SOFTMAX GRADIENT TAMPERING: DECOUPLING THE BACKWARD PASS FOR IMPROVED FITTING

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Softmax Gradient Tampering, a technique for modifying the gradients in the backward pass of neural networks in order to enhance their accuracy. Our approach transforms the predicted probability values using a power-based probability transformation and then recomputes the gradients in the backward pass. This modification results in a smoother gradient profile, which we demonstrate empirically and theoretically. We do a grid search for the transform parameters on residual networks. We demonstrate that modifying the softmax gradients in ConvNets may result in increased training accuracy, thus increasing the fit across the training data and maximally utilizing the learning capacity of neural networks. We get better test metrics and lower generalization gaps when combined with regularization techniques such as label smoothing. Softmax gradient tampering improves ResNet-50's test accuracy by 0.52% over the baseline on the ImageNet dataset. Our approach is very generic and may be used across a wide range of different network architectures and datasets.

1 INTRODUCTION

Smooth gradient flow is the key to successful convergence of deep neural networks. Batch Normalizing Ioffe & Szegedy (2015), Weight Standardization Qiao et al. (2019), and Group Normalization Wu & He (2018) are all types of normalization techniques that smooth the gradient landscape in the backward pass. Batch normalization smoothes the loss and gradient surfaces by limiting their lipschitzness Santurkar et al. (2018). Methods of smoothing gradients by manipulating the labels distribution include label smoothing Szegedy et al. (2016), mixup Zhang et al. (2017).

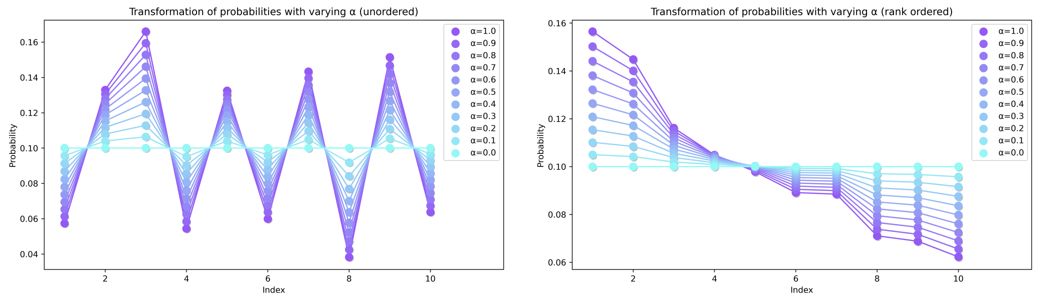
In this paper, we manipulate the gradients in an unconventional way to impose smooth gradients in the backward pass. This technique is one of a family of Gradient Tampering methods, called Softmax Gradient Tampering since it is conducted at the softmax stage. We demonstrate that Softmax Gradient Tampering improves convergence fit and therefore improves training and testing accuracy.

Our main contributions in this paper are the following:

1. We propose Softmax Gradient Tampering that improves the learning capacity of neural networks and allows for a much better fit on the training dataset. This reflects both in training and validation accuracy metrics.
2. We theoretically analyze how softmax gradient tampering works from the perspective of gradient smoothness.
3. We provide comprehensive findings from different models, and datasets.

2 METHOD: SOFTMAX GRADIENT TAMPERING

The softmax gradient tampering technique is described in this section. Softmax gradient tampering modifies the gradient in the backward pass. It does not alter the behaviour in the forward pass. Before we describe our method, we briefly explain the forward pass itself. A neural network with parameters W generates C logits denoted by z for every input vector x. z is given as $z = Wx$. Then a set of probability values p_i are generated from the logits using a softmax function which



(a) Unordered distribution

(b) Rank ordered distribution

Figure 1: Illustration of the impact of softmax gradient tampering on probability values of an arbitrary distribution (number of classes, $C = 10$). As α reduces from 1 to 0, the probability distributions (left: unordered or right: ordered) becomes flatter and flatter, approaching the uniform distribution of $1/C$ for every index.

is defined as $p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} z_j}$. p_i and z_i represent the predicted probability values and the logits for the i^{th} class respectively. Following this step, the loss function is invoked and the loss between the predicted probability values the ground truth labels (which is also a probability distribution) is calculated. If the loss function is cross-entropy loss, then the value of the loss is given as $\mathcal{L} = -\sum_{i=1}^{C} q_i \log{(p_i)}$ where q_i is the ground truth label of the i^{th} class for a particular training example. By backpropagation rule, we can calculate the gradient of the loss with respect to the logits which takes the form $\frac{\partial \mathcal{L}}{\partial z_i} = p_i - q_i$.

