Altering Backward Pass Gradients to Improve Convergence

Bishshoy Das*, Milton Mondal*, Brejesh Lall, Shiv Dutt Joshi, and Sumantra Dutta Roy

Indian Institute of Technology Delhi, Hauz Khas, New Delhi - 110016, India
{bishshoy.das, milton.mondal, brejesh, sdjoshi, sumantra}@ee.iitd.ac.in

Abstract

In standard neural network training, the gradients in the backward pass are determined by the forward pass. As a result, the two stages are coupled. This is how most neural networks are trained hitherto. Gradient modification in the backward pass has seldom been studied in the literature. In this paper we explore decoupled training, where we alter the gradients in the backward pass. We propose a simple yet powerful method called PowerGrad Transform (PGT), that alters the gradients before the weight update in the backward pass and significantly enhances the predictive performance of a convolutional neural network. PGT trains networks to arrive at a better optima at convergence. It is computationally efficient, and adds no additional cost to either memory or compute, but results in improved final accuracies on both the training and test datasets. PowerGrad Transform is easy to integrate into existing training routines, requiring just a few lines of code. PGT accelerates training and makes it possible for the network to better fit the training data. With decoupled training, our method improves baseline accuracies for ResNet-50 by 0.73%, for SE-ResNet-50 by 0.66% and by more than 1.0% for the non-normalized ResNet-18 network on the ImageNet classification task.

1. Introduction

Backpropagation is traditionally used for training deep neural networks [9]. Gradients are computed using basic calculus principles to adjust the weights during backpropagation [8]. Alternatives to traditional gradients has rarely been studied in the literature hitherto. In normal training procedures, gradients are computed immediately in the backward pass utilizing the values obtained in the forward pass. This makes the backward pass coupled with the forward pass. However, decoupling the backward pass from the forward pass by modifying the gradients to improve training efficiency and final convergent accuracy has hardly been explored. In this paper we explore the landscape of decoupling the forward pass from the backward pass by altering the gradients and subsequently updating the network's parameters with the modified gradients. There are several ways to achieve gradient modification in the backward pass. We discuss a few techniques in Fig. 1.

Type 0: No modification: In this method, we calculate the gradients using the standard calculus rules and use the chain rule to calculate the gradients of the rest of the network's parameters, also known as backpropagation as portrayed in Fig. 1(a). The network is then updated with the gradient descent equation:

$$W_i^{t+1} = W_i^t - \lambda \, \nabla_{W_i}(L) \qquad i = D, D-1, \ldots, 1 \quad (1)$$

Type I: Independent gradient modification at multiple points: Here the gradients are first computed using standard procedure and then individually altered as shown in Fig. 1(b). Gradient clipping [12] and Adaptive gradient clipping [1] are examples of such modifications. In both of these methods, the gradients are first computed using standard rules and then they are modified using some function. It can be described as:

$$W_i^{t+1} = W_i^t - \lambda \, f(\nabla_{W_i}(L)) \qquad i = D, D-1, \ldots, 1 \quad (2)$$

where the gradients $\nabla_{W_i}(L)$ are transformed using the transformation function 'f' before the weight update.

*Equal contribution.

