in addition to optimizing the common loss function, we would further impose an extra penalty on a specific property, the gradient norm of the loss function. The motivation of penalizing the gradient norm of loss function is to encourage the optimizer to find a minimum that lies in a relatively flat neighborhood region, since such flat minima have been demonstrated to be able to lead to better model generalization than sharp ones (Hochreiter & Schmidhuber, 1997). Figure 1 gives a toy example that illustrates the association between gradient norm and flatness of minima intuitively, and we would further demonstrate this from the perspective of Lipschitz continuity in Section 3.2.

Unfortunately, for practical implementation, optimizing the gradient norm in a straightforward way would involve the full calculation of Hessian matrix, which is not feasible for current hardwares. Here, by leveraging the approximation techniques, we present a simple and efficient scheme for computing the gradient of this gradient norm. The scheme would avoid the computation of the second-order derivative, and instead use basic algebraic operations between only the first-order derivatives for approximation, thus it could be implemented in practice easily. In particular, we find that the sharpness-aware minimization (SAM) scheme (Foret et al., 2021) is actually one special case of our scheme, where the hyperparameters are set to specific values.

In our experiments, we apply extensive model architectures on Cifar- $\{10, 100\}$  datasets and ImageNet datasets, respectively. These models would include both simple and complex convolutional neural network architectures, as well as the recent vision transformer architectures. We observe that the model performance could be generally improved via our optimization scheme, and such improvements could be up to 70% greater than SAM’s improvements over the standard training. Also, in some cases, training could be more stable when using our scheme compared to the SAM scheme. Finally, we provide a guide on hyperparameter selection in expectation to achieve the best improvements in practice.

## 2. Related Works

**Regularization techniques** Regularization could widely refer to techniques that in some way help improve the model generalization, including penalty function methods (Smith et al., 1997), data augmentation (Devries & Taylor, 2017; Cubuk et al., 2018), dropout regularizations (Srivastava et al., 2014; Wan et al., 2013), normalization techniques (Ioffe & Szegedy, 2015; Ba et al., 2016; Wu & He, 2018) and so on. For penalty function methods, extra terms would be added and optimized along with the loss function, which targets to impose constraint on specific property of models. In particular, the weight norm has been demonstrated to be an important property related to the model capacity (Neyshabur et al., 2017), and penalizing the weight  $L^2$ -norm (Krogh & Hertz, 1991; Loshchilov & Hutter, 2019) has become, in a sense, the essential ingredient in modern training recipes. Others like Yoshida & Miyato (2017) penalize the spectral norm of weights for reducing the models’ sensitivity to input perturbation.

**Flat minima** On the other hand, our work is also relevant to the research of flat minima. In Hochreiter & Schmidhuber (1997), the authors first point out that well generalized models may have flat minima. Since then, the association between flatness of minima and model generalization have been studied from both empirical (Keskar et al., 2017) and

theoretical perspectives (Dinh et al., 2017; Neyshabur et al., 2017). Although SGD optimizer and some of its variants (such as momentum) could somehow serve as implicit regularizations that favors flat minima (Goodfellow et al., 2016; Wu et al., 2018; Xie et al., 2021), researchers also desire to bias the optimizers in an explicit way in pursuit of smoother surface and flatter minima to further improve model performance, especially for modern scalable models. But in practical optimization, explicitly finding flat minima is non-trivial. Recently, Foret et al. (2021) treat it as a minimax optimization problem, and solve it by introducing an efficient procedure, called SAM. Model generalization could be improved significantly compared to using vanilla SGD optimizations. Further, based on Foret et al. (2021), Kwon et al. (2021) propose the Adaptive SAM, where optimization could keep invariant to a specific weight-rescaling operation discussed in (Neyshabur et al., 2015; Dinh et al., 2017); Zheng et al. (2021) perform gradient descent twice in one step to solve the corresponding minimax problem, one for the inner maximization optimization and the other for the outer minimization optimization.

## 3. Method

### 3.1. Basic Setting

Given a training dataset  $\mathcal{S} = \{(x_i, y_i)\}_{i=0}^n$  drawn i.i.d from the distribution  $\mathcal{D}$ , a neural network  $f(\cdot; \theta)$  is trained to learn this distribution. The neural network is parametrized with parameters  $\theta$  in weight space  $\Theta$ , which would be optimized via minimizing an empirical loss function  $L_{\mathcal{S}}(\theta) = \frac{1}{N} \sum_{i=1}^N l(\hat{y}_i, y_i, \theta)$  where  $\hat{y}_i = f(x_i; \theta)$  denoting the predicted label for input  $x_i$ .

When imposing penalty on the gradient norm of the loss function, a term with respect to it could be added on the loss function  $L_{\mathcal{S}}(\theta)$  simply,

$$L(\theta) = L_{\mathcal{S}}(\theta) + \lambda \cdot \|\nabla_{\theta} L_{\mathcal{S}}(\theta)\|_p \quad (1)$$

where  $\|\cdot\|_p$  denotes the  $L^p$ -norm and  $\lambda$  is the penalty coefficient and  $\lambda \in \mathbb{R}_+$  (in the experiment section, we also investigate the results where  $\lambda \in \mathbb{R}_-$ ). And for clarity, we would use  $L^2$ -norm ( $p = 2$ ) in the following demonstration since it is the most commonly used metric in deep learning.

### 3.2. Gradient Norm and Lipschitz Continuity

Generally, penalizing the gradient norm of loss function would motivate the loss function to have small Lipschitz constant in local. If the loss function has a smaller Lipschitz constant, it would indicate that the loss function landscape is flatter, which in consequence could lead to better model generalization.

