FIXNORM: DISSECTING WEIGHT DECAY FOR TRAINING DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Weight decay is a widely used technique for training Deep Neural Networks(DNN). It greatly affects generalization performance, but the underlying mechanisms are not fully understood. Recent works show that for layers followed by normalizations, weight decay mainly affects the *effective learning rate*. However, although normalizations have been extensively adopted in modern DNNs, layers such as the final fully-connected layer do not satisfy this precondition. For these layers, the effects of weight decay are still unclear. In this paper, we comprehensively investigate the mechanisms of weight decay and find that except for influencing effective learning rate, weight decay has another distinct mechanism that is equally important: affecting generalization performance by controlling *cross-boundary risk*. These two mechanisms together give a more comprehensive explanation for the effects of weight decay. Based on this discovery, we propose a new training method called **FixNorm**, which discards weight decay and directly controls the two mechanisms. We also propose a practical method to tune hyperparameters of FixNorm, finding near-optimal solutions 2$\sim$3 times faster than Bayesian Optimization. On ImageNet classification task, training EfficientNet-B0 with FixNorm achieves 77.7%, which outperforms the original baseline by a clear margin. Surprisingly, when scaling MobileNetV2 to the same FLOPS and applying the same tricks with EfficientNet-B0, training with FixNorm achieves 77.4%, which shows the importance of well-tuned training procedures and further verifies the effectiveness of our approach. We set up more well-tuned baselines using FixNorm, to facilitate fair comparisons in the community.

1 INTRODUCTION

Weight decay is an important trick and has been widely used in training Deep Neural Networks. By constraining the weight magnitude, it is widely believed that weight decay regularizes the model and improve generalization performance(Krizhevsky et al., 2012; Bös, 1996; Bos & Chug, 1996; Krogh & Hertz, 1992). Recently, a series of works (Van Laarhoven, 2017; Zhang et al., 2018; Hoffer et al., 2018) propose that for layers followed by normalizations, such as BatchNormalization(Ioffe & Szegedy, 2015), the main effect of weight decay is increasing the *effective learning rate*(ELR). Take the widely adopted Conv-BN block as an example. Since BatchNormalization is scale-invariant, the weight norm of the convolution layer does not affect the block's output, which contradicts the regularization effect of weight decay. On the contrary, the weight norm affects the step size of the weight update, e.g., a larger weight norm results in smaller step size. By constraining the weight norm from unlimitedly growing, weight decay increases the step size of weight update, thus increasing the ELR.

This interesting discovery arouses our questions: Does it covers the full mechanism(s) of weight decay? If the answer is true, weight decay would be unnecessary since its effect fully overlaps with the original learning rate. Hoffer et al. (2018) shows that the performance can be recovered when training without weight decay by applying an LR correction technique. However, the LR correction coefficient at each step depends on the original training statistics(with weight decay), which makes this technique only a verification of the effective learning rate hypothesis, but can not be used as a practical training method. Moreover, the ELR hypothesis only applies to layers followed by normalizations, and there are layers that do not satisfy this requirement. For example, the final fully-connected layer that is commonly used in classification tasks. For these layers, the

effects of weight decay are usually omitted (Hoffer et al., 2018), or simply the original weight decay is preserved (Zhang et al., 2018). These problems indicate that the mechanisms of weight decay are still not fully understood.

In this paper, we try to investigate the above problems. For the convolution layers that followed by normalizations, we find that simply fixing the overall weight norm to a constant fully recovers the effect of weight decay. For the final fully-connected layer, we find that there is a special effect introduced by weight decay, which influences the generalization performance by controlling the **cross-boundary risk**. This mechanism is as important as the former investigated ELR, and they together capture most of the effects of weight decay. We then propose a new training scheme called **FixNorm**, which discards the weight decay and directly controls the effects of two main mechanisms. By using FixNorm, we fully recover the performance of popular CNNs on large scale classification dataset ImageNet(Deng et al., 2009). Further, we show that the hyperparameters of FixNorm can be easily tuned and propose an efficient tuning method **FixNorm-tune**, which find near-optimal solutions 2$\sim$3 times faster than Baeysian Optimization. Specifically, by applying FixNorm-tune, we achieve 77.7%(+0.4%) with EfficientNet-B0, 79.5%(+0.3%) with EfficientNet-B1, 73.97%(+1.9%) with MobileNetV2.

Training tricks and network tricks show significant impacts on performance, introducing difficulties in tuning and bringing ambiguities to comparisons. We show that this can be mitigated by using FixNorm-tune. For example, by simply scaling MobileNetV2 to the same FLOPS and applying the same tricks of EfficientNet-B0, training with FixNorm achieves 77.4% top-1 accuracy, while the default training process only gets 76.72%. To facilitate fairer comparisons, we apply our FixNorm-tune method to representative CNN architectures and set up new baselines under different settings.

Our contributions can be summarized as follows:

- Except for increasing the effective learning rate, we discover a new mechanism of weight decay which controls the cross-boundary risk, and give a better understanding of weight decay's effect on generalization performance.
- We propose a new training scheme called FixNorm that discards the weight decay and directly controls the effects of two main mechanisms, which fully recovers the accuracy of weight decay training and makes hyperparameters easier to tune.
- We propose a practical method to tune hyperparameters of FixNorm which is 2$\sim$3 times faster than Bayesian Optimization and demonstrate its efficiency, robustness, and SOTA performance on large scale datasets like ImageNet, MS COCO and Cityscapes.
- By using our approach, we establish well-tuned baselines for popular networks, which we hope can facilitate fairer comparisons in the community.

2 Dissecting weight decay for training Deep Neural Networks

2.1 Revisiting the effective learning rate hypothesis

We aim at further understanding the mechanisms of weight decay. Towards this, we first briefly revisit the effective learning rate(ELR) hypothesis. As noted in Hoffer et al. (2018), when BN is applied after a linear layer, the output is invariant to the channel weight vector norm. Denoting a channel weight vector with $\boldsymbol{w}$ and $\hat{\boldsymbol{w}} = \boldsymbol{w}/\|\boldsymbol{w}\|_2$, channel input as $\boldsymbol{x}$, we have

$$\text{BN}(\boldsymbol{w}\boldsymbol{x}) = \text{BN}(\|\boldsymbol{w}\|_2\hat{\boldsymbol{w}}\boldsymbol{x}) = \text{BN}(\hat{\boldsymbol{w}}\boldsymbol{x}) \tag{1}$$