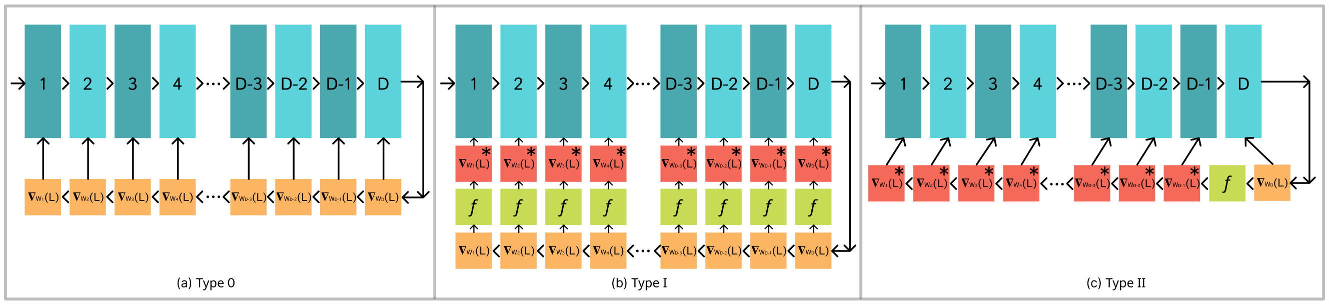


Figure 1: Different ways of altering gradients in the backward pass. Blue blocks denote a different layers. Orange blocks indicate the backward graph with unmodified gradients. Green blocks represent transformation functions, while red blocks indicate transformed gradients.

Type II: Gradient modification at a point very early in the backward graph: In this type of modification, the gradient is altered at a very early stage in the backward computation graph and then all subsequent gradients are generated using the values obtained with the modified gradients. We illustrate this type of modification in Fig. 1(c). Because of the chain rule, network parameters whose gradients are connected to the point of alteration in the computation graph also gets subsequently altered. It can be described as:

$$W_D^{t+1} = W_D^t - \lambda\, f(\nabla_{W_D}(L)) \quad (3)$$

$$W_i^{t+1} = W_i^t - \lambda\, \nabla_{W_i}(L)^* \qquad i = D-1, \dots, 1 \quad (4)$$

where the gradient $\nabla_{W_D}(L)$ is first transformed using the transformation function 'f' and then this transformed gradient is propagated through the rest of the backward graph. All other gradient vectors $\nabla_{W_i}(L)^*$ are computed as is, but because of the early injection of the transformed gradient $\nabla_{W_D}(L)$, all other gradient vectors that are connected to the transformed gradient through the chain rule ($\nabla_{W_i}(L)^*$, i=D-1,...,1), gets subsequently altered.

Type I is computationally more expensive than type II as it requires altering the gradients of each and every parameter individually. Type II modification recomputes gradients at each and every location through the natural flow of backpropagation. We propose PowerGrad Transform (PGT), a type II method that modifies the gradients at the softmax layer.

The following are the major contributions of this paper:

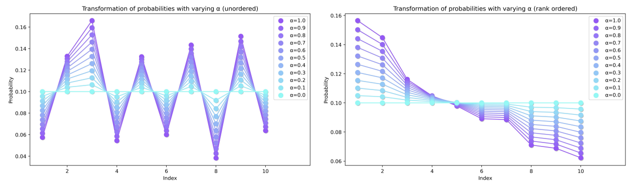
1. We propose PowerGrad Transform, which decouples the backward and the forward passes of neural network training and enables a considerably better fit to the dataset. PGT is a performance enhancement method that alters the gradients in the backward pass before the update step leading to accelerated training and a significant boost in the network's predictive performance.

2. We perform theoretical analysis of the properties of the PowerGrad transformation and explore its effect on weight parameters and gradients, logits and loss values (section 3.1). We show that in non-BN networks, PGT can be used to increase the network's convergence rate and improve the final accuracy.

3. We provide complete results from a variety of models (non-BN and BN ResNets, SE-ResNets) using the ImageNet dataset. We empirically conclude that PGT helps a network to improve by locating a more optimum convergence point. Additionally, we conduct ablation studies and compare its effects to those of regularization methods.

2. Related Works

Gradient Clipping [12] is a gradient modification method that involves clipping/altering the gradients with respect to a predefined threshold value during backward propagation through the network and updating the weights using the clipped gradients [18][14]. By rescaling the gradients, the weight updates are likewise rescaled, significantly reducing the risk of an overflow or underflow [11]. GC can be used for training networks without batch-normalization. At larger batch sizes, the clipping threshold in GC becomes highly sensitive and requires extensive finetuning for various models, batch sizes, and learning rates. As we demonstrate later in our studies, GC is not as effective in improving the performance of non-normalized networks. Adaptive Gradient Clipping [1] is developed to further enhance backward pass gradients than what is performed by GC. It takes into account the fact that the ratio of the gradient norm to the weight norm can provide an indication of the expected change in a single step of optimization. Label smoothing, introduced by Szegedy et al. [15], utilizes smoothing of the ground truth labels as a method to impose regularization on the logits and the weights of the neural network. AGC performs better than GC in non-normalized networks. However, we show that PGT outperforms both in networks such as ResNets.