Regarding the term "flat minima", it is a rather intuitive concept. Based on the description in (Hochreiter & Schmid-

huber, 1997), a flat minimum denotes "a large connected region in weight space where the error remains approximately constant". However, the mathematical descriptions may differ (Hochreiter & Schmidhuber, 1997; Neyshabur et al., 2017; Dinh et al., 2017; Keskar et al., 2017; Chaudhari et al., 2017), although they may convey similar core ideas. Here, we would only follow the basic concept in our demonstration.

We would start from the Lipschitz continuous. Given  $\Omega \subset \mathbb{R}^n$ , for function  $h : \Omega \rightarrow \mathbb{R}^m$ , it is called Lipschitz continuous if there exists a constant  $K$  that satisfies,

$$\|h(\theta_1) - h(\theta_2)\|_2 \leq K \cdot \|\theta_1 - \theta_2\|_2 \quad (2)$$

for  $\forall \theta_1, \theta_2 \in \Omega$ . And the Lipschitz constant generally refers to the smallest  $K$  of the function. Further, for  $\forall \theta \in \Omega$ ,  $h$  is locally Lipschitz continuous if  $\theta$  has a neighborhood  $\mathcal{A}$  that  $h|_{\mathcal{A}}$  is Lipschitz continuous.

Intuitively, the Lipschitz constant describes the upper bound on the output change in the input space  $\Omega$ . In particular, for  $h|_{\mathcal{A}}$ , it would indicate the supremum of output change in the neighborhood  $\mathcal{A}$ . In other words, for small Lipschitz constants, given any two points in  $\mathcal{A}$ , the gap between their outputs is limited to a small range. In fact, this is essentially a kind of description of "flat minima".

So for a minimum  $\theta_i$  and the loss function  $L(\theta)$ , according to the mean value theorem, the differentiability could lead to that for  $\forall \theta'_i \in \mathcal{A}$ ,

$$\|L(\theta'_i) - L(\theta_i)\|_2 = \|\nabla L(\zeta)(\theta'_i - \theta_i)\|_2 \quad (3)$$

where  $\zeta = c\theta_i + (1-c)\theta'_i$ ,  $c \in [0, 1]$ . And the Cauchy-Schwarz inequality gives,

$$\|L(\theta'_i) - L(\theta_i)\|_2 \leq \|\nabla L(\zeta)\|_2 \|\theta'_i - \theta_i\|_2 \quad (4)$$

When  $\theta'_i \rightarrow \theta$ , the corresponding Lipschitz constant approximates to  $\|\nabla L(\theta_i)\|_2$ . Therefore, we would expect to reduce  $\|\nabla L(\theta_i)\|_2$  to give small Lipschitz constants such that models could converge to flat minima.

Additionally, it should be especially discriminated that some works (Yoshida & Miyato, 2017; Virmaux & Scaman, 2018) try to regularize the Lipschitz constant of DNNs in the input space such that models would be more stable to the perturbation in the input space. This is not the same as penalizing the gradient norm of loss function, which would function in the weight space.

### 3.3. Gradient Calculation of Loss with Gradient Norm Penalty

During practical optimization, we need to calculate the gradient of current loss (Equation 1),

$$\nabla_{\theta} L(\theta) = \nabla_{\theta} L_S(\theta) + \nabla_{\theta} (\lambda \cdot \|\nabla_{\theta} L_S(\theta)\|_p) \quad (5)$$

Based on the chain rule, Equation 5 could be simplified as,

$$\nabla_{\theta} L(\theta) = \nabla_{\theta} L_S(\theta) + \lambda \cdot \nabla_{\theta}^2 L_S(\theta) \frac{\nabla_{\theta} L_S(\theta)}{\|\nabla_{\theta} L_S(\theta)\|} \quad (6)$$

In Appendix, we have provided detailed procedures for this simplification from Equation 5 to Equation 6.

Apparently, Equation 6 involves the calculation of Hessian matrix. For DNNs, it is infeasible to straightforwardly solve such a Hessian matrix since the dimension in weight space is too huge. Appropriate approximation method should be implemented in this calculation.

In Equation 6, the Hessian matrix is essentially a linear operator  $\mathbf{H}(\cdot)$  that functions on the corresponding gradient vector. Here, local Taylor expansion would be employed to approximate the operation results between the Hessian matrix and the gradient vector. From the Taylor expansion, we have

$$\nabla_{\theta} L_S(\theta + \Delta\theta) = \nabla_{\theta} L_S(\theta) + \mathbf{H}\Delta\theta + \mathcal{O}(\|\Delta\theta\|^2) \quad (7)$$

When choosing  $\Delta\theta = r\mathbf{v}$  where  $r$  is a small value and  $\mathbf{v}$  is a vector, Equation 7 would be,

$$\mathbf{H}\mathbf{v} = \frac{\nabla_{\theta} L_S(\theta + r\mathbf{v}) - \nabla_{\theta} L_S(\theta)}{r} + \mathcal{O}(r) \quad (8)$$

Further, assigning  $\mathbf{v} = \frac{\nabla_{\theta} L_S(\theta)}{\|\nabla_{\theta} L_S(\theta)\|}$ ,

$$\mathbf{H} \frac{\nabla_{\theta} L_S(\theta)}{\|\nabla_{\theta} L_S(\theta)\|} \approx \frac{\nabla_{\theta} L(\theta + r \frac{\nabla_{\theta} L_S(\theta)}{\|\nabla_{\theta} L_S(\theta)\|}) - \nabla_{\theta} L(\theta)}{r} \quad (9)$$