In such case, the gradient is scaled by $1/\|\boldsymbol{w}\|_2$:

$$\frac{\partial \text{BN}(\boldsymbol{w}\boldsymbol{x})}{\partial(\boldsymbol{w})} = \frac{\partial \text{BN}(\|\boldsymbol{w}\|_2\hat{\boldsymbol{w}}\boldsymbol{x})}{\partial(\|\boldsymbol{w}\|_2\hat{\boldsymbol{w}})} = \frac{1}{\|\boldsymbol{w}\|_2}\frac{\partial \text{BN}(\hat{\boldsymbol{w}}\boldsymbol{x})}{\partial \hat{\boldsymbol{w}}} \tag{2}$$

This scale invariance makes the key feature of the weight vector is its *direction*. As in Hoffer et al. (2018), when the weights are updated through stochastic gradient descent with learning rate η

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\nabla \boldsymbol{L}_t(\boldsymbol{w}_t) \tag{3}$$

one can derive that the step size of the weight direction is approximately proportional to

$$\hat{\boldsymbol{w}}_{t+1} - \hat{\boldsymbol{w}}_t \propto \frac{\eta}{\|\boldsymbol{w}_t\|_2^2} \tag{4}$$

Based on this formulation, the ELR hypothesis can be explained as follows: when applying weight decay to layers followed by normalization, it prevents weight norm from unlimitedly growing, which preserves the step size of weight update, thus "increasing the effective learning rate".

However, this phenomenon is still not fully investigated. Hoffer et al. (2018) proposes a learning rate correction technique that can train DNNs to similar performance without weight decay. However, this technique needs to mimic the effective step size *from training with weight decay*, which can not be used as practical training methods. On the other hand, as the hypothesis is based on the scale invariance brought by normalizations, there are layers that do not satisfy this precondition. For example, the final fully-connected(FC) layers that commonly used in classification tasks. As experiments in Zhang et al. (2018)(Figure 4), there is a clear gap between whether weight decay is applied to the final FC layer. These problems indicate that the mechanisms of weight decay are still not fully understood.

2.2 Discarding weight decay for convolution layers

We first consider discarding weight decay for convolution layers. Since the ELR hypothesis indicates that the main effect of weight decay on convolution layers is produced by constraining weight vector norm, we investigate how weight vector norm changes during training. Denoting the weights in all convolution layers as a single verctor $\boldsymbol{W}^{\text{Conv}}$, we plot $\|\boldsymbol{W}^{\text{Conv}}\|_2$ in Fig 1. ResNet50(He et al., 2016) is trained on ImageNet for 100 epochs with $lr = 0.4$ and weght decay$\lambda = 0.0001$. Other settings follow general setups in section 3.

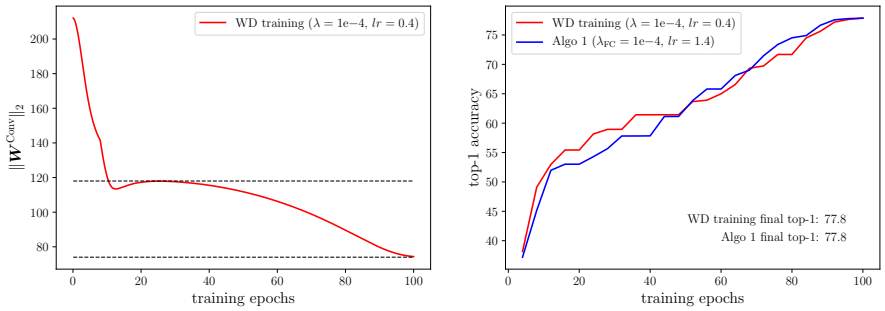

Figure 1: **Left:** $\|\boldsymbol{W}^{\text{Conv}}\|_2$ of WD training **Right:** top-1 accuracy for Algo 1 and WD training, both learning rates are found by gridsearch

From Fig 1 left, the curve can be divided into two phases. In the first few epochs, $\|\boldsymbol{W}^{\text{Conv}}\|_2$ decreases rapidly, which will increase the ELR according to the ELR hypothesis. However, this effect is duplicated with the learning rate warmup strategy, thus can be discarded. In the second phase, $\|\boldsymbol{W}^{\text{Conv}}\|_2$ changes slowly in a relatively stable range. We suppose that in this phase, the main effect is to keep ELR stable, while exactly following the trajectory of $\|\boldsymbol{W}^{\text{Conv}}\|_2$ is not necessary. Therefore, we propose to directly fix $\|\boldsymbol{W}^{\text{Conv}}\|_2$ to a constant, instead of implicitly controlling it with weight decay. We implement this by rescaling $\boldsymbol{W}^{\text{Conv}}$ to match the norm at initialization after each optimization step. The whole algorithm is summarized in Algo 1.

Algorithm 1 Fixing the weight norm of convolution layers

Input: initial learning rate lr, total steps T, weight decay on final FC layer λ_{FC}, training samples $\boldsymbol{x}$, corresponding labels $\boldsymbol{y}$

Initialization: random initialize weight vector $\boldsymbol{W_0}$

for t in 0, ..., $T-1$ **do**

  $\boldsymbol{x}, \boldsymbol{y} \leftarrow \text{BatchSampler}(t)$ ▷ sample a batch of data and lable

  $\widehat{\mathcal{L}}_t(\boldsymbol{W}_t) \leftarrow \sum \mathcal{L}(f(x; \boldsymbol{W}_t), y) + \frac{1}{2}\lambda_{\text{FC}}\|\boldsymbol{W}_t^{\text{FC}}\|_2^2$ ▷ only apply weight decay on final FC layer

  $\eta_t \leftarrow \text{GetLRScheduleMultiplier}(t)$ ▷ get current learning rate multiplier value

  $\boldsymbol{W}_{t+1} \leftarrow \text{Optimizer}(\boldsymbol{W}_t, lr \times \eta_t, \nabla\widehat{\mathcal{L}}_t(\boldsymbol{W}_t))$ ▷ update weight with some optimizer

  $\boldsymbol{W}_{t+1}^{\text{Conv}} \leftarrow \boldsymbol{W}_{t+1}^{\text{Conv}} \frac{\|\boldsymbol{W}_0^{\text{Conv}}\|_2}{\|\boldsymbol{W}_{t+1}^{\text{Conv}}\|_2}$ ▷ rescale Conv weight to the norm at initialization

end for

Since the fixed weight norm in Algo 1 is substantially different from weight decay(WD) training, their optimal learning rates are different. To be fair, we grid search the lr for both Algo 1 and WD training and compare the best performance. As shown in Fig 1 right, Algo 1 achieves same top-1

accuracy of WD training. These results demonstrate that for convolution layers followed by BN, weight decay can be discarded. Note that we preserve λ^{FC} for the final FC layer, which will be addressed in the next subsection.