(a) Unordered distribution (b) Rank ordered distribution

Figure 2: Illustration of the impact of PowerGrad Transform on probability values of an arbitrary distribution (number of classes, $C = 10$). $\alpha = 1$ indicates the unmodified distribution. 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.

Knowledge distillation (KD) [6] is a process in which two networks are trained with hard and soft labels alternatively. Variants of knowledge distillation include self-distillation [19], identical student network distillation [3], channel distillation [4], regularizing wrong predictions [17] and representation or embedding based knowledge distillation [16]. Even though both PGT and KD requires probability manipulation, the key difference is that in the latter the transformation is applied in the forward pass, while PGT is a backward pass modification only. In distillation settings the temperature parameter is a part of the network's computation graph. In the case of PowerGrad Transform, we directly tamper the gradients without introducing any change in the forward pass. PGT differs from self-knowledge distillation as it neither introduces any additional sub-modules nor creates different ensembles to improve the performance of the model. PGT follows the standard neural network training mechanism with modified gradients.

3. PowerGrad Transform

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 is defined as $p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$. 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 $L = -\sum_{i=1}^{C} q_i \log{(p_i)}$ where q_i is the ground truth label of the i^{th} class for a particular training example. By standard gradient update rule, we can calculate the gradient of the loss with respect to the logits which takes the form $\frac{\partial L}{\partial z_i} = p_i - q_i$.

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

$$p'_i = \frac{p_i^{\alpha}}{\sum_{j=1}^{C} p_j^{\alpha}} \qquad i = 1, \dots, C \quad 0 \le \alpha \le 1 \tag{5}$$

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

$$\widehat{\frac{\partial L}{\partial z_i}} = p'_i - q_i \tag{6}$$

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).

3.1. Properties of the PowerGrad transformation

We use the same setup as described in section 3. To explore the properties of PGT, we start by investigating the effect of the transform on the softmax probabilities.

Lemma 1. For any arbitrary probability distribution P with probability values given by p_i for $i = 1, \dots, C$, the corresponding transformed probability values p'_i given by [Eq. 5] has a threshold $\left(\sum_{j=1}^{C} p_j^{\alpha}\right)^{\frac{1}{\alpha-1}}$ and

$$\begin{aligned} p'_i \ge p_i, \text{ if } p_i \le \left(\sum_{j=1}^{C} p_j^{\alpha}\right)^{\frac{1}{\alpha-1}} \\ p'_i < p_i, \text{ if } p_i > \left(\sum_{j=1}^{C} p_j^{\alpha}\right)^{\frac{1}{\alpha-1}} \end{aligned} \tag{7}$$

We call this threshold, the *stationary threshold*. The stationary threshold is that value of p_i that does not change after the transformation. Therefore, when p_i is greater than the *stationary threshold*, $p'_i < p_i$.

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 $i = 1, \dots, C$,.

Proof. From Eq. (7), we see that the stationary threshold at $\alpha = 0$ is $1/C$. Also, following from the definition of the transformed probabilities (Eq. 5) 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.

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. Therefore, 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 in the following theorem.

Theorem 1. For any arbitrary probability distribution P with probability values p_i for $i = 1, \dots, C$, the stationary threshold of the transformed distribution P' with probability values $p'_i = \frac{p_i^\alpha}{\sum_{j=1}^{C} p_j^\alpha}, 0 \le \alpha \le 1$ 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.