Now, based on Equation 9, Equation 6 would be,

$$\begin{aligned} \nabla_{\theta} L(\theta) &= \nabla_{\theta} L_S(\theta) \\ &+ \frac{\lambda}{r} \cdot (\nabla_{\theta} L_S(\theta + r \frac{\nabla_{\theta} L_S(\theta)}{\|\nabla_{\theta} L_S(\theta)\|}) - \nabla_{\theta} L_S(\theta)) \\ &= (1 - \alpha) \nabla_{\theta} L_S(\theta) \\ &+ \alpha \nabla_{\theta} L_S(\theta + r \frac{\nabla_{\theta} L_S(\theta)}{\|\nabla_{\theta} L_S(\theta)\|}) \end{aligned} \quad (10)$$

where  $\alpha = \frac{\lambda}{r}$ , and we would call  $\alpha$  the balance coefficient.

Accordingly, we need to set two basic parameters  $\lambda$  and  $r$  to perform gradient norm penalty. For  $\lambda$ , it denotes the penalty coefficient, which controls the degree of the regularization on the gradient norm. However, the connections between weight norm, gradient norm and model generalization is subtle during training. Currently, how much the gradient norm should be penalized in practical training still requires some further tuning effort. As for  $r$ , it is used for appropriating the Hessian multiplication operation (Equation 8). Notably,  $r$  should be set carefully here since it would directly affect

**Algorithm 1** Optimization Scheme of Penalizing Gradient Norm

**Input:** Training set  $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=0}^N$ ; loss function  $L(\cdot)$ ; batch size  $B$ ; learning rate  $\eta$ ; total step  $T$ ; balance coefficient  $\alpha$ ; approximation scalar  $r$ .

**Parameter:** Model parameters  $\theta$

**Output:** Optimized weight  $\hat{\theta}$

- 1: Parameter initialization  $\theta_0$ .
- 2: **for** step  $t = 1$  **to**  $T$  **do**
- 3: Get batch data pairs  $\mathcal{B} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=0}^B$  sampled from training set  $\mathcal{S}$ .
- 4: Calculate the gradient  $\mathbf{g}_1 = \nabla_{\theta} L_{\mathcal{S}}(\theta)$  based on the batch samples.
- 5: Add  $r \frac{\nabla_{\theta} L_{\mathcal{S}}(\theta)}{\|\nabla_{\theta} L_{\mathcal{S}}(\theta)\|}$  on the current parameter  $\theta_t$ , which makes  $\theta_t' = \theta_t + r \frac{\nabla_{\theta} L_{\mathcal{S}}(\theta)}{\|\nabla_{\theta} L_{\mathcal{S}}(\theta)\|}$ .
- 6: Calculate the gradient  $\mathbf{g}_2 = \nabla_{\theta} L_{\mathcal{S}}(\theta)$  at  $\theta = \theta_t'$ .
- 7: Calculate the final gradient  $\mathbf{g} = (1 - \alpha)\mathbf{g}_1 + \alpha\mathbf{g}_2$ .
- 8: (SGD optimizer) Update parameter with final gradient,  $\theta_{k+1} = \theta_k - \eta \cdot \mathbf{g}$ .
- 9: **end for**
- 10: **return** Final optimization  $\hat{\theta}$ .

the approximation precision (Pearlmutter, 1994). On the one hand,  $r$  is expected to be small enough such that the term  $\mathcal{O}(r)$  in approximation could be safely ignored. But on the other hand, as  $r$  becomes smaller, the perturbed weight  $\theta + rv$  will gradually weaken the effect of  $v$  and approach the reference weight  $\theta$ , which makes  $\nabla_{\theta} L_{\mathcal{S}}(\theta + rv)$  and  $\nabla_{\theta} L_{\mathcal{S}}(\theta)$  too close when appropriating  $Hv$ . Therefore, we should also avoid setting  $r$  too small to provide enough precision of  $v$ .

In practice, we would further take an approximation for computing the second term in Equation 10 to avoid the Hessian computation caused by the chain rule,

$$\nabla_{\theta} L_{\mathcal{S}}(\theta + r \frac{\nabla_{\theta} L_{\mathcal{S}}(\theta)}{\|\nabla_{\theta} L_{\mathcal{S}}(\theta)\|}) \approx \nabla_{\theta} L_{\mathcal{S}}(\theta)|_{\theta=\theta+r \frac{\nabla_{\theta} L_{\mathcal{S}}(\theta)}{\|\nabla_{\theta} L_{\mathcal{S}}(\theta)\|}} \quad (11)$$

In summary, Algorithm 1 gives the full procedures of our optimization scheme. It should be mentioned that Algorithm 1 only shows our scheme when using SGD optimizer. For other optimizers or update strategies (such as Adam), one could add specific operations before step 8.

Particularly, if  $\alpha = 1$ , it is actually the SAM optimization (Foret et al., 2021). This indicates that SAM is a special implementation of penalizing gradient norm, where the penalty coefficient  $\lambda$  is always set equal to  $r$ . However, given the distinct roles of the two parameters, we can not anticipate that models would achieve best performance every time at  $\lambda = r$ . Such a binding deployment, while reducing one parameter, also limits our tuning. We would further show that

SAM could not be the best implementation in the following experiment section.

## 4. Experiments

We would demonstrate the effectiveness of our proposed scheme by investigating performance on image classification tasks. In all our experiments, we would compare our scheme with two other training schemes, one is the standard training scheme and the other one is the SAM training scheme. Besides, all the experiments are deployed using the JAX framework on the NVIDIA DGX Station A100.