The softmax gradient tampering technique is now described. We introduce a hyperparameter α, which takes a value between $[0, 1]$ and regulates the degree of tampering. The softmax gradient tampering method modifies the predicted probability values in the backward pass as follows:

$$p_i' = \frac{p_i^{\alpha}}{\sum_{j=1}^{C} p_j^{\alpha}} \qquad i = 1, \ldots, C \quad 0 \leq \alpha \leq 1 \tag{1}$$

The above transformation changes the gradient of the loss with respect to the logits as follows:

$$\frac{\partial \mathcal{L}}{\partial z_i} = p_i' - q_i \tag{2}$$

The rest of the backward pass proceeds as usual. We denote the original probability distribution as P (with values p_i at the i^{th} index) and the transformed distribution as P' (with values p_i' at the i^{th} index).

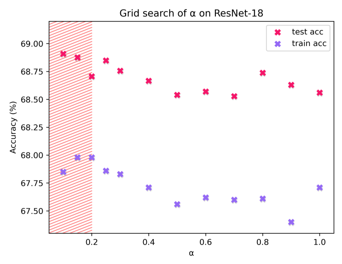
A code snippet of the algorithm implemented using PyTorch Paszke et al. (2019) is given in the appendix. In PyTorch, the available method to modify gradients in the backward pass is using the `register_hook` function which is called in the `forward` function.

The backward pass gradient profile is smoothed by the transformation of the probability distribution using equation 1. Because the gradient at the softmax layer's input ($\partial \mathcal{L}/\partial z_i$) is directly proportional to the predicted probabilities p_i, smoothing out the probability distribution also smooths out the gradient profile. We explore the effect of this change from a theoretical standpoint in section 5. In section 3, we offer empirical data and experiments to support the proposed method.

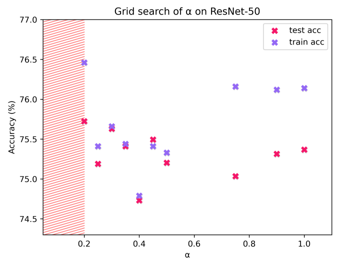
3 EXPERIMENTS

3.1 GRID SEARCH EXPERIMENTS

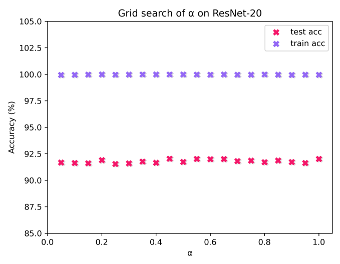
We do a thorough hyperparameter search on the ResNet-18 He et al. (2016) architecture and the ImageNet-1K dataset, changing the value of α from 0.1 to 0.9. We begin with a coarse grid search



(a)
ResNet-18
ImageNet-1K



(b)
ResNet-50
ImageNet-1K



(c)
ResNet-20
CIFAR-10

Figure 2: Results of grid search on different architectures and datasets. Areas with red shading are more likely to have erratic performance.

in 0.1 increments. The optimum values of α are found to be between 0.2 and 0.4. Then we do a fine grid search in 0.05 increments between 0.2 and 0.4. We find that the optimum performance for ResNet-18 is achieved at a $\alpha = 0.25$. We discover that the optimum value of α is 0.3 by repeating the grid search for ResNet-50 He et al. (2016) in the range of 0.2 to 0.4 with increments of 0.05. We utilise a batch size of 1024 over four V100 GPUs for each grid search experiment. Each experiment is carried out with a 50 epoch budget. In the first 2 epochs, we raise the learning rate linearly from 4×10^{-4} to 0.4. The learning rate is then decayed using a cosine scheduler (single schedule with no restarts) Loshchilov & Hutter (2016) from 0.4 at the start of the $2nd$ epoch to 4×10^{-4} at the end of the $46th$ epoch. We continue training for the next 4 epochs at a constant learning rate of 4×10^{-4} (cooldown phase). The plots are shown in Figure (2). We do a grid search on the ResNet-20 He et al. (2016) architecture and the CIFAR-10 dataset, with a epoch budget of 360. However, we get no advantage for any value of α. This is covered in more detail in section 3.3.

Figure (2a,b) shows a pattern in which training accuracy rises when α is lowered from 1 to 0, indicating the method's potential as an algorithm to drive the network towards better learnt feature representations and therefore a better fit on the training dataset. This is due to the transform's nature, which smoothes the gradients. This line of reasoning is expanded upon in section 5.