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

If $a_1, \dots, a_n$ and $b_1, \dots, b_n$ are non-negative numbers, then using the log sum inequality,
we get $\sum_{j=1}^{n} a_j \log\left(\frac{a_j}{b_j}\right) \ge \left(\sum_{j=1}^{n} a_j\right)\log\left(\frac{\sum_{j=1}^{n} a_j}{\sum_{j=1}^{n} b_j}\right)$.

Setting $a_j = p_j^\alpha$ and $b_j = 1$, we get the following lower bound

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

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

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

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

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

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

Substituting (12) in (10),

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

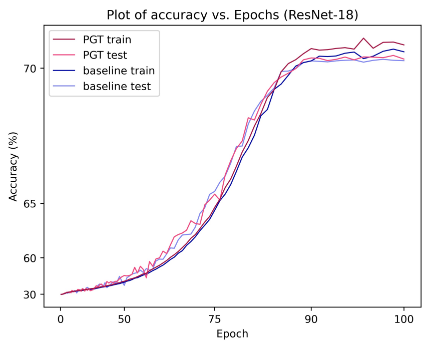
We conclude that the stationary threshold is a monotonic non-decreasing function with respect to α. Also the derivative of PGT function with respect to the true probabilities is non-negative which in turn means that the transformation is an order-preserving map. All values greater than the threshold move towards the threshold after transformation and all values below the 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.

4. Experiments

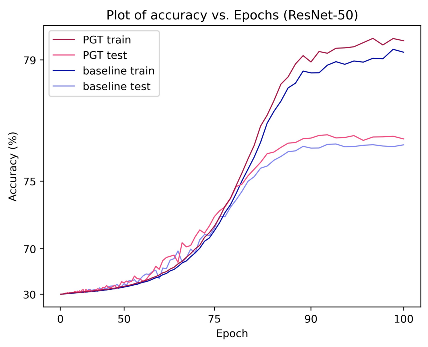
We perform experiments on different variants ResNets using the ImageNet-1K dataset [2]. All models are trained on four V100 GPUs with a batch size of 1024. We utilize a common set of hyperparameters for all experiments, which are as follows: 100 epoch budget, 5 epochs linear warmup phase beginning with a learning rate of 4×10^{-4} and ending with a peak learning rate of 0.4, a momentum of 0.9 and weight decay of 5×10^{-4}, the SGD Nesterov optimizer and mixed precision. 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 [10]. We find $\alpha = 0.25$ and $\alpha = 0.05$ to be good choices for ResNet-18 and ResNet-50, though larger values such as $\alpha = 0.3$ also have good performance as well. The experimental results are shown in Table 1, and we mention the value of the PGT hyperparmeter (α) in each experiment.[1].

In our experiments with Squeeze-and-Excitation variant of ResNet-50 i.e. SE-ResNet-50[7], we observe significant improvements; a 0.97% boost in training accuracy and a 0.734% increase in test accuracy. In our experiments with ResNet-50, we find an increase of 0.5% and 0.656% performance enhancement over the cosine scheduler baseline for training and testing respectively. The corresponding improvement over the step scheduler baseline for ResNet-50 is 0.57% (training) and 0.524% (testing). ResNet-101 sees a higher improvement in training fit, 0.81% to be exact, while the improvement over the test set is 0.362%. Smaller networks such as SE-ResNet-18 and ResNet-18 sees accuracy boosts which are smaller but nevertheless positive. With consistent improvements in training accuracies across all cases, we conclude PGT helps networks learn better representations and arrive at better optimas during convergence.