2.3 Effects on final fully-connected layers

To investigate the effects of weight decay on the final FC layer, we first set $\lambda^{\text{FC}} = 0$ for Algo 1. The experiment result shows that the accuracy drops from 77.8% to 76.4%, which means that weight decay has essential effects on the final FC layer. Are these effects associated with the ELR hypothesis? To verify this, we first try to make the final FC layer scale-invariant. We apply three modifications on Algo 1: (1) replace original FC layer with WN-FC layer; (2) replace $\boldsymbol{W}^{\text{Conv}}$ in line 6 of Algo 1 with $\boldsymbol{W}^{\text{Conv+FC}}$; (3) set $\lambda^{\text{FC}} = 0$. This modified algorithm is denoted as Algo 1@WN-FC. WN-FC layer is normal FC layer applied with weight normalization(Salimans & Kingma, 2016). Original FC and WN-FC layer are formulated as follows(g is a learnable parameter):

$$\text{FC}(x; \boldsymbol{W}^{\text{FC}}) = x^{\mathsf{T}} \boldsymbol{W}^{\text{FC}} \tag{5}$$

$$\text{WN-FC}(x; \boldsymbol{W}^{\text{FC}}, g) = \frac{x^{\mathsf{T}} \boldsymbol{W}^{\text{FC}}}{\|\boldsymbol{W}^{\text{FC}}\|_2} \times g \tag{6}$$

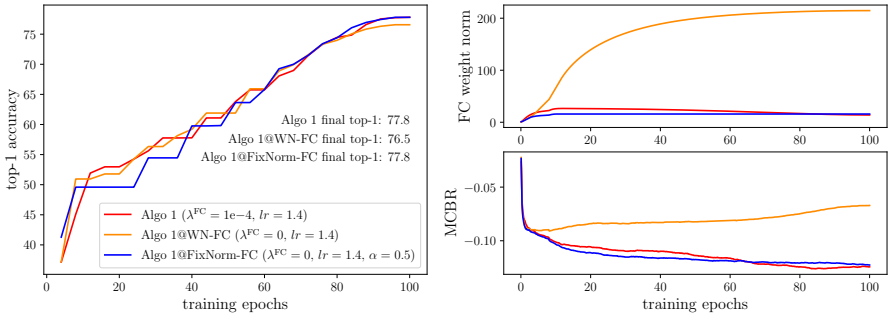

Figure 2: Training ResNet50 on ImageNet with Algo 1, Algo 1@WN-FC and Algo 1@FixNorm-FC. **Left:** top-1 accuracy **Top right:** weight norm of the final FC layer ($\|\boldsymbol{W}^{\text{FC}}\|_2$ for FC, g for WN-FC and FixNorm-FC) **Bottom right:** MCBR

We compare Algo 1 and Algo 1@WN-FC by training ResNet50 on ImageNet. As can be seen in Fig 2 left, there is still a clear gap between Algo 1 and Algo 1@WN-FC. Since Algo 1@WN-FC already preserves ELR, this gap implies that weight decay has additional effects beyond preserving ELR on the final FC layer.

Now lets take a closer look at the WN-FC layer. Combine equation 6 with softmax cross-entropy loss $\mathcal{L}$, $s_i = \frac{x^{\mathsf{T}} \boldsymbol{W}_i}{\|\boldsymbol{W}\|_2} g$ denotes the logits value of class i, p_i denotes the probability of class i, k for the label class and j for other classes. We have(for full derivations, please refer to Appendix B):

$$\mathcal{L}(x, k) = -\log p_k = -\log \frac{e^{s_k}}{\sum e^{s_j}} \tag{7}$$

$$-\frac{\partial \mathcal{L}(x, k)}{\partial x} = \frac{g}{\|\boldsymbol{W}\|_2} \sum_{j \neq k} p_j (\boldsymbol{W}_k - \boldsymbol{W}_j) \tag{8}$$

We visualize Equation 8 in Fig 3. It shows that the gradient is actually driving x from the other class center $\boldsymbol{W}_j$ towards the label class center $\boldsymbol{W}_k$, where the magnitude mainly depends on p_j and g. Note that g will continuously grow through training(larger g makes loss lower when x is correctly classified). In the meantime, p_j will shrink faster because of the nature of softmax transform, thus weakening the overall gradient. This will leave x being closer to the class boundary between $\boldsymbol{W}_j$ and $\boldsymbol{W}_k$(larger $\cos\beta$). In this case, x is more likely to cross the class boundary, e.g., be misclassified, at a slight distribution shift. In other words, it will lead to poor generalization. To quantitively verify this explaination, we define **Mean Cross-Boundary Risk**(MCBR)

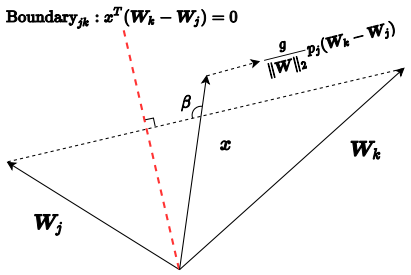

Figure 3: Ilustration of feature space

for a batch of training samples:

$$\text{MCBR}(B, \boldsymbol{W}) = \frac{1}{|B| \cdot (\#class - 1)} \sum_{(x,y) \in B} \sum_{j \neq y} \cos(x, \boldsymbol{W}_j - \boldsymbol{W}_y) \tag{9}$$

MCBR shows how much x is lean to the class boundaries, ranging from -1 to 1. We suppose that the larger weight norm of the final FC layer(or g for WN-FC layer) will lead to higher MCBR and worse generalization performance. We compare these metrics for Algo 1 and Algo 1@WN-FC in Fig 2. It can be observed that without constraint, g of Algo 1@WN-FC continuously grows and leads to higher MCBR compared to Algo 1, which explains why Algo 1@WN-FC has worse generalization performance.

Based on these analyses, we propose a new FC layer that constrains g in equation 6 from exceeding a given upper bound, denoted as FixNorm-FC. The upperbound is controled by a hyperparameter a. $\sqrt{\#class}$ normalizes the upper bound across different number of classes.

$$\text{FixNorm-FC}(x; \boldsymbol{W}^{FC}, g, \alpha) = \frac{x^{\mathsf{T}} \boldsymbol{W}^{FC}}{\|\boldsymbol{W}^{FC}\|_2} \times \min(g, \alpha * \sqrt{\#class}) \tag{10}$$

We replace the WN-FC layer with the FixNorm-FC layer in Algo 1@WN-FC, denoted as Algo 1@FixNorm-FC. We choose a proper value for α and show the results for this new algorithm in Fig 2. By simply constraining the upper limit of g, Algo 1@FixNorm-FC maintains low MCBR and fully closes the accuracy gap. This result reveals the true effect of weight decay on final FC layers: constraining the weight norm, controlling the **cross-boundary risk**, and finally influencing the generalization performance.