The generalization gap is lowest when $\alpha = 0.2$ but it comes at the expense of test accuracy, so we disregard it. Values of $\alpha < 0.2$ cause issues with half precision training, therefore we choose $\alpha = 0.25$. In our trails with ResNet-18, we find optimal values of $\alpha = 0.25$, while in our experiments with ResNet-50, we get optimal values of $\alpha = 0.3$. We port these values for all 100 epoch schedules, as well as the Squeeze-and-Excitation versions of the ResNets. We indeed find a peak at $\alpha = 0.1$, but we found that such low α values are typically unstable in most subsequent tests, especially with mixed precision or half precision arithmetic. According to our observations in section 3.5, low values of α frequently result in high logit norms, which directly cause floating point problems Micikevicius et al. (2017).

3.2 Experiment on the ImageNet dataset

We perform experiments on the ImageNet-1K dataset Deng et al. (2009), which has 1.28 million images divided into 1000 classes. ImageNet-1K's test set consists of 5000 images per class. We train models based on several ResNets, including the Squeeze-and-Excitation versions. We observe a constant 0.1% to 0.15% rise in ResNet-18's test accuracy. What's more interesting is that we're seeing comparable improvements in training accuracy. Softmax gradient tampering is not a regularization technique, as shown by this. It just allows the network to better fit the dataset. In section 4, we show how combining softmax gradient tampering with regularization techniques such as label smoothing may improve network performance and reduce the generalization gap.

We find significantly larger improvements in experiments using ResNet-50, a 0.5% increase over the step scheduling baseline and a 0.3% increase over the cosine scheduling baseline. In studies with Squeeze-and-Excitation networks, we use the cosine scheduler. In these experiments, we find consistent improvements across the board, with SEResNet-18 Hu et al. (2018) gaining 0.4% and

Table 1: Results and comparisons for networks trained on ImageNet-1K. α denotes the hyperparameter that controls the amount of gradient tampering at softmax, eq. (1). Rows with $\alpha \neq 1$ are experiments conducted using the proposed method. $\alpha = 1.0$ denotes no tampering.

Model	**Scheduler**	**Hyperparameters**	**Train Acc**	**Test Acc**	**Gap**
ResNet-18	Step	α=1.0	69.95	69.704	0.246
	Step	α=0.25	70.3	69.844	0.456
	Cosine	α=1.0	70.38	70.208	0.172
	Cosine	α=0.25	70.53	**70.298**	0.232
ResNet-50	Step	α=1.0	78.99	75.97	3.02
	Step	α=0.3	79.56	76.494	3.066
	Cosine	α=1.0	79.18	76.56	2.62
	Cosine	α=0.3	79.43	**76.886**	2.544
ResNet-101	Cosine	α=1.0	82.29	77.896	4.394
	Cosine	α=0.3	83.10	**78.258**	4.842
SEResNet-18	Cosine	α=1.0	71.42	71.09	0.33
	Cosine	α=0.25	71.6	**71.436**	0.164
SEResNet-50	Cosine	α=1.0	81.5	77.218	4.282
	Cosine	α=0.3	82.47	**77.952**	4.518

SEResNet-50 Hu et al. (2018) gaining 0.7%. As we saw with ResNet-18, we find continuous improvements in training performance because the networks are just training better and arriving to better optimas at convergence. ResNet-101 training provides a modest advantage of 0.35%.

All models are trained on four V100 GPUs with a batch size of 1024. We utilise the same set of hyperparameters in each experiment, which are as follows: 100 epoch budget, 5 epochs linear warmup phase beginning with a learning rate of 4×10^{-4} and ending with a peak learning rate of 0.4. In our studies, we employ either a step scheduler (dividing the learning rate by 10 at the 30^{th}, 60^{th}, and 90^{th} epochs) or a cosine decay scheduler. We observe a consistent improvement in test metrics when utilising cosine scheduling across all of our experiments. Finally, towards the conclusion of training, we impose a 10 epoch cooling phase in which the network is trained with a constant learning rate schedule set at 4×10^{-4}. Other hyperparameters include a weight decay value of 5×10^{-4} and a momentum of 0.9 for the SGD Nesterov optimizer. All of our models are trained using mixed floating point precision. The findings are shown in Table 1.