### 4.1. Cifar10 and Cifar100

In this section, we would use Cifar10 and Cifar100 as our experimental datasets, and would separately apply the convolutional neural network (CNN) architectures and the vision transformer (ViT) architecture for the corresponding benchmark performance tests.

**Convolutional neural network** For CNN architectures, five different architectures would be involved, including relatively simple architectures (VGG16 (Simonyan & Zisserman, 2015)) and complex architectures (WideResNet (Zagoruyko & Komodakis, 2016) and PyramidNet (Han et al., 2017)).

For training datasets, we would employ two kinds of augmentations. The first one is the basic augmentation, where samples are padded with additional four pixels on each boundary, randomly flipped in horizontal and then cropped randomly to size  $32 \times 32$ . The second one is the cutout augmentation, where cutout regularization (Devries & Taylor, 2017) would be implemented moreover based on the basic augmentation. Specifically, cutout regularization is a common data augmentation technique for CNNs, which randomly masks out a square region in the input with a given mask value (generally zero).

Our investigation would focus on the comparisons between three different training schemes, namely the standard SGD scheme, SAM scheme and our scheme. Considering the connections between these three training schemes, we would adopt a "greedy" strategy to reduce tuning cost during implementation,

1. We would first train models using the standard training scheme ( $\lambda = 0$  in our scheme), and record the common training hyperparameters (such as learning rate, weight decay) that could give the best model performance.
2. Based on the best hyperparameters recorded in Step 1, we would then set the balance coefficient  $\alpha = 1$  to train the same models using the SAM scheme, and perform a grid search on the parameter  $r$ .

Table 1. Testing error rate of CNN models on Cifar10 and Cifar100 when implementing the three training schemes.

	Cifar10		Cifar100	
VGG16	Basic	Cutout	Basic	Cutout
Standard	7.07	5.31	28.78	<b>26.98</b>
SAM	6.91	<b>6.17</b>	28.62	27.13
<b>Ours</b>	<b>6.72</b>	<b>5.19</b>	<b>28.48</b>	27.07
VGG16-BN	Basic	Cutout	Basic	Cutout
Standard	5.74 $\pm$ 0.09	4.39 $\pm$ 0.07	25.22 $\pm$ 0.31	24.69 $\pm$ 0.25
SAM	5.24 $\pm$ 0.08	4.16 $\pm$ 0.11	24.23 $\pm$ 0.29	23.35 $\pm$ 0.33
<b>Ours</b>	<b>4.88<math>\pm</math>0.12</b>	<b>4.02<math>\pm</math>0.08</b>	<b>24.04<math>\pm</math>0.18</b>	<b>23.07<math>\pm</math>0.26</b>
WideResNet-28-10	Basic	Cutout	Basic	Cutout
Standard	3.53 $\pm$ 0.10	2.81 $\pm$ 0.07	18.99 $\pm$ 0.12	16.92 $\pm$ 0.10
SAM	2.78 $\pm$ 0.07	2.43 $\pm$ 0.13	16.53 $\pm$ 0.13	14.87 $\pm$ 0.16
<b>Ours</b>	<b>2.52<math>\pm</math>0.09</b>	<b>2.16<math>\pm</math>0.11</b>	<b>16.02<math>\pm</math>0.19</b>	<b>14.28<math>\pm</math>0.16</b>
WideResNet-SS 2 $\times$ 96	Basic	Cutout	Basic	Cutout
Standard	2.82 $\pm$ 0.05	2.39 $\pm$ 0.06	17.19 $\pm$ 0.19	15.85 $\pm$ 0.14
SAM	2.37 $\pm$ 0.09	2.11 $\pm$ 0.13	15.22 $\pm$ 0.19	14.32 $\pm$ 0.15
<b>Ours</b>	<b>2.28<math>\pm</math>0.13</b>	<b>2.01<math>\pm</math>0.10</b>	<b>14.93<math>\pm</math>0.10</b>	<b>14.03<math>\pm</math>0.17</b>
PyramidNet-SD	Auto Aug + Cutmix		Auto Aug + Cutmix	
Standard	1.66 $\pm$ 0.11		10.83 $\pm$ 0.14	
SAM	1.41 $\pm$ 0.08		10.33 $\pm$ 0.13	
<b>Ours</b>	<b>1.30<math>\pm</math>0.07</b>		<b>10.12<math>\pm</math>0.17</b>	

- Finally, based on the previous information, we would further perform a grid search on the balance coefficient  $\alpha$  to adjust the penalty coefficient.

Basically, the involved model architectures have been extensively studied for the standard training scheme. It is not necessary to perform a heavy grid search, and we could just follow the common hyperparameters used in the related literatures. Next, we would perform a grid search on the scaler  $r$  over the set  $\{0.01, 0.02, 0.05, 0.1, 0.2\}$ . This setting is actually the same as it in (Foret et al., 2021). And for a fair comparison, we would train the models to reach at comparable results as reported in their paper. After determining the best value of  $r$ , we would moreover perform a grid search on the balance coefficient  $\alpha$  in the range 0.1 to 0.9 at an interval of 0.1.

For each model, we would train with five different random seeds, and record the convergence model performance on testing sets during training. And then we would report the mean value and the standard deviation, as shown in Table 1.

In table 1, totally five model cases are involved: the original VGG16 architecture and that with batch normalization regularization, the WideResNet-28-10 architecture and that

with Shake-Shake regularization (Gastaldi, 2017), and the PyramidNet-270 architecture with Shake-Drop regularization (Yamada et al., 2019). We could see that for our training scheme, the model performance could be improved to some extent compared to the other two schemes.