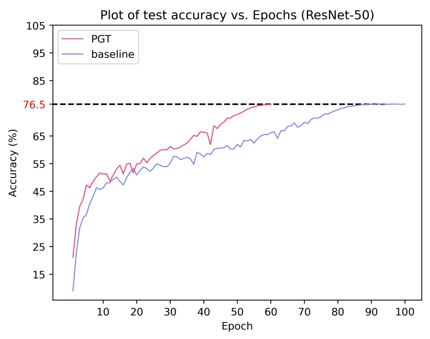
[1]Reproducible code, training recipes, checkpoints and training logs are provided at: https://github.com/skalien/power-grad-transform



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



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



(c) Plot of test accuracies vs. time/iterations (ResNet-50)

Figure 3: Log-log plots of training and test accuracies and comparison with baseline of batch-normalized variants: **(a)** ResNet-18 ($\alpha = 0.25$), **(b)** ResNet-50 ($\alpha = 0.3$). **(c)** Training speed comparison between PGT (60 epochs) and baseline (100 epochs). They both converge to the same test accuracy (76.5%) on ImageNet-1K. PGT's accelerated training saves 40% of the epoch budget.

Table 1: Results and comparison table for networks trained on ImageNet-1K. Best training and test accuracies are highlighted in red and blue respectively. Accuracy differences are highlighted in yellow.

Model	**Scheduler**	**Method**	**PGT (α)**	**Train Acc.(%)**	**Train Diff(%)**	**Test Acc.(%)**	**Test Diff(%)**
SE-ResNet-50	Cosine	Baseline	-	81.5	**+0.97**	77.218	**+0.734**
		PGT	0.3	**82.47**		**77.952**	
ResNet-50	Cosine	Baseline	-	79.18	**+0.5**	76.56	**+0.656**
		PGT	0.05	**79.68**		**77.216**	
ResNet-50	Step	Baseline	-	78.99	**+0.57**	75.97	**+0.524**
		PGT	0.05	**79.56**		**76.494**	
ResNet-101	Cosine	Baseline	-	82.29	**+0.81**	77.896	**+0.362**
		PGT	0.3	**83.1**		**78.258**	
SE-ResNet-18	Cosine	Baseline	-	71.42	**+0.18**	71.09	**+0.346**
		PGT	0.25	**71.6**		**71.436**	
ResNet-18	Step	Baseline	-	69.95	**+0.35**	69.704	**+0.14**
		PGT	0.25	**70.3**		**69.844**	
ResNet-18	Cosine	Baseline	-	70.38	**+0.15**	70.208	**+0.09**
		PGT	0.25	**70.53**		**70.298**	

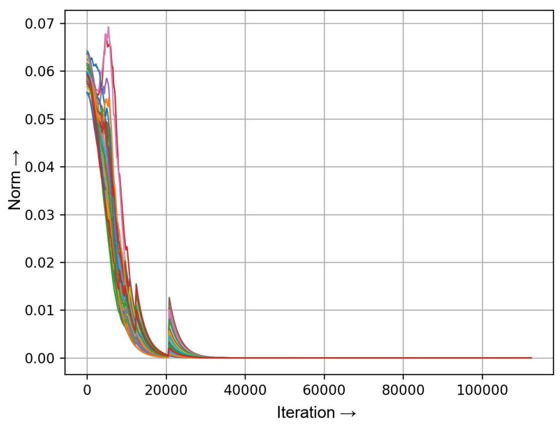


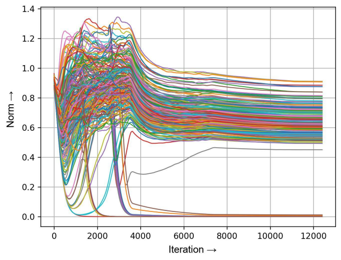
Figure 4: Zeroing out of feature maps in the second layer non-normalized ResNet-18.

Per epoch training and test accuracy plots of ResNet-18 and ResNet-50 (both with and without PGT) are shown in Fig. 3(a,b). Practioners can also choose to accelerate training and save as much as 40% of the epoch budget [Fig. 3(c)].