3.3 Experiment on CIFAR datasets

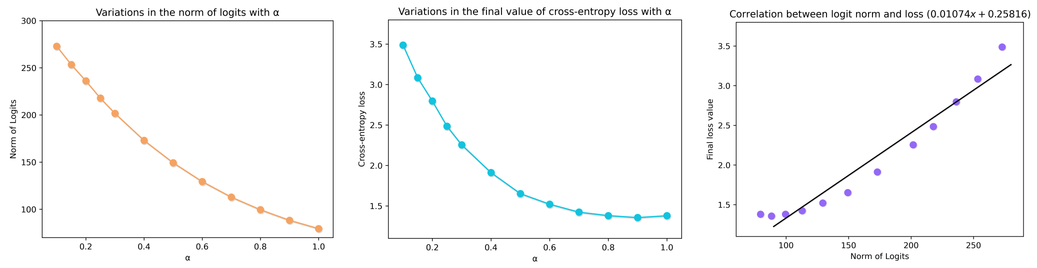
The CIFAR-10 dataset Krizhevsky et al. (2014) is divided into 10 classes, each with 5000 datapoints for training and 1000 datapoints for testing. In comparison to ImageNet-1K, it is a much smaller dataset. The most frequent situation seen when training models on CIFAR-10 is that the models fully overfit the training dataset, achieving almost 100% accuracy. Given that the proposed method (section 2) is not intended as a regularization technique, no improvements in the training of such datasets are anticipated. We see the same thing in Figure 2c, where we show the results of a hyperparameter grid search. Different values of α do not impair ResNet-20's performance on CIFAR-10, but neither do they improve on the baseline.

3.4 Towards removing Normalization

Experiments are carried out on non-Batch Normalized networks. We specifically utilize the ResNet-18 model and train it on ImageNet-1K without any of the BatchNorm layers. We see a substantial increase in test accuracy of about 1%. Similarly to prior findings, we get a 1.1% increase in training accuracy, leading us to infer that when softmax gradient tampering is used, the network's convergence optima is significantly superior. Table 2 shows the findings.

3.5 Effect on Loss, Logits and other Metrics

As seen in Figure (3a), the logit norm increases as α falls from 1 to 0. In the initial training phase, softmax gradient tampering causes the gradient in both the valid and wrong classes to rise as the



(a) Logit vs. α (b) Cross-entropy loss vs. α (c) Logit-Loss correlation

Figure 3: Plots of various statistical measures: **(a)** Variations in the logit norm vs. α. The logit norm is calculated per image over the test set of ImageNet-1K (at the linear layer of the ResNet-18 architecture) and then averaged. **(b)** Variations in the final loss values obtained for different α settings. **(c)** Correlation between loss and logits. Regression line equation :$(0.01074x + 0.25816)$.

Table 2: Results for non-Batch Normalized ResNets on ImageNet-1K. α denotes the hyperparameter that controls the amount of gradient tampering at softmax, eq. (1). Rows with $\alpha \neq 1$ are experiments conducted using the proposed method. $\alpha = 1.0$ denotes no tampering.

Model	**Scheduler**	**Hyperparameters**	**Train Acc**	**Test Acc**	**Gap**
ResNet-18 (W/O BN)	Cosine	α=1.0	68.86	66.796	2.064
	Cosine	α=0.25	69.97	**67.796**	2.174

network misclassifies majority of the examples. Due to the non-linear nature of the softmax function, the predicted probability distribution may frequently be close to a delta distribution, even in instances of misclassifications. But softmax gradient tampering reshapes the predicted probability distribution. It redistributes probability from the confident classes to the less confident ones. Gradient rises in all classes. We also see that the final value of the loss rises as α drops, and that the logit norm and the final value of the loss are linearly correlated Figure (3c).

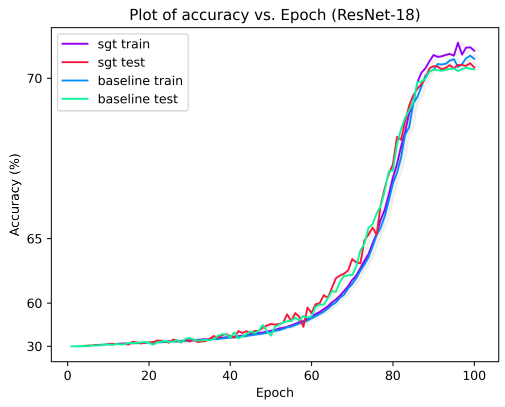
ResNet-18 and ResNet-50 per epoch training and test accuracies with and without softmax gradient manipulation are shown in Figure (4a,b). In all instances, we observe a gain. This means that softmax gradient tampering forces the network to learn better representations.