Now we can summarise the two main effects of weight decay: for convolution layers followed by normalizations, weight decay preserves ELR; for final FC layers, weight decay controls the **cross-boundary risk**. Algo 1@FixNorm-FC(referred to **FixNorm** for simplicity) fully recovers these two effects. Beyond that, there is an additional advantage: its hyperparameters are easier to tune. We will show this in 2.4 and 3.1.

2.4 Tuning hyperparameters for FixNorm training

Section 2.2 and 2.3 investigate the two main effects of weight decay. While normal weight decay training controls these two effects through lr and λ, FixNorm training controls them through lr and α. Since these hyperparameters greatly influence the final accuracy, it is important to know how to tune them.

For normal weight decay training, many works (Smith, 2018; Loshchilov & Hutter, 2017) show that optimal values for lr and λ are tightly coupled. This means that they must be tuned jointly, which is usually inefficient. Fortunately, our analysis can explain why they are tightly coupled: weight decay controls the two effects with a single hyperparameter. For example, when one changes the value of λ, the ELR and the cross-boundary risk are both affected, then lr needs to be adjusted accordingly to get the optimal ELR.

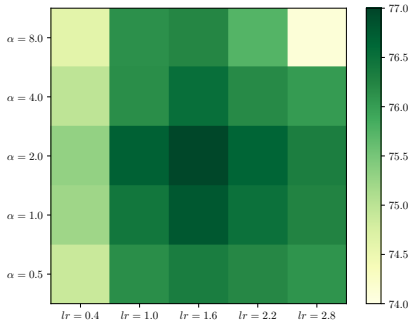

Figure 4: Top-1 accuracy for different lr and α. ResNet50 trained for 50 epochs

For FixNorm training, however, lr and α should not be coupled since they control the two effects separately. To verify this, we grid search lr and α and show the corresponding top-1 accuracy in Fig 4. It clearly shows that the optimal value of lr does not depends on the value of α and vice versa. This suggests that we can tune lr and α *separately*, which significantly reduces the cost.

We propose a simple yet effective approach to tune lr and α for FixNorm training. We introduce two priors to tune lr efficiently.

- Top-1 accuracy is approximately a concave function of lr
- The best lr for shorter training is usually larger than that for longer training

The first prior is mainly an empirical finding, while the second one can be partially explained by the correlation between generalization performance and weight distance from initialization(Hoffer et al., 2017): shorter training may require larger lr to travel far enough from the initialization in weight space to generalize well. These two priors motivate us to *use best lr under shorter training as an upper bound for that under longer training*. This strategy can adaptively shrink the search range and locate the best lr in a wide range with relatively low cost. After the best lr is found, we fix it and grid search for the best α. The overall method is summarized in Algo 2(**FixNorm-tune**).

Algorithm 2 Tuning lr and α for FixNorm training

Input: number of lr tuning rounds N, learning rate range $[lr_{min}, lr_{max}]$, learning rate split number K, training steps of each lr tuning round $\boldsymbol{T} = [T_0, T_1, ..., T_{N-1}]$ where $T_i \leq T_{i+1}$, alpha candidates $[\alpha_0, \alpha_1, ..., \alpha_{m-1}]$
Output: α_{best}, lr_{best}, acc_{best}
Initialization: $\alpha_{best} = \alpha_0$, $lr_{best} =$ NULL, $acc_{best} = 0$

Phase 1 - Find lr_{best}

for r in 0, ..., $N-1$ **do**
  $\boldsymbol{LR} \leftarrow$ UniformSample(lr_{min}, lr_{max}, K) ▷ uniformly sample K values from current lr range
  $\boldsymbol{Acc} \leftarrow \{\text{FixNormTrain}(LR_k, \alpha_{best}, T_r) | \ k \in \{0, ..., K-1\}\}$ ▷ run FixNorm training for each lr
  $idx \leftarrow \arg\max \boldsymbol{Acc}$ ▷ find the index of the best accuracy
  $lr_{max} \leftarrow \boldsymbol{LR}_{idx}$ ▷ update the lr upper bound by the corresponding lr of the best accuracy
  if $\boldsymbol{Acc}_{idx} > acc_{best}$ **then** ▷ update the best accuracy and the best lr
    $acc_{best} \leftarrow \boldsymbol{Acc}_{idx}$
    $lr_{best} \leftarrow \boldsymbol{LR}_{idx}$
  end if
end for

Phase 2 - Find α_{best}

$\boldsymbol{Acc} \leftarrow \{\text{FixNormTrain}(lr_{best}, \alpha_i, T_{N-1}) | \ i \in \{1, ..., m-1\}\}$ ▷ fix $lr = lr_{best}$ and run FixNorm training for each α
$idx \leftarrow \arg\max \boldsymbol{Acc}$ ▷ find the index of the best accuracy
if $\boldsymbol{Acc}_{idx} > acc_{best}$ **then** ▷ update the best accuracy and the best α
  $acc_{best} \leftarrow \boldsymbol{Acc}_{idx}$
  $\alpha_{best} \leftarrow \alpha_{idx}$
end if

3 EXPERIMENTS

General setups We perform experiments on ImageNet classification task (Deng et al., 2009) which contains 1.28 million training images and 50000 validation images. Our general training settings are mainly adapted from He et al. (2019), which include Nesterov Accelerated Gradient (NAG) descent(Nesterov, 1983), one-cycle cosine learning rate decay(Loshchilov & Hutter, 2016) with linear warmup at first 4 epochs(Goyal et al., 2017) and label smoothing with $\epsilon = 0.1$(Szegedy et al., 2016). We do not use mixup augmentation(Zhang et al., 2017). All the models are trained on 16 Nvidia V100 GPUs with a total batch size of 1024. Other settings follow reference implementations of each model. We leave experiments for detection, segmentation and group normalization(Wu & He, 2018) in Appendix D.

FixNorm-tune setups For Algo 2, we set $N = 2$, learning rate range $[0.2, 3.2]$, $K = 5$, $\boldsymbol{T} = [0.2T_{max}, T_{max}]$ where T_{max} is the max training steps, α candidates [0.5, 1.0, 2.0, 4.0, 8.0, 16.0]. The search contains two lr tuning rounds and one α tuning round. The total computational resources consumed are $K \times 0.2T_{max} + K \times T_{max} + (6-1) \times T_{max} = 11T_{max}$, which is 11 times of a single training process.