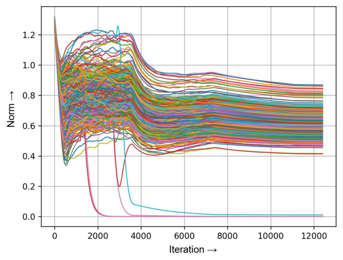
4.1. Empirical studies on networks without Batch Normalization

We examine issues that occur in non-normalized networks (networks without BN layers). We use ResNet-18 [5] as the foundation model trained on ImageNet-1K [13]. Deeper networks such as ResNet-34 and ResNet-50 are impossible to train without Batch Normalization courtesy of the increased depth and so we solely focus on ResNet-18. We designate different layers with their corresponding layer indices. Throughout the training process, we monitor variations in the the per-filter L2-norm of each layer's weights. In Fig. 5(a), some filters of layer 11 achieve a norm of zero during training. We refer to this event as 'Zeroing Out' (Fig. 4), and it occurs when one of the channels (or filters) of a weight tensor gets fully filled with zeros and such filters do not contribute at all to determine the input-output relationship of a dataset, as the feature tensor it produces is also filled with zeros for the corresponding filter. When a filter once zeroes out, it does not recover with further training, as all gradients that it receives in future iterations are all zeros.

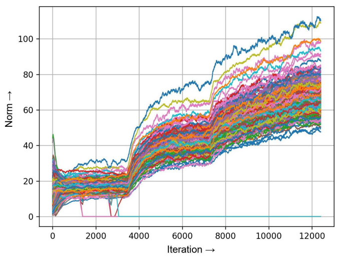
Fig. 5(b) is the plot of the final conv layer's filters (layer 19) and the features output of final global average pooling (GAP) layer respectively. We observe a number of filters and features in the final layer completely zeroing out. The feature vector after the GAP layer [Fig. 5(c)] directly inter-



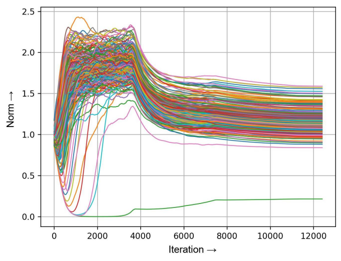
(a) Layer 11
baseline



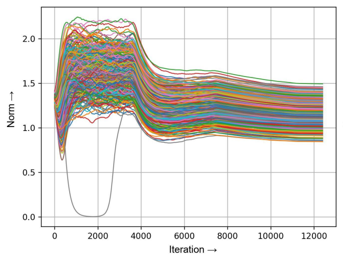
(b) Layer 19
baseline



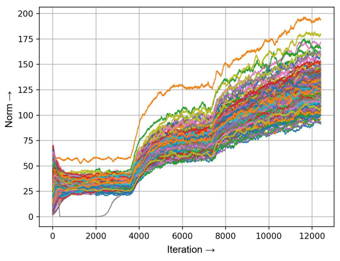
(c) GAP
features
baseline



(d) Layer 11
PGT



(e) Layer 19
PGT



(f) GAP
features
PGT

Figure 5: Norm vs iter. plots demonstrating layer characteristics (the zero out phenomena) and the efficacy of PGT over baseline. Each colour represents a different filter or feature vector of a particular layer of a non-BN variant of ResNet-18.

faces with the fully connected layer. Therefore any zeroed out features leads to permanent information loss, as it does not contribute to the learning of decision boundaries in the fully connected layer. Also, zeroed out weights tensors lead to zeroed out gradients hence stopping training for all subsequent iterations leading to a collapse in training for the affected layers. With large batch sizes, it is possible that an entire layer zeroes out as shown in the figure below. Gradient modification methods such as PGT can alleviate the zeroing out phenomena as we observe that the number of zeroed out filters has considerably reduced [Fig. 5(d, e)]. The final feature tensor [Fig. 5(f)] with PGT enabled, does not contain any zeroed out regions indicating that information loss is mitigated as the features pass on from the feature extracting layers to the fully connected layer.