For the original VGG16 model, training could frequently fail with relatively large learning rate, sometimes all five trials may fail especially for the SAM scheme. However, small learning rate may not lead to the best performance. Here, we would simply report the best training case (possibly trained more than five times and not following the greed strategy strictly).

Basically, the results are quite close when using the three training schemes. But it should be highly noted about the results when implementing cutout augmentation. On Cifar10, the performance using SAM scheme may be much worse than that using standard scheme. This is because that for the optimal common training hyperparameters in the standard scheme, all of the values in our grid search on approximation scalar  $r$  fail to train from the start using SAM scheme. We have to lower the learning rate to stabilize the training. In contrast, by setting appropriate balance coefficient  $\alpha$ , our scheme could utilize the optimal common training hyper-

parameters, which could improve the performance slightly. However, on Cifar100, the standard training would yield the best performance.

When applying the batch normalization regularization on the VGG model, such training failures would be largely alleviated, although they may still happen in few trials. We would see that with batch normalization, VGG16 could receive significantly performance gains. The best performances are achieved via our scheme, which could be as low as 4% testing error rate on Cifar10.

As for the WideResNet architecture, we could find that our scheme could significantly improve the performance by 1% on Cifar10 and near 3% on Cifar100 compared to the standard training scheme. Our improvements could be 38% on average and up to remarkable 70% (on Cifar10 with cutout, ours is 0.65 while SAM’s is 0.38, which is  $0.65 = (1 + 71\%) \times 0.38$ ) more than the SAM’s improvements over the standard training scheme. This confirms the effectiveness of our scheme, and further demonstrates that SAM is not the best case in our scheme.

Regarding the WideResNet with Shake-Shake regularization (WideResNet-SS in the table), our improvements could not be as significant as that on the WideResNet architecture, but still give about 20% improvement compared to the SAM’s improvement over the standard training scheme.

Finally, we would investigate the PyramidNet architecture with the Shake-Drop regularization (PyramidNet-SD in the table). We would adopt the auto-augmentation policy (Cubuk et al., 2018) and the cutmix regularization (Yun et al., 2019) for data augmentation. Here, only three random seeds are used for training. We could see that our scheme again improves the performance on both Cifar10 and Cifar100.

**Vision transformer** We would like to investigate the effectiveness of our scheme on the recent vision transformer architectures (Dosovitskiy et al., 2021). Our investigation would focus on the ViT-Ti16 and ViT-S16 architectures introduced in their paper.

Here, we would adopt the same greedy search strategy as in the previous section for the three training schemes. Likewise, five random seeds are used for each model. But for data augmentation, we would not use cutout augmentation here since we find such augmentation would not boost the model performance. Intuitively, the vision transformer architecture would cut the image into small patches, and utilize the relationship between these patches to make decisions. In this way, the cutout regularization may not be helpful for models to learn the relationship between patches. Here, we would replace the cutout augmentation with a heavy augmentation, considering that vision transformer architectures are generally data hungry models. In the heavy augmenta-

Table 2. Testing error rate of ViT models on Cifar10 dataset when implementing the three training schemes.

	Cifar10	
ViT-Ti16	Basic	Heavy
Standard	15.92 $\pm$ 0.17	14.68 $\pm$ 0.14
SAM	15.33 $\pm$ 0.18	13.77 $\pm$ 0.12
Ours	<b>14.75<math>\pm</math>0.17</b>	<b>13.52<math>\pm</math>0.21</b>
ViT-S16	Basic	Heavy
Standard	14.55 $\pm$ 0.14	13.31 $\pm$ 0.11
SAM	13.91 $\pm$ 0.18	12.63 $\pm$ 0.09
Ours	<b>13.66<math>\pm</math>0.16</b>	<b>12.29<math>\pm</math>0.19</b>

Table 3. Testing error rate of ViT models on Cifar100 dataset when implementing the three training schemes.

	Cifar100	
ViT-Ti16	Basic	Heavy
Standard	40.21 $\pm$ 0.20	38.93 $\pm$ 0.28
SAM	38.89 $\pm$ 0.23	37.61 $\pm$ 0.19
Ours	<b>38.58<math>\pm</math>0.27</b>	<b>37.15<math>\pm</math>0.21</b>
ViT-S16	Basic	Heavy
Standard	38.43 $\pm$ 0.19	37.58 $\pm$ 0.22
SAM	37.98 $\pm$ 0.23	36.77 $\pm$ 0.25
Ours	<b>37.32<math>\pm</math>0.28</b>	<b>36.59<math>\pm</math>0.22</b>

tion, we would perform a series of operations, including resizing to  $72 \times 72$ , random flipping, random rotating, random zooming, random cropping and finally resizing to  $48 \times 48$ . We would adopt the  $4 \times 4$  patch size in both the basic augmentation and the heavy augmentation. Models would be trained much longer using heavy augmentation than those using basic augmentation (1200 v.s 300 epochs). In addition, we would use extra operations like label smoothing and drop path as used in (Dosovitskiy et al., 2021) when using the heavy augmentation.

Table 2 & 3 presents the corresponding testing error rate. We could see that even if using heavy augmentation, the performances of vision transformer architectures would be much worse than those of CNN architectures. And in the table, we could find that the performances could be improved via implementing our scheme. This further confirms the broad applicability of our scheme.