3.1 FIXNORM-TUNE ON RESNET50-D AND MOBILENETV2

To demonstrate the effectiveness of our method, we first apply Algo 2 on two well-studied architectures: ResNet50-D(He et al., 2019) and MobileNetV2(Sandler et al., 2018). We follow He et al. (2019) and train 120 epochs for ResNet50-D and 150 epochs for MobileNetV2. Their reference top-1 accuracies reported are 78.37% and 72.04%. We also adopt Bayesian Optimization(BO)(Snoek et al., 2012) to search the learning rate and weight decay for normal weight decay training (BO+WD). We use the same learning rate range $[0.2, 3.2]$ for BO, and the weight decay range is set to $[0.00001, 0.0005]$. Results are shown in Table 1.

Table 1: Results for FixNorm-tune and BO+WD

	FixNorm-tune			BO + WD			Reference		
	lr	α	top-1(%)±std	lr	λ	top-1(%)±std	lr	λ	top-1(%)±std
ResNet50-D	1.4	0.5	**78.62±0.054**	0.53	8.6e-5	78.53±0.047	0.4	1e-4	78.37±0.051
MobileNetV2	0.5	16.0	**73.20±0.032**	0.64	2.2e-5	72.84±0.024	0.2	4e-5	72.04±0.027

As in Table 1, both FixNorm-tune and BO+WD outperform reference settings by a clear margin. Note that the reference settings have already been heavily refined in He et al. (2019). We further show details of FixNorm-tune and BO+WD in Fig 5. Our method shows two advantages compared to BO+WD. First, our method finds better solutions at a lower cost. As shown in Fig 5 left, the cost of our method is 11 times of normal training while BO+WD requires 25 and 35, and our final results are even better than that found by BO+WD. Second, our method is more stable and barely needs meta-tuning, while BO itself has many tunable meta hyperparameters (Lindauer et al., 2019). We use the same FixNorm-tune setups for all the experiments, including in Table 2. These setups are intuitive, and the method performs consistently well across all the settings.

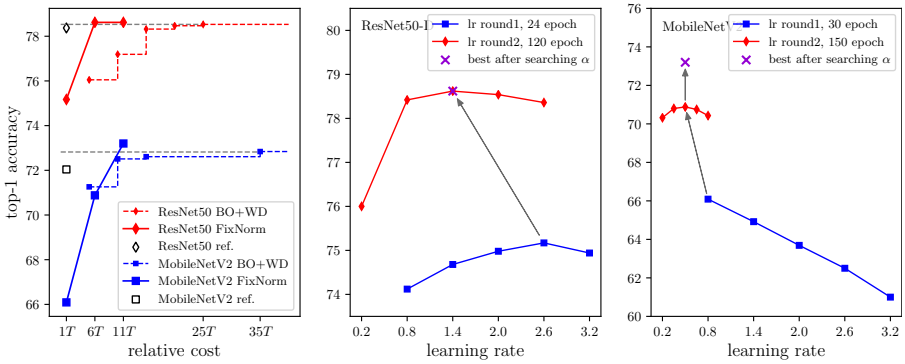

Figure 5: **Left:** search progress for FixNorm and BO+WD **Middle:** FixNorm search details for ResNet50-D **Right:** FixNorm search details for MobileNetV2

To better understand our method, we show more details about FixNorm-tune in Fig 5 middle and right for ResNet50-D and MobileNetV2, respectively. There are two lr searching rounds and one α searching round according to our setups. In the first lr round, both experiments starts with lr values in [0.8, 1.4, 2.0, 2.6, 3.2] (which are uniformly sampled from initial range [0.2, 3.2]) and train for $1/5$ total epochs(24 epochs and 30 epochs respectively). Best lr values are very different in this round: $lr_1 = 2.6$ for ResNet50-D and $lr_1 = 0.8$ for MobileNetV2. Taking these values as new upper bound, [0.2, lr_1] are further split, and corresponding lr values are evaluated in the second round, for full total epochs. The best lr values are 1.4 and 0.5 in this round. These values are fixed, and then α is searched. For ResNet50-D, the initial $\alpha = 0.5$ is already the best, while for MobileNetV2 a better $\alpha = 16.0$ is found. From Fig 5 one can find that the patterns match two priors introduced in section 2.4.

3.2 New state-of-the-arts with FixNorm-tune

Many powerful networks have been proposed recently. These networks usually adopt many tricks and hard to tune. To fully exploit the capabilities of these networks, we apply FixNorm-tune to optimize them further. We also apply advanced tricks to basic models like MobileNetV2 and ResNet50-D. These tricks are used by EfficientNet(Tan & Le, 2019), including SE-layer(Hu et al., 2018), swish activation(Ramachandran et al., 2017), stochastic depth training(Huang et al., 2016) and AutoAugment(Cubuk et al., 2019). The results are shown in Table 2. We can find that:

- Our method consistently outperforms reference settings. Strong baselines like EfficientNet can be further improved by our method, specifically +0.4% and +0.3% for B0 and B1.
- Tunning matters. When simply apply tricks to MobileNetV2 and scale to the same FLOPS with B0 and B1, FixNorm-tune achieves 77.4% and 79.18% top-1 accuracy, while default training settings only get 76.72 and 78.75. This difference can lead to unreliable conclusions when compared to EfficientNet.

- Best lr and α are different among models, even for the same model with different settings. This suggests that we should tune for each setting to fully exploit their performance.

Table 2: **Results with FixNorm-tune** models with * are applied with tricks, †means the result is obtained using lr and λ from basic settings(MobileNetV2 150 and ResNet50-D 120)

Model	#Epochs	Top-1	Top-1 (ref.)	#Params	#FLOPS	lr	α
MobileNetV2	150	73.20	72.04	3.5M	300M	0.5	16.0
MobileNetV2	350	73.97	73.38†	3.5M	300M	0.35	8.0
EfficientNet-B0	350	77.72	77.30	5.3M	384M	0.5	4.0
EfficientNet-B1	350	79.52	79.20	7.8M	685M	0.8	8.0
MobileNetV2×1.12*	350	77.40	76.72†	4.7M	386M	0.5	8.0
MobileNetV2×1.54*	350	79.18	78.75†	8.0M	682M	0.65	4.0
ResNet50-D	120	78.62	78.37	25.6M	4.3G	1.4	0.5
ResNet50-D	350	79.29	79.04†	25.6M	4.3G	1.1	0.5
ResNet50-D*	350	81.27	80.80†	28.1M	4.3G	1.1	1.0

4 Related works

We only highlight the most related works in this section and leave other works in Appendix E due to the page limit.