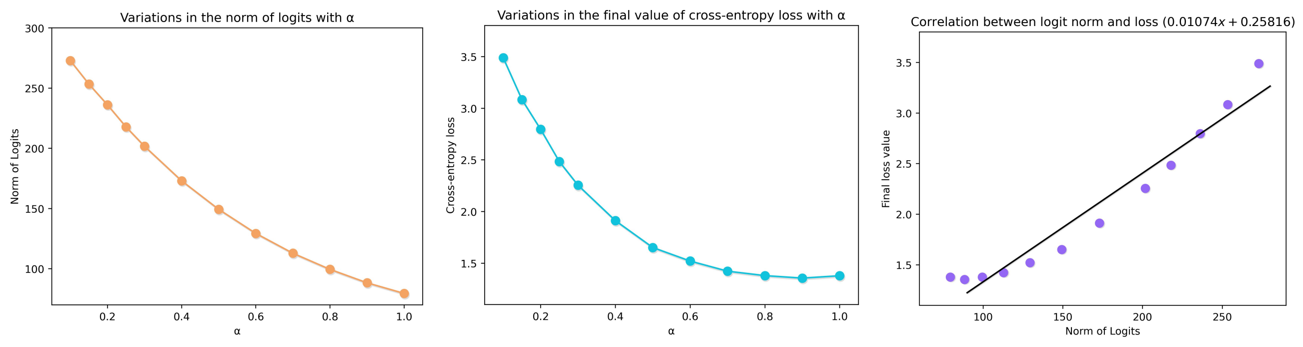
In Table 2 we find that the baseline performance for high batch sizes (1024) is drastically inferior to baselines for other batch sizes. PGT helps regain some of the lost performance by 0.682% (65.498% vs. 64.816%). At a batch size of 512, invoking PGT improves the training accuracy baseline by 1.48% and the test accuracy baseline by 0.684%, while at a batch size of 256, the improvement in training and test accuracies are 1.11% and 1.018% respectively. In comparison, the test accuracy improvements obtained by GC and AGC at batch size of 256 is much less at 0.27% and 0.5%, respectively. On the training accuracy front, since we get a significant boost (1.48% at batch size of 512 and 1.11% at batch size of 256), it leads us to infer that when PowerGrad Transform is used, the network fits the training dataset more tightly and the convergence optima is significantly superior. Combining GC with PGT is not very significant. When AGC and PGT are combined, we see a tremendous increase in test accuracy of over 2.06% over the baseline.

Table 2: Results for non-normalized ResNet-18 on ImageNet-1K. Best training and test accuracies are highlighted in red and blue respectively. Top differences in training and test accuracies are marked in yellow.

Batch Size	**Method**	**PGT** (α)	**Train Acc.(%)**	**Train Diff(%)**	**Test Acc.(%)**	**Test Diff(%)**
1024	Baseline	-	66.27	-	64.816	-
1024	PGT	0.92	**66.62**	**+0.35**	**65.498**	**+0.682**
512	Baseline	-	68.02	-	66.552	-
512	PGT	0.25	**69.5**	**+1.48**	**67.236**	**+0.684**
256	Baseline	-	68.86	-	66.796	-
256	GC	-	69.04	+0.18	67.064	+0.268
256	AGC	-	69.06	+0.2	67.298	+0.502
256	PGT	0.25	**69.97**	**+1.11**	**67.814**	**+1.018**
256	Baseline	-	68.86	-	66.796	-
256	GC+PGT	0.25	68.67	-0.19	67.088	+0.292
256	AGC+PGT	0.25	**70.92**	**+2.06**	**68.856**	**+2.06**

4.2. Effect on Loss, Logits and other metrics

As seen in Fig. 6(a), the logit norm increases as α falls from 1 to 0. We also see that the final value of the loss rises as α drops, and that the logit norm and the final value of



(a)
Logit vs. α

(b)
Cross-entropy loss vs. α

(c)
Logit-Loss correlation

Figure 6: 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: $(0.01074x + 0.25816)$.

the loss are linearly correlated Fig. 6(c). What is surprising is that: *it is possible to achieve higher accuracies even though the loss values are larger during convergence*. In our studies with ResNet-50, for instance, if PGT is not invoked, the corresponding test accuracy is 76.56% [Table 1] with a cross-entropy loss value of 0.852. However, if PGT is activated with $\alpha = 0.05$, the corresponding test accuracy is 77.216% [Table 1] with a loss value of 2.360. This is markedly different from the coupled gradient descent based training procedures. PGT enables the network to arrive at such optima where the loss values are high, but both training and test perfomance is better. From these plots of logits and losses we find that with the inclusion of PGT in the training process, the model can have access to such regions of the loss landscape that is otherwise inaccessible to traditional gradient based training procedures.