4 Ablation Study

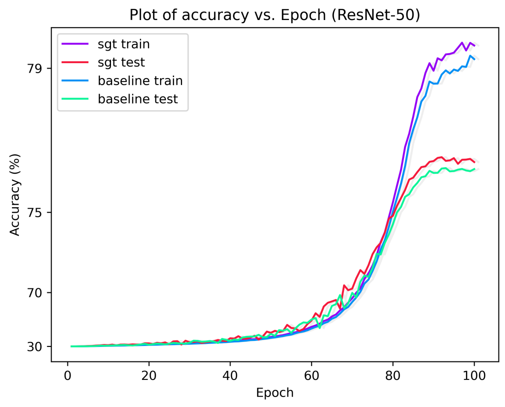
We conduct an ablation study to investigate the consequences of softmax gradient tampering. We have previously demonstrated from the experiments depicted in section 3 that softmax gradient tampering causes the network to arrive at a better optima and fit a particular training dataset better than the baseline. Furthermore, based on our CIFAR-10 tests, we have experimentally shown that the technique performs best on varied datasets. Because most networks on the CIFAR-10 dataset begin overfitting extremely early, and therefore a non-regularizing technique like softmax gradient tampering has little effect on the network's performance. As a result, we combine our approach with other regularization techniques to see whether this is really the case. If softmax gradient tampering is not a regularizer, the generalization gap will widen. The generalization gap would be reduced if we added a regularizer on top of it. The findings are shown in the Table 3. We see that adding softmax gradient tampering increases the gap between the training and test accuracies, while introducing a label smoothing regularizer decreases the gap. When both softmax gradient tampering and label smoothing are applied concurrently, the best accuracies are achieved.

5 Why does Softmax Gradient Tampering works?

In this section, we describe why softmax gradient tampering work. We use the same setup as described in section 2. We start by investigating the effect of the transform on the softmax probabilities, and then we prove that the transform smoothens the gradient profile.



(a) Plot of training and test accuriacies (ResNet-18)



(b) Plot of training and test accuriacies (ResNet-50)

Figure 4: Log-linear plots of training and test accuracies and comparison with baseline: **(a)** ResNet-18 ($\alpha = 0.25$), **(b)** ResNet-50 ($\alpha = 0.3$). SGT (Softmax Gradient Tampering) improves upon baseline in all cases.

Lemma 1. For any arbitrary probability distribution P with probability values given by p_i for $i = 1, \ldots, C$, the corresponding transformed probability values p'_i given by [1] has a threshold $\left(\sum_{j=1}^{C} p_j^\alpha\right)^{\frac{1}{\alpha-1}}$ and

$$\begin{aligned} p'_i \geq p_i, \text{ if } p_i \leq \left(\sum_{j=1}^{C} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \\ p'_i < p_i, \text{ if } p_i > \left(\sum_{j=1}^{C} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \end{aligned} \tag{3}$$

Proof. The proof follows directly from the definition of p'_i. We notice that if $p'_i < p_i$, we get:

$$\frac{p_i^\alpha}{\sum_{j=1}^{C} p_j^\alpha} < p_i \tag{4}$$

$$\Rightarrow p_i^{\alpha-1} < \sum_{j=1}^{C} p_j^\alpha \tag{5}$$

$$\Rightarrow p_i > \left(\sum_{j=1}^{C} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \tag{6}$$

□

We call this threshold, the *stationary threshold*. The stationary threshold is that value of p_i that does not change after the transformation.

Proposition 1. At $\alpha = 0$, the stationary threshold equals $1/C$ and all values of the transformed distribution p'_i reduces reduces to the uniform distribution for $i = 1, \ldots, C$,.

Proof. From equation (3), we see that the stationary threshold at $\alpha = 0$ is $1/C$. Also, following from the definition of the transformed probabilities (1) we conclude that if $\alpha = 0$, then all values of p'_i are $1/C$. Therefore the transformed distribution at $\alpha = 0$ is a uniform distribution.

Since we have established that values of p_i which are greater than the stationary threshold decreases and move down towards the stationary threshold, and values in p_i lower than the stationary threshold moves up towards the stationary threshold, this transformation makes the distribution more uniform (i.e. it smooths out the actual distribution) as α is decreased from 1 and down towards 0. This final observation we prove in the following theorem.

Theorem 1. For any arbitrary probability distribution P with probability values p_i for $i = 1, \ldots, C$, the stationary threshold of the transformed distribution P' with probability values $p'_i = \frac{p_i^\alpha}{\sum_{j=1}^{C} p_j^\alpha}, 0 \leq \alpha \leq 1$ is a monotonically non-decreasing function with respect to α.

Proof. To prove monotonicity, we first compute the gradient of the stationary threshold with respect to the variable in concern, α.

$$\frac{\partial}{\partial\alpha}\left(\sum_{j=1}^{c} p_j^\alpha\right)^{\frac{1}{\alpha-1}} = \left(\sum_{j=1}^{c} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \left(\frac{\sum_{j=1}^{c} p_j^\alpha \log(p_j)}{(\alpha-1)\sum_{j=1}^{c} p_j^\alpha} - \frac{\log\left(\sum_{j=1}^{c} p_j^\alpha\right)}{(\alpha-1)^2}\right) \tag{7}$$

$$= \frac{1}{\alpha(\alpha-1)^2}\left(\sum_{j=1}^{c} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \left(\frac{(\alpha-1)\sum_{j=1}^{C} p_j^\alpha \log\left(p_j^\alpha\right)}{\sum_{j=1}^{c} p_j^\alpha} - \alpha \log\left(\sum_{j=1}^{c} p_j^\alpha\right)\right) \tag{8}$$