Understanding weight decay Recently, a series of works (Van Laarhoven, 2017; Zhang et al., 2018; Hoffer et al., 2018) propose that when combined with normalizations, the main effect of weight decay is increasing ELR, which is contrary to the previous understanding and motivates new perspectives. Van Laarhoven (2017) first introduces the ELR hypothesis and provides derivations for different optimizers, while both Hoffer et al. (2018); Zhang et al. (2018) give additional evidence supporting the hypothesis. Hoffer et al. (2018) also proposes norm-bounded Weight Normalization, which fixes the norm of each convolution layer separately. By doing this, their method fixes the ELR of each layer, which *highly depends on the initialization of each layer*. Differently, we fix the norm of all convolution layers as a whole and maintains the *global* ELR, which is more robust and demonstrates SOTA performance on large scale experiments. Layer-wise ELR controlling is an interesting problem and may lead to new perspectives for weight initialization techniques. Similar to Hoffer et al. (2018), Xiang et al. (2019) also proposes modifications to Weight Normalization based on ELR hypothesis. They identify the problems when using weight decay with Weight Normalization and propose $\epsilon-$shifted L_2 regularizer to constrain weight norm to ϵ with coefficient λ. Beyond the ELR hypothesis, Li & Arora (2019) derives a closed-form between learning rate, weight decay, momentum, and proposes an exponentially increasing learning rate schedule. Their work mainly discusses the linkage of three hyperparameters, while our work focuses on the underlying mechanisms of weight decay. Except for the ELR hypothesis, Loshchilov & Hutter (2017) identifies problems when applying weight decay to Adam optimizer, which improves generalization performance and decouples it from the learning rate. These works bring interesting perspectives for understanding weight decay, yet our work has distinct differences and contributions. First, our work investigates the effect on final FC layers and find a new mechanism that complements the understanding of weight decay on generalization performance, which is mostly ignored by previous works. Second, our method, including FixNorm and FixNorm-tune, are both concise and effective and demonstrate SOTA performance on large scale datasets.

5 Conclusion

In this paper, we find a new mechanism of weight decay on final FC layers, which affects generalization performance by controlling *cross-boundary risk*. This new mechanism complements the ELR hypothesis and gives a better understanding of weight decay. We propose a new training method called **FixNorm**, which discards weight decay and directly controls the two mechanisms. We also propose an effective, efficient, and robust method to tune hyperparameters of FixNorm, which can consistently find near-optimal solutions in a few trials. Experiments on large scale datasets demonstrate our methods, and a series of SOTA baselines are established for fair comparisons. We believe this work brings new perspectives and may motivate interesting ideas like controlling layer-wise ELR and automatically adjusting cross-boundary risk.

REFERENCES

Siegfried Bös. Optimal weight decay in a perceptron. In *International Conference on Artificial Neural Networks*, pp. 551–556. Springer, 1996.

Siegfried Bos and E Chug. Using weight decay to optimize the generalization ability of a perceptron. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pp. 241–246. IEEE, 1996.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Adam D Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(Oct):2879–2904, 2011.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 113–123, 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.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 1731–1741. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6770-train-longer-generalize-better-closing-the-generalization-gap-in-large-batch-pdf.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2160–2170, 2018.

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.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pp. 950–957, 1992.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. *arXiv preprint arXiv:1910.07454*, 2019.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Marius Lindauer, Matthias Feurer, Katharina Eggensperger, André Biedenkapp, and Frank Hutter. Towards assessing the impact of bayesian optimization's own hyperparameters. *arXiv preprint arXiv:1908.06674*, 2019.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pp. 543–547, 1983.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pp. 901–909, 2016.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018. URL `http://arxiv.org/abs/1803.09820`.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 2951–2959. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf`.

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.

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Twan Van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.

Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018. URL `http://arxiv.org/abs/1803.08494`.

Li Xiang, Chen Shuo, Xia Yan, and Yang Jian. Understanding the disharmony between weight normalization family and weight decay: $\epsilon-$shifted L_2 regularizer. *arXiv preprint arXiv:1911.05920*, 2019.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.

Yuhui Yuan and Jingdong Wang. Ocnet: Object context network for scene parsing. *arXiv preprint arXiv:1809.00916*, 2018.

Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. *arXiv preprint arXiv:1810.12281*, 2018.

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

A APPENDIX—ALGORITHMS

In section 2.3, we apply three modifications to Algo 1: (1) replace original FC layer with WN-FC layer; (2) replace $\boldsymbol{W}^{\text{Conv}}$ in line 6 of Algo 1 with $\boldsymbol{W}^{\text{Conv+FC}}$; (3) set $\lambda^{\text{FC}} = 0$. This modified algorithm is denoted as Algo 1@WN-FC. Here we show the full algorithm as follows:

Algorithm 1 @WN-FC

Input: initial learning rate lr, total steps T, training samples $\boldsymbol{x}$, corresponding labels $\boldsymbol{y}$
Replace: replace original final FC layer with WN-FC layer
Initialization: random initialize weight vector $\boldsymbol{W_0}$

1: **for** t in 0, ..., $T-1$ **do**
2: $\quad \boldsymbol{x}, \boldsymbol{y} \leftarrow \text{BatchSampler}(t)$ ▷ sample a batch of data and lable
3: $\quad \widehat{\mathcal{L}}_t(\boldsymbol{W}_t) \leftarrow \sum \mathcal{L}(f(x; \boldsymbol{W}_t), y) + \frac{1}{2}\lambda_{\text{FC}} \|\boldsymbol{W}_t^{\text{FC}}\|_2^2$ ▷ only apply weight decay on final FC layer
4: $\quad \eta_t \leftarrow \text{GetLRScheduleMultiplier}(t)$ ▷ get current learning rate multiplier value
5: $\quad \boldsymbol{W}_{t+1} \leftarrow \text{Optimizer}(\boldsymbol{W}_t, lr \times \eta_t, \nabla \widehat{\mathcal{L}}_t(\boldsymbol{W}_t))$ ▷ update weight with some optimizer
6: $\quad \boldsymbol{W}_{t+1}^{\text{Conv+FC}} \leftarrow \boldsymbol{W}_{t+1}^{\text{Conv+FC}} \frac{\|\boldsymbol{W}_0^{\text{Conv+FC}}\|_2}{\|\boldsymbol{W}_{t+1}^{\text{Conv+FC}}\|_2}$ ▷ rescale Conv and FC weight to the norm at initialization
7: **end for**

The Algo 1@FixNorm-FC is similar to Algo 1@WN-FC, while the only different is that WN-FC layer is replaced by FixNorm-FC layer. The full algorithm is shown as follows:

Algorithm 1 @FixNorm-FC

Input: initial learning rate lr, total steps T, momentum μ, training samples $\boldsymbol{x}$, corresponding labels $\boldsymbol{y}$
Replace: replace original FC layer by FixNorm-FC layer
Initialization: random initialize weight vector $\boldsymbol{W_0}$