5. Ablation Study

We conduct ablation studies to investigate the effects of PowerGrad Transform for different values of the hyperparameter (α), where we use the ResNet-50 architecture and combine our proposed method with different schedulers, regularization techniques and different values of α. We report our findings in Table 3. First we examine the effect of PGT on the step scheduler baseline in order to later compare it to the cosine scheduler baseline. **Row-1)** We begin with the step scheduler baseline (75.97%). **Row-2)** PGT improves upon the step scheduler baseline (test set) by a substantial margin with 0.524% (76.494% as opposed to 75.97%). **Row-3)** Introducing the cosine scheduler yields a 0.59% improvement (76.56% vs. 75.97%) over the step scheduler. **Row-4)** After introducing label smoothing, the test accuracy relative to the cosine scheduler baseline increases by only 0.138% (from 76.56% to 76.698%). **Row-5)** However, introducing PGT with $\alpha = 0.3$ alone (without label smoothing) improves the cosine scheduler baseline by 0.326% (76.886% vs. 76.56%). **Row-6)** Combining PGT ($\alpha = 0.3$) with label smoothing improves the performance on the test set further by 0.408% (from 76.56% to 76.968%) and reduces the generalization gap (from 2.54% to 1.5%). However, the impact of combining PGT with label smoothing can vary depending on the value of the hyperparameter (α). **Row-7)** With a PGT hyperparameter value of $\alpha = 0.05$, we notice the greatest performance improvement, 1.246% over the step scheduler test baseline and 0.656% over the cosine scheduler test baseline. **Row-8)** Adding label smoothing to PGT ($\alpha = 0.05$) hurts performance even though it reduces the generalization gap.

Table 3: Ablation study for ResNet-50 on ImageNet-1K.

#Row	**Scheduler**	**Label Smoothing**	**PGT (α)**	**Train Acc.(%)**	**Test Acc.(%)**	**Gap(%)**
1.	Step	✗	✗	78.99	75.97	3.02
2.	Step	✗	0.3	79.56	76.494	3.066
3.	Cosine	✗	✗	79.18	76.56	2.62
4.	Cosine	0.1	✗	78.81	76.698	2.112
5.	Cosine	✗	0.3	79.43	76.886	2.544
6.	Cosine	0.1	0.3	78.47	76.968	1.502
7.	Cosine	✗	0.05	79.68	77.216	2.464
8.	Cosine	0.1	0.05	77.69	76.39	1.3

6. Conclusion

PowerGrad Transform enables a significantly better fit to the dataset as measured by training and test accuracy metrics. With PGT, gradient behavior is enhanced and weights attain better values in normalized networks and degenerate states are avoided in non-BN networks. We provide theoretical analyses of the transformation. With different network topologies and datasets, we are able to show the potential of PGT and explore its impacts from an empirical standpoint. PGT helps the network to improve its learning capabilities by locating a more optimum convergence point and simultaneously speeds up training.

References

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

[2] 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*, pages 248–255. Ieee, 2009.

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

[4] 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.

[5] 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.

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

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

[8] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988.

[9] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

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

[11] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2(417):1, 2012.

[12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[14] Samuel Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. In *International Conference on Machine Learning*, pages 9058–9067. PMLR, 2020.

[15] 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*, pages 2818–2826, 2016.

[16] 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.

[17] 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*, pages 13876–13885, 2020.

[18] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*, 2019.

[19] 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*, pages 4320–4328, 2018.