If $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ are non-negative numbers, then using the log sum inequality, we get $\sum_{j=1}^{n} a_j \log\left(\frac{a_j}{b_j}\right) \geq \left(\sum_{j=1}^{n} a_j\right) \log\left(\frac{\sum_{j=1}^{n} a_j}{\sum_{j=1}^{n} b_j}\right)$. Setting $a_j = p_j^\alpha$ and $b_j = 1$, we get the following lower bound

$$\sum_{j=1}^{C} p_j^\alpha \log\left(p_j^\alpha\right) \geq \left(\sum_{j=1}^{C} p_j^\alpha\right) \log\left(\frac{1}{C}\sum_{j=1}^{C} p_j^\alpha\right) \tag{9}$$

Substituting (9) in (8), we get:

$$\frac{\partial}{\partial\alpha}\left(\sum_{j=1}^{c} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \geq \frac{1}{\alpha(\alpha-1)^2}\left(\sum_{j=1}^{c} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \left((1-\alpha)\log(C) - \log\left(\sum_{j=1}^{C} p_j^\alpha\right)\right) \tag{10}$$

p^α is concave, and so by Jensen's inequality we get the following upper bound for the second term:

$$\left(\frac{1}{C}\sum_{j=1}^{C} p_j\right)^\alpha \geq \frac{1}{C}\sum_{j=1}^{C} p_j^\alpha \tag{11}$$

$$\Rightarrow \log\left(\sum_{j=1}^{C} p_j^\alpha\right) \leq (1-\alpha)\log(C) \tag{12}$$

Substituting (12) in (10),

$$\frac{\partial}{\partial\alpha}\left(\sum_{j=1}^{c} p_j^\alpha\right)^{\frac{1}{\alpha-1}} \geq 0 \tag{13}$$

□

We conclude from the analysis that the stationary threshold is a monotonic non-decreasing function with respect to α, which in turn means that the transformation is an order-preserving map. All values greater than the threshold move towards the threshold after transformation and all values below the

Table 3: Ablation study for ResNet-50 on ImageNet-1K. LS column denotes whether label smoothing has been applied. A smoothing hyperparameter value of 0.1 has been used. α denotes the hyperparameter that controls the amount of gradient tampering at softmax, eq. (1). Rows with $\alpha \neq 1$ are experiments conducted using the proposed method. $\alpha = 1.0$ denotes no tampering.

Model	**Scheduler**	**LS**	**Hyperparameters**	**Train Acc**	**Test Acc**	**Gap**
ResNet-50	Cosine		α=1.0	79.18	76.56	2.62
	Cosine	✓	α=1.0	78.81	76.698	2.112
	Cosine		α=0.3	79.43	76.886	2.544
	Cosine	✓	α=0.3	**78.47**	**76.968**	**1.502**

threshold also move towards the threshold, and the threshold itself moves monotonically towards $1/C$ as α is decreased from 1 to 0. This concludes that the transformation smooths the original distribution. Since the gradient with respect to logits ($\partial\mathcal{L}/\partial z_i$) is a direct function of the smoothed out distribution, this smoothes out the gradients as well.

6 Related Work

In this section, we examine techniques that are related to gradient tampering and emphasize the distinctions between the methods.

Relationship to Label Smoothing: Label smoothing, introduced by Szegedy et al. Szegedy et al. (2016), utilizes smoothing of the ground truth labels as a method to impose regularization on the logits and the weights of the neural network.

The formulation is as follows. If q_i is the value at the i^{th} index of the one hot encoded vector of the ground truth label, then the transformed distribution of the labels q_i' is:

$$q_i' = \begin{cases} 1-\epsilon & \text{if } i = y, \\ \frac{\epsilon}{K-1} & otherwise \end{cases} \tag{14}$$

where ϵ is a hyperparameter with a value in $[0, 1]$.

Müller et al. investigates how label smoothing work in Müller et al. (2019). Using visualizations of penultimate layer activations, they demonstrate that label smoothing calibrates the networks predictions and aligns them with the accuracies of the predictions. Parallel works in label smoothing include Reed et al. (2015); Zhang et al. (2021); Reed et al. (2014); Xie et al. (2016); Dubey et al. (2018); Li et al. (2019).