1: **for** t in 0, ..., $T-1$ **do**
2: $\quad \boldsymbol{x}, \boldsymbol{y} \leftarrow \text{BatchSampler}(t)$ ▷ sample a batch of data and lable
3: $\quad \widehat{\mathcal{L}}_t(\boldsymbol{W}_t) \leftarrow \sum \mathcal{L}(f(x; \boldsymbol{W}_t), y) + \frac{1}{2}\lambda_{\text{FC}} \|\boldsymbol{W}_t^{\text{FC}}\|_2^2$ ▷ only apply weight decay on final FC layer
4: $\quad \eta_t \leftarrow \text{GetLRScheduleMultiplier}(t)$ ▷ get current learning rate multiplier value
5: $\quad \boldsymbol{W}_{t+1} \leftarrow \text{Optimizer}(\boldsymbol{W}_t, lr \times \eta_t, \nabla \widehat{\mathcal{L}}_t(\boldsymbol{W}_t))$ ▷ update weight with some optimizer
6: $\quad \boldsymbol{W}_{t+1}^{\text{Conv+FC}} \leftarrow \boldsymbol{W}_{t+1}^{\text{Conv+FC}} \frac{\|\boldsymbol{W}_0^{\text{Conv+FC}}\|_2}{\|\boldsymbol{W}_{t+1}^{\text{Conv+FC}}\|_2}$ ▷ rescale Conv and FC weight to the norm at initialization
7: **end for**

B APPENDIX—DERIVATIONS

The complete derivations of equation 8 are as follows. $\mathcal{L}$ denotes the softmax cross-entropy loss, $s_i = \frac{x^\intercal \boldsymbol{W}_i}{\|\boldsymbol{W}\|_2} g$ denotes the logits value of class i, p_i denotes the probability of class i, k for the label class and j for other classes. We have,

$$\mathcal{L}(x,k) = -\log p_k = -\log \frac{e^{s_k}}{\sum e^{s_i}} \tag{11}$$

$$\tag{12}$$

We first derive $\frac{\partial p_k}{\partial s_k}$:

$$\frac{\partial p_k}{\partial s_k} = \frac{e^{s_k}\sum e^{s_j} - e^{s_k}e^{s_k}}{\left(\sum e^{s_j}\right)^2} \tag{13}$$

$$= \frac{e^{s_k}}{\sum e^{s_j}} - \left(\frac{e^{s_k}}{\sum e^{s_j}}\right)^2 \tag{14}$$

$$= p_k - {p_k}^2 \tag{15}$$

$$= p_k(1-p_k) \tag{16}$$

also for $j \neq k$, we have,

$$\frac{\partial p_k}{\partial s_j} = \frac{-e^{s_k}e^{s_j}}{\left(\sum e^{s_i}\right)^2} \tag{17}$$

$$= -p_k p_j \tag{18}$$

combine them with $\frac{\partial s_i}{\partial x} = \frac{\boldsymbol{W}_i}{\|\boldsymbol{W}\|_2} g$, we have,

$$\frac{\partial p_k}{\partial x} = \frac{\partial p_k}{\partial s_k}\frac{\partial s_k}{\partial x} + \sum_{j\neq k}\frac{\partial p_k}{\partial s_j}\frac{\partial s_j}{\partial x} \tag{19}$$

$$= p_k(1-p_k)\frac{\boldsymbol{W}_k}{\|\boldsymbol{W}\|_2} g + \sum_{j\neq k} -p_k p_j \frac{\boldsymbol{W}_j}{\|\boldsymbol{W}\|_2} g \tag{20}$$

$$= \frac{p_k g}{\|\boldsymbol{W}\|_2}\left((1-p_k)\boldsymbol{W}_k + \sum_{j\neq k} -p_j \boldsymbol{W}_j\right) \tag{21}$$

$$= \frac{p_k g}{\|\boldsymbol{W}\|_2}\left(\boldsymbol{W}_k + \sum -p_i \boldsymbol{W}_i\right) \tag{22}$$

$$= \frac{p_k g}{\|\boldsymbol{W}\|_2}\left(\sum p_i(\boldsymbol{W}_k - \boldsymbol{W}_i)\right) \tag{23}$$

$$= \frac{p_k g}{\|\boldsymbol{W}\|_2}\left(\sum_{j\neq k} p_j(\boldsymbol{W}_k - \boldsymbol{W}_j)\right) \tag{24}$$

$$\tag{25}$$

and finally,

$$-\frac{\partial \mathcal{L}(x,k)}{\partial x} = -\frac{\partial \mathcal{L}(x,k)}{\partial p_k}\frac{\partial p_k}{\partial x} \tag{26}$$

$$= \frac{1}{p_k}\frac{p_k g}{\|\boldsymbol{W}\|_2}\sum_{j\neq k} p_j(\boldsymbol{W}_k - \boldsymbol{W}_j) \tag{27}$$

$$= \frac{g}{\|\boldsymbol{W}\|_2}\sum_{j\neq k} p_j(\boldsymbol{W}_k - \boldsymbol{W}_j) \tag{28}$$

C Appendix—More Details

Parameters other than convolution and FC weights For modern CNNs like ResNet or MobileNet, the majority of parameters come from weights of convolution and FC layers. Other parameters are mainly biases and γ and β of BN layers. As in He et al. (2019), the no-bias-decay strategy is applied to avoid overfitting, which does not use weight decay on these parameters. We empirically find that this strategy does not harm performance, so we adopt this strategy in our FixNorm method, which means we do not fix the norm of biases and γ and β parameters. Experiments in Table 2 also include architectures with SE-blocks, which have FC layers that are not followed by normalizations. Since these layers are not directly followed by softmax cross-entropy loss, we find that they do not suffer from the problem identified in section 2.3. So we simply replace these layers with WN-FC layers and add the weights into the norm-fixing process(line 6 in Algo 1@FixNorm-FC). In summary, our FixNorm method considers weights of convolution layers, final FC-layers, and FC layers of SE-blocks. Other parameters like biases and γ and β of BN layers are excluded from the norm fixing process.

D Appendix—More results

D.1 Extending FixNorm-FC for pixel-wise classification

The FixNorm-FC is proposed to replace the original final FC layer in classification tasks. There are other forms of classification tasks that do not use FC layers, such as segmentation and object detection. For segmentation, the models are usually fully convolutional, and the last convolution layer is used for pixel-wise classification. This also applies to Region Proposal Networks(Ren et al., 2015) used in object detection or methods that produce dense detections like RetinaNet(Lin et al., 2017). These tasks still share the nature of the classification task, therefore the cross-boundary risk still needs to be controlled. Our FixNorm-FC layer can be easily extended to these tasks because the final convolution layer can be viewed as a normal FC layer that shares weight across spatial positions. Denote the weight of the final convolution layer as $\boldsymbol{W}^{\text{Conv}}$ with shape $[c_{out}, c_{in}, k_h, k_w]$, we define,

$$\text{FixNorm-Conv}(x; \boldsymbol{W}^{\text{Conv}}, g, \alpha) = \frac{\text{Conv}(x, \boldsymbol{W}^{\text{Conv}})}{\|\boldsymbol{W}^{\text{Conv}}\|_2} \times \min(g, \alpha * \sqrt{c_{out}}) \tag{29}$$

This layer is a straightforward extension of the FixNorm-FC layer, which will be used in experiments on segmentation and detection later.