**Parameter study** Further, we would investigate the impact on model performance as choosing different balance coefficients  $\alpha$  and approximation scalars  $r$  in our optimization scheme, which we would illustrate using the WideResNet-

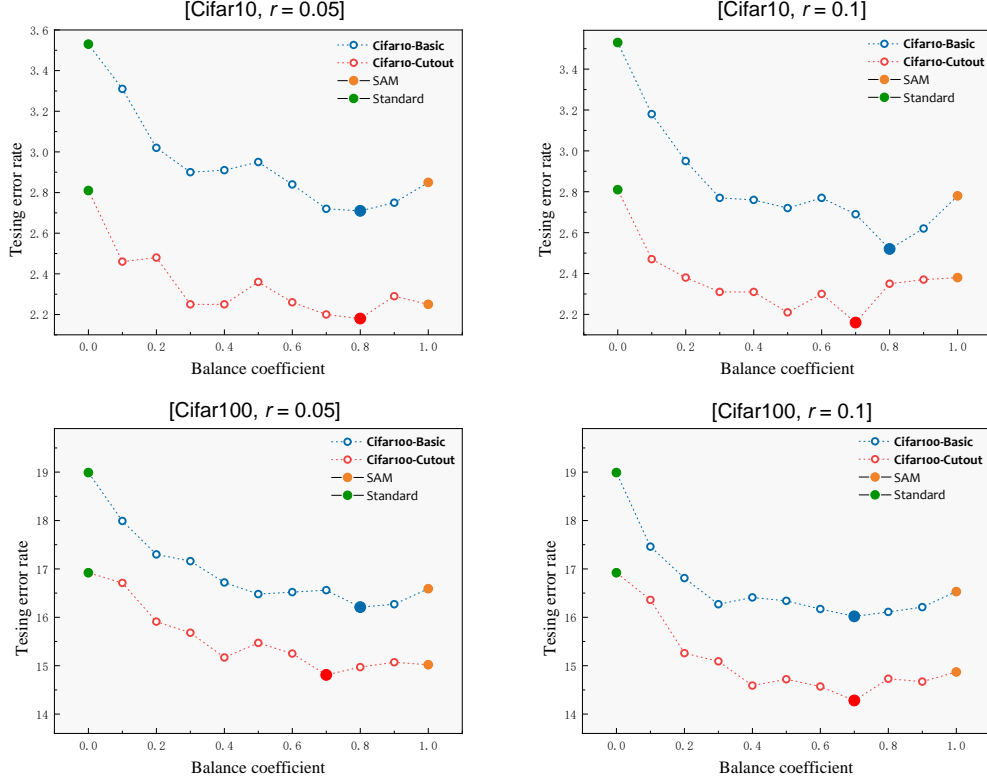


Figure 2. Testing error rate when trained with different hyperparameter  $r$  and  $\alpha$ . The upper row denotes the error rate on Cifar10 while the lower row denotes the error rate on Cifar100, and the left column denotes  $r = 0.05$  while the right column denotes  $r = 0.1$ . The points colored green are results using the standard scheme, and the points colored orange are results using the SAM scheme.

28-10 model architecture.

When performing the grid search on approximate value  $r$  in the SAM training experiments, we observe that models would have relatively better performances when setting  $r = 0.05$  and  $r = 0.1$ . This observation is the same as that in (Foret et al., 2021). Then, the grid search over balance coefficient  $\alpha$  is performed moreover based on  $r = 0.05$  and  $0.1$ . Figure 2 shows the results. In Figure 2, rows denote results on Cifar10 and Cifar100, respectively, and blue lines in the plots denote adopting basic data augmentation while red lines denote adopting cutout data augmentation. As we would see in the figure, from the standard training scheme ( $\alpha = 0$ ) to the SAM training scheme ( $\alpha = 1$ ), each curve may experience a decrease and then an increase in the testing error rate. Based on the figure, we could find that these models would achieve best performance when the balance coefficient  $\alpha$  is set around 0.7 or 0.8.

In addition to the basic set in the previous deployment of  $\alpha$ , we would like to further investigate cases where  $\alpha \notin [0, 1]$ . Extra deployments would be implemented over two other sets,  $\mathcal{B}_a = \{-0.1, -0.2, -0.5\}$  and  $\mathcal{B}_b = \{1.1, 1.2, 1.3, 1.5, 2.0\}$ .

For  $\mathcal{B}_a$ , since its values are all negative, this causes that the penalty coefficient of gradient norm in Equation 1 becomes negative, which makes it a reward as increasing the gradient norm during optimization. Generally, if  $\lambda \leq 0$ , the optimization would not be fully ensured, since we are adding a negative term on the loss. And in all of our trials, no matter trained on Cifar10 or Cifar100 datasets, the models completely fail to converge even if gradient clip regularization is adopted. The gradient would be unstable, and may even explode immediately after the training start. But this instead shows the effectiveness of our penalty scheme.

As for  $\mathcal{B}_b$ , since the values in it are all greater than 1.0, the penalty on the gradient norm becomes larger, and the operation relationship in Equation 10 shifts from addition to subtraction. This may somehow be harmful to training, as shown in figure 3. In the figure, the left plot denotes the gradient norm of loss function with respect to the training epochs, while the right plot denotes the testing error rate on Cifar10 datasets.