Softmax gradient tampering works in a much different way. As we proved in section 5 that our proposed transformation of probabilities smooths the prediction vector which in turn has the effect of smoothing the gradient landscape, since the gradient vector arising from softmax is $p_i' - q_i$. This has the opposite effect of label smoothing, because the gradient vector arising from softmax in a scenario with label smoothing is $p_i - q_i'$. Hence, softmax gradient tampering is *not* a regularization method. From our studies, we show that it in fact leads to larger generalization gap (Table 1, 3). Coupled with softmax gradient tampering, label smoothing closes the generalization gap.

Relationship to Knowledge Distillation: The formulation of softmax gradient tampering is similar to the temperature based logit suppression formulation of Hinton et al. Hinton et al. (2015). Knowledge distillation Ba & Caruana (2014) is a process in which two networks are trained. First a teacher network is trained on a given dataset and then the soft-labels (the predicted probabilities) of the teacher network are used to train another network, called the student network. In similar style of label smoothing, knowledge distillation aims to generate smooth gradient behaviour by forcing the network not to overpredict on any single image. As in case of label smoothing, the student network's weights are automatically penalized if the network assigns a probability value higher than the soft labels generated by the teacher network.

The formulation of temperature based logit suppression introduced as a special case of logit matching, so far has only been used in the context of distillation. Even though the formulation of temperature based logit formulation leads to the same formulation of softmax gradient tampering, there is one key difference, and that is the aforementioned logit suppression is applied in the forward pass. This means that the temperature variable becomes a part of the computation graph. In the case of softmax gradient tampering, where we directly tamper the gradients without introducing any change in the forward pass is a more generalized way of arriving at the smoothness effects of the gradients in the backward pass. The introduction of temperature based logit suppression in the forward pass also leads to making inverse changes in the learning rate or in the formulation of the loss function. However, in case of softmax gradient tampering, no such changes are required. In section 3, we demonstrate that networks using softmax gradient tampering method can be trained using the same hyperparameters as the ones used in the baseline models.

Variants of knowledge distillation include Zhang et al. (2019); Furlanello et al. (2018); Passalis & Tefas (2018); Ge et al. (2019); Yun et al. (2020); Wang et al. (2021); Aguilar et al. (2020); Yao et al. (2018); Wang et al. (2020); Peng et al. (2019); Ge et al. (2018); Zhang et al. (2018b); Liu et al. (2018); Cho & Hariharan (2019); Yuan et al. (2020). Aside from that, the smoothing effects on gradients have never been studied before, and this is the first time, to the best of our knowledge, that this has been done in this study.

Relationship to Gradient Clipping (and variants thereof): Gradient clipping is often used as means of limiting exploding gradients in the backward pass. It normalizes and scales the gradient, if the norm of the gradient exceeds a predetermined threshold.

The formulation is as follows: If G is a gradient vector which is $\partial\mathcal{L}/\partial\theta$ (the gradient of the loss $\mathcal{L}$ with respect to the parameters θ)

$$G \rightarrow \begin{cases} \lambda \frac{G}{\|G\|} & \text{if } \|G\| > \lambda, \\ G & \text{otherwise} \end{cases} \tag{15}$$

It works in practice as it helps the network to avoid gradient explosion during training, but it doesn't really make the loss landscape smoother than it already is. It simply alters the scale of the gradients with respect to the norm. Brock et al. Brock et al. (2021) finds that the clipping threshold λ is an extremely sensitive hyperparameter.

Adaptive gradient clipping is a variant of gradient clipping. The adaptiveness of adaptive gradient clipping comes from the use of the Frobenius norm of the weight matrix. NF-Nets Brock et al. (2021) uses Adaptive Gradient Clipping (AGC) and achieves close to state-of-the-art accuracies in ImageNet benchmark without the use of traditional normalization methods.

Gradient clipping methods and the variants thereof are also gradient tampering methods, i.e. they alter the behaviour of the backward pass independently from the forward pass. Our proposed method softmax gradient tampering works directly at the softmax level, whereas gradient clipping methods work at the parameter level, requiring much more compute. Softmax gradient tampering on the other hand requires only a small modification and significantly lower computation overhead.

Similar lines of theoretical analysis as in section 5 on logits and temperatures include Zhang et al. (2018a); Pereyra et al. (2017).

7 Discussion

We showed the application of softmax gradient tampering, a technique for providing a smooth gradient during neural network training. With the use of different network topologies and datasets, we were able to show its potential and explore its impacts from a theoretical standpoint.

8 Reproducibility

Reproducible code, training recipes for experiments in section 3, pretrained checkpoints and training logs are provided at: `https://github.com/bishshoy/softmax-gradient-tampering`. Code adapted from Wightman (2019).