D.2 Experiments on Cityscapes

Setups The Cityscapes dataset(Cordts et al., 2016) is a task for urban scene understanding. We follow the basic training settings in Yuan & Wang (2018). We use 2975 images for training and 500 images for validation. The initial learning rate is set as 0.01 and weight decay as 0.0005. The original image size is 1024×2048, and we use a crop size of 769×769. All the models are trained on 4 Nvidia V100 GPUs for 40000 iterations with a total batch size of 8. The poly learning rate policy is used. We use the ResNet-101 + Base-OC(Yuan & Wang, 2018) as the baseline model.

Modifications We replace the last convolution layer with FixNorm-Conv when trained with our FixNorm method.

FixNorm-tune setups The main settings are the same with that on ImageNet, such as $N = 2$, $K = 5$, $T = [0.2T_{max}, T_{max}]$, α candidates as [0.5, 1.0, 2.0, 4.0, 8.0, 16.0]. The only difference is that we adapt the learning rate range to [0.005, 0.1]. The reason is that the models are finetuned on a pre-trained model from ImageNet, therefore the default learning rate is smaller.

The results are shown in table 3. FixNorm-tune clearly outperforms the baseline, and the improvements are larger when the cosine learning rate is applied. Note that the mIoU actually drops when using the cosine learning rate with default hyperparameters, while FixNorm-tune finds the appropriate hyperparameters and improves the performance.

Table 3: **Results with FixNorm-tune on Cityscapes**

Model	#Iters	Val. mIoU(%)	hyperparameters
baseline	40000	78.7	$lr = 0.01$, $\lambda = 0.0005$
baseline w/ cosine lr	40000	78.3	$lr = 0.01$, $\lambda = 0.0005$
FixNorm-tune	40000	79.4	$lr = 0.0335$, $\alpha = 1.0$
FixNorm-tune w/ cosine lr	40000	79.7	$lr = 0.043$, $\alpha = 1.0$

D.3 Experiments on MS COCO

Setups To verify our FixNorm-tune method on object detection task, we train RetinaNetLin et al. (2017) on MS COCOLin et al. (2014). We following common practice and use the COCO trainval35k split, and report results on the minival split. We use the ResNet50-FPN backbone, while the base ResNet50 model is pre-trained on ImageNet. The RetinaNet is trained with stochastic gradient descent(SGD) on 8 Nvidia V100 GPUs with a total batch size of 16. The models are trained for 90k iterations with default learning rate 0.01, which is then divided by 10 at 60k and 80k iterations. The default weight decay is 0.0001. The α_{focal} is set to 0.25 and the γ_{focal} is set to 2.0. The standard smooth L_1 loss is used for box regression. We use horizontal image flipping as the only data augmentation, and the image scale is set to 800 pixels.

Modifications To make the RetinaNet compatible with our method, we add Weight Normalization layers to all the convolution layers that are not followed by normalizations (include layers in FPN and classification subnet and bounding-box prediction subnet), for all the models. We also replace the last convolution layer of the classification subnet with FixNorm-Conv when trained with our FixNorm method. The last convolution layer of the bounding-box prediction subnet is used for regression task, which does not suffer from the problem identified in section 2.3, so we do not replace it with FixNorm-Conv.

FixNorm-tune setups The main settings are the same with that on ImageNet, such as $N = 2$, $K = 5$, $T = [0.2T_{max}, T_{max}]$, α candidates as [0.5, 1.0, 2.0, 4.0, 8.0, 16.0]. As the models are finetuned on a pre-trained model, we use the same learning rate range of [0.005, 0.1] as in segmentation experiments.

The results are shown in table 4. FixNorm-tune clearly outperforms the baseline, and the improvements are larger when the the cosine learning rate is applied.

Table 4: **Results with FixNorm-tune on MS COCO**

Model	#Iters	Val. AP(%)	hyperparameters
baseline	90000	36.5	lr=0.01, wd=0.0001
baseline w/ cosine lr	90000	36.2	lr=0.01, wd=0.0001
FixNorm-tune	90000	36.9	lr=0.0145, $\alpha = 0.5$
FixNorm-tune w/ cosine lr	90000	37.1	lr=0.0145, $\alpha = 0.5$

D.4 EXPERIMENTS ON GROUP NORMALIZATION

Except for the widely used Batch Normalization layer, other types of normalization layers have been proposed. According to the ELR hypothesis, FixNorm should also work for them. To verify the effectiveness of our method, we conduct experiments with Group Normalization on ImageNet. We choose ResNet50-D in Table 2 and replace BN layers by Group Normalization layers with a group number of $G = 32$. As shown in Table 5, when trained with Group Normalization, FixNorm-tune consistently improves the top-1 accuracy over the reference result as when trained with BN.

Table 5: **Results for FixNorm-tune with Group Normalization** models with * are applied with tricks, †means the result is obtained using lr and λ from basic settings(ResNet50-D 120)

Model	#Epochs	Top-1	Top-1 (ref.)	#Params	#FLOPS	lr	α
ResNet50-D*	350	81.27	80.80†	28.1M	4.3G	1.1	1.0
ResNet50-D*(G=32)	350	80.92	80.31†	28.1M	4.3G	1.4	1.0

E APPENDIX—ADDITIONAL RELATED WORKS

Hyperparameter Optimization (HPO). Hyperparameter Optimization is an important topic for effectively training DNNs. One straightforward method is grid search, which is only affordable for a very limited number of hyperparameters since the combinations grow exponentially. Random search(Bull, 2011) is a popular alternative that selects hyperparameter combinations randomly, which can be more efficient when the resource is constrained. Bayesian Optimization(BO) (Brochu et al., 2010) further improves efficiency by building a model on historical information to guide the selection. Hyperband(Li et al., 2017) allocates different budgets to random configurations and rejects bad ones according to the performance obtained under low budgets. BOHB (Falkner et al., 2018) combines BO with Hyperband to select more promising configurations. Both Hyperband and BOHB highly relies on the assumption that performance under different budgets is consistent. However, this assumption is not always valid, and these methods may suffer from the low rank-correlation of performance under different budgets(Ying et al., 2019). While these methods are universal black-box optimization methods, our tuning method leverages more priors of hyperparameters. Our method suggests that with a better understanding of the underlying mechanisms, we can develop a method that is more effective and efficient.