We could find that these large coefficients indeed impose much heavier penalties on the gradient norm of loss function. The gradient norm would drop faster and faster as  $\alpha$  increases from 1.1 to 2.0. For  $\alpha = 2.0$ , the gradient

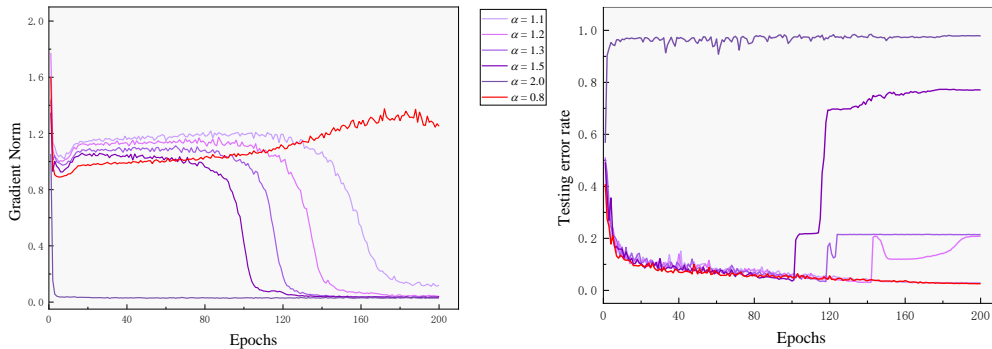


Figure 3. Evolutions of gradient norm of loss function (left) and testing error rate on Cifar10. Purple curves represent the results where  $\alpha \in \mathcal{B}_b$ , while the red curve represents the optimal results where  $\alpha = 0.8$ .

norm would drop to near zero immediately after the training start. As for other values in  $\mathcal{B}_b$ , although the gradient norm would keep stable for a while, it would suddenly drop rapidly within only several epochs. As soon as the gradient norm begins to drop rapidly, the testing error rate would increase immediately. When the gradient norms reach near zero, the testing error rates become stable. Interestingly, the final convergence error rates may be different even though the corresponding gradient norms are all near zero.

In summary, one should impose the penalty on the gradient norm with appropriate parameters in practice, where  $\alpha = 0.8$  and  $r \in \{0.05, 0.1\}$  are highly recommended for achieving the best performance. Based on our observation, this deployment could also give the best performance for most of our training, not just the WideResNet-28-10 architecture.

## 4.2. ImageNet

Next, we would check the effectiveness of our scheme on the large-scale dataset, namely ImageNet. For model architectures, we would adopt the VGG16-BN, ResNet50 and ResNet101 in our investigation. Likewise, we would still adopt the three training schemes for comparisons. However, unlike using the greedy strategy for hyperparameter searching in the previous section, we would directly set  $r = 0.05$  according to (Foret et al., 2021) and perform only a slight grid search on  $\alpha$  over  $\{0.7, 0.8\}$  based on our parameter study. For data augmentation, we just follow the prior works (He et al., 2016; Simonyan & Zisserman, 2015). Here, we would train each model with three different random seeds. Besides, all models are trained within 100 epochs with a cosine learning rate schedule.

Table 4 reports top-1 and top-5 testing error rates for different models. As we could see in the table, the model generalization could be improved when using our scheme compared to the other two schemes. Again, this confirms

Table 4. Testing error rate of models on ImageNet dataset when implementing the three training schemes.

	ImageNet	
	Top-1 Accuracy	Top-5 Accuracy
VGG16-BN		
Standard	26.89 $\pm$ 0.12	8.88 $\pm$ 0.06
SAM	26.41 $\pm$ 0.13	8.60 $\pm$ 0.05
Ours	<b>26.12<math>\pm</math>0.16</b>	<b>8.44<math>\pm</math>0.06</b>
ResNet50		
Standard	23.64 $\pm$ 0.17	7.01 $\pm$ 0.09
SAM	23.16 $\pm$ 0.11	6.72 $\pm$ 0.06
Ours	<b>22.87<math>\pm</math>0.15</b>	<b>6.59<math>\pm</math>0.11</b>
ResNet101		
Standard	21.97 $\pm$ 0.09	6.11 $\pm$ 0.07
SAM	21.02 $\pm$ 0.10	5.31 $\pm$ 0.09
Ours	<b>20.53<math>\pm</math>0.13</b>	<b>5.18<math>\pm</math>0.08</b>

the effectiveness of our scheme for practical training.

## 5. Conclusion

In this paper, we introduce an effective scheme for penalizing the gradient norm of loss function during training optimization. In our scheme, no Hessian computation would be involved, making it efficient to be implemented in practical optimization. We confirm the effectiveness of our training scheme via image classification experiments which involve extensive model architecture on commonly used datasets. By comparing with two baselines (the standard training scheme and SAM scheme) on Cifar and ImageNet dataset, we show the superiority of our scheme, where several new state-of-art performances are achieved. Remarkably, the improvement of using our scheme may be at most 70% greater



than that of using the SAM scheme. Also, we perform a parameter study to guide the setting of optimal hyperparameters in practice. It is shown that one should carefully set the parameters, in case of losing precision of approximation during penalty.

## Acknowledgements

We would like to thank all the reviewers and the meta-reviewer for their helpful comments and kindly advices. We would like to thank Yuhan Li and Chuncheng Zhao from Intelligence Sensing Lab at Tsinghua University for the discussions.