Instability caused by half precision floating point operations is frequent in neural network training procedures, and it is significant when the logits reach high values Micikevicius et al. (2017). When training ResNet-50 with high batch sizes (1024), we experienced sporadic problems. We were able to easily stabilize the training by training the first epoch without softmax gradient tampering and then training the other epochs with softmax gradient tampering enabled. However, when we trained ResNet-50 with smaller batch sizes (512), we had no problems with reproducibility.

REFERENCES

Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge distillation from internal representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7350–7357, 2020.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2654–2662, 2014.

Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4794–4802, 2019.

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

Abhimanyu Dubey, Otkrist Gupta, Pei Guo, Ramesh Raskar, Ryan Farrell, and Nikhil Naik. Pairwise confusion for fine-grained visual classification. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 70–86, 2018.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International Conference on Machine Learning*, pp. 1607–1616. PMLR, 2018.

Shiming Ge, Shengwei Zhao, Chenyu Li, and Jia Li. Low-resolution face recognition in the wild via selective knowledge distillation. *IEEE Transactions on Image Processing*, 28(4):2051–2062, 2018.

Shiming Ge, Zhao Luo, Chunhui Zhang, Yingying Hua, and Dacheng Tao. Distilling channels for efficient deep tracking. *IEEE Transactions on Image Processing*, 29:2610–2621, 2019.

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*, pp. 770–778, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55(5), 2014.

Changsheng Li, Chong Liu, Lixin Duan, Peng Gao, and Kai Zheng. Reconstruction regularized deep metric learning for multi-label image classification. *IEEE transactions on neural networks and learning systems*, 31(7):2294–2303, 2019.

Xuan Liu, Xiaoguang Wang, and Stan Matwin. Improving the interpretability of deep neural networks with knowledge distillation. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 905–912. IEEE, 2018.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 4696–4705, 2019.

Nikolaos Passalis and Anastasios Tefas. Unsupervised knowledge transfer using similarity embeddings. *IEEE transactions on neural networks and learning systems*, 30(3):946–950, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-...-deep-learning-library.pdf`.

Zhimao Peng, Zechao Li, Junge Zhang, Yan Li, Guo-Jun Qi, and Jinhui Tang. Few-shot image recognition with knowledge transfer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 441–449, 2019.

Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. Regularizing neural networks by penalizing confident output distributions. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. 2019.

Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.

Scott E. Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems*, pp. 2488–2498, 2018.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Jiyue Wang, Pei Zhang, and Yanxiong Li. Memory-replay knowledge distillation. *Sensors*, 21(8): 2792, 2021.

Ning Wang, Wengang Zhou, Yibing Song, Chao Ma, and Houqiang Li. Real-time correlation tracking via joint model compression and transfer. *IEEE Transactions on Image Processing*, 29:6123–6135, 2020.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing cnn on the loss layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4753–4762, 2016.

Jiangchao Yao, Jiajie Wang, Ivor W Tsang, Ya Zhang, Jun Sun, Chengqi Zhang, and Rui Zhang. Deep learning from noisy image labels with quality embedding. *IEEE Transactions on Image Processing*, 28(4):1909–1922, 2018.

Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3903–3911, 2020.

Sukmin Yun, Jongjin Park, Kimin Lee, and Jinwoo Shin. Regularizing class-wise predictions via self-knowledge distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13876–13885, 2020.

Chang-Bin Zhang, Peng-Tao Jiang, Qibin Hou, Yunchao Wei, Qi Han, Zhen Li, and Ming-Ming Cheng. Delving deep into label smoothing. *IEEE Transactions on Image Processing*, 30:5984–5996, 2021.

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

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3713–3722, 2019.

Xu Zhang, Felix Xinnan Yu, Svebor Karaman, Wei Zhang, and Shih-Fu Chang. Heated-up softmax embedding. *arXiv preprint arXiv:1809.04157*, 2018a.

Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4320–4328, 2018b.

A Appendix

In PyTorch, the available method to modify gradients in the backward pass is using the `register_hook` function which is called in the `forward` function.

```
# Modification of gradients take place
# inside the forward function
def forward(self, x, labels):
    # x: Input tensor with shape [N, C, H, W]
    # labels: One hot ground truth with shape [N, C]
    logits = self.layers(x)

    def softmaxGradientTampering(grad):
        alpha = 0.3
        # Recover the predicted probabilites
        # generated in the forward pass
        grad_temp = grad / grad.shape[0]
        pred = grad_temp + labels
        # Transform the predicted probabilities
        pred = pred ** alpha
        pred /= pred.sum(-1, keepdim=True)
        # Modify the gradient
        modified_grad = pred - labels
        modified_grad = modified_grad * grad.shape[0]
        return modified_grad
    if self.training:
        logits.register_hook(softmaxGradientTampering)
    return logits
```

Figure 5: Python code of Softmax Gradient Tampering based on PyTorch.