## References

- Ba, L. J., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXivPreprint*, abs/1607.06450, 2016.
- Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J. T., Sagun, L., and Zecchina, R. Entropy-sgd: Biasing gradient descent into wide valleys. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.
- Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. *arXivPreprint*, abs/1805.09501, 2018.
- Devries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXivPreprint*, abs/1708.04552, 2017.
- Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70, pp. 1019–1028, 2017.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houshy, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.
- Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-aware minimization for efficiently improving generalization. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.
- Gastaldi, X. Shake-shake regularization of 3-branch residual networks. In *5th International Conference on Learning Representations, ICLR 2017, Workshop Track Proceedings*, 2017.
- Goodfellow, I. J., Bengio, Y., and Courville, A. C. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3.
- Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: training imagenet in 1 hour. *arXivPreprint*, abs/1706.02677, 2017.
- Han, D., Kim, J., and Kim, J. Deep pyramidal residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pp. 6307–6315, 2017.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pp. 770–778, 2016.
- Hochreiter, S. and Schmidhuber, J. Flat minima. *Neural Comput.*, 9(1):1–42, 1997.
- Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, volume 37, pp. 448–456, 2015.
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and 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.
- Krogh, A. and Hertz, J. A. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, pp. 950–957, 1991.
- Kwon, J., Kim, J., Park, H., and 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*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5905–5914, 2021.
- Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.
- Neyshabur, B., Salakhutdinov, R., and Srebro, N. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing*, pp. 2422–2430, 2015.
- Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pp. 5947–5956, 2017.
- Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural Comput.*, 6(1):147–160, 1994.

- Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.
- Smith, A. E., Coit, D. W., Baeck, T., Fogel, D., and Michalewicz, Z. Penalty functions. *Handbook of evolutionary computation*, 97(1):C5, 1997.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.
- Virmaux, A. and Scaman, K. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pp. 3839–3848, 2018.
- Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, volume 28, pp. 1058–1066, 2013.
- Wu, L., Ma, C., and E, W. How SGD selects the global minima in over-parameterized learning: A dynamical stability perspective. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pp. 8289–8298, 2018.
- Wu, Y. and He, K. Group normalization. In *Computer Vision - ECCV 2018*, volume 11217, pp. 3–19, 2018.
- Xie, Z., Sato, I., and Sugiyama, M. A diffusion theory for deep learning dynamics: Stochastic gradient descent exponentially favors flat minima. In *9th International Conference on Learning Representations, ICLR 2021*, OpenReview.net, 2021.
- Yamada, Y., Iwamura, M., Akiba, T., and Kise, K. Shake-drop regularization for deep residual learning. *IEEE Access*, 7:186126–186136, 2019. ISSN 2169-3536.
- Yoshida, Y. and Miyato, T. Spectral norm regularization for improving the generalizability of deep learning. *arXivPreprint*, abs/1705.10941, 2017.
- Yun, S., Han, D., Chun, S., Oh, S. J., Yoo, Y., and Choe, J. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019*, pp. 6022–6031. IEEE, 2019.
- Zagoruyko, S. and Komodakis, N. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016*, 2016.
- Zheng, Y., Zhang, R., and 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. Simplification Process of Equation 6

The Equation 6 is,

$$\nabla_{\theta}L(\theta) = \nabla_{\theta}L_S(\theta) + \nabla_{\theta}(\lambda \cdot \|\nabla_{\theta}L_S(\theta)\|_p) \quad (12)$$

where would like to simply the second term  $\nabla_{\theta}(\lambda \cdot \|\nabla_{\theta}L_S(\theta)\|_2)$ .

For  $\theta = [\theta_1, \theta_2, \dots, \theta_n]^T$ , the 2-norm function is,

$$g(\theta) := \|\theta\|_2 = \sqrt{\theta_1^2 + \theta_2^2 + \dots + \theta_n^2} \quad (13)$$

The partial derivative of  $g(\theta)$  with respect to  $\theta_i$  denotes,

$$\frac{\partial g(\theta)}{\partial \theta_i} = \frac{\theta_i}{\sqrt{\theta_1^2 + \theta_2^2 + \dots + \theta_n^2}} = \frac{\theta_i}{\|\theta\|_2} = \frac{\theta_i}{g(\theta)} \quad (14)$$

Therefore,

$$\nabla_{\theta}g(\theta) = \left[ \frac{\theta_1}{g(\theta)}, \frac{\theta_2}{g(\theta)}, \dots, \frac{\theta_n}{g(\theta)} \right]^T \quad (15)$$

The gradient of  $\theta$  denotes  $h(\theta) := \nabla_{\theta}L(\theta)$ . And the term  $\nabla_{\theta}(\|\nabla_{\theta}L(\theta)\|_2)$  could be simplified as,

$$\begin{aligned} \nabla_{\theta}(\|\nabla_{\theta}L(\theta)\|_2) &= \nabla_{\theta}(g \circ h)(\theta) \\ &= (\nabla_{\theta}g(\zeta)|_{\zeta=h(\theta)}) \cdot (\nabla_{\theta}h(\theta)) \\ &= \left( \frac{\zeta}{g(\zeta)} \Big|_{\zeta=h(\theta)} \right) \cdot (\nabla_{\theta}h(\theta)) \\ &= \left( \frac{\nabla_{\theta}L(\theta)}{\|\nabla_{\theta}L(\theta)\|_2} \right) \cdot (\nabla^2 L(\theta)) \\ &= \frac{1}{\|\nabla_{\theta}L(\theta)\|_2} \cdot \nabla^2 L(\theta) \cdot \nabla_{\theta}L(\theta) \end{aligned} \quad (16)$$

Back to Equation 6, the equation could be simplified based on Equation 16,

$$\begin{aligned} \nabla_{\theta}L(\theta) &= \nabla_{\theta}L_S(\theta) + \nabla_{\theta}(\lambda \cdot \|\nabla_{\theta}L_S(\theta)\|_p) \\ &= \nabla_{\theta}L_S(\theta) + \lambda \cdot \nabla^2 L_S(\theta) \cdot \frac{\nabla_{\theta}L_S(\theta)}{\|\nabla_{\theta}L_S(\theta)\|_2} \end{aligned} \quad (17)$$