THREE MECHANISMS OF WEIGHT DECAY REGULARIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Weight decay is one of the standard tricks in the neural network toolbox, but the reasons for its regularization effect are poorly understood, and recent results have cast doubt on the traditional interpretation in terms of L2 regularization. Literal weight decay has been shown to outperform L2 regularization for optimizers for which they differ. We empirically investigate weight decay for three optimization algorithms (SGD, Adam, and KFAC) and a variety of network architectures. We identify three distinct mechanisms by which weight decay exerts a regularization effect, depending on the particular optimization algorithm and architecture: (1) increasing the effective learning rate, (2) regularizing approximated input-output Jacobian norm, and (3) reducing the effective damping coefficient for second-order optimization. Our results provide insight into how to improve the regularization of neural networks.

1 INTRODUCTION

Weight decay is one of the standard tricks in the neural network toolbox; there are some classic theoretical results about generalization (Bos & Chug, 1996; Bös, 1996; Krogh & Hertz, 1992), and is used in a large fraction of neural networks today (Huang et al., 2017; He et al., 2016). In the gradient descent setting, weight decay can be derived from the gradient of the L2 norm of the weights; hence, weight decay has traditionally been interpreted as a form of L2 regularization. However, several recent findings cast doubt on this interpretation:

- Weight decay has sometimes been observed to improve training accuracy, not just generalization performance (e.g. Krizhevsky et al. (2012)).
- Loshchilov & Hutter (2017) found that when using Adam as the optimizer, literally applying weight decay (i.e. scaling the weights by a factor less than 1 in each iteration) was a far more effective regularizer than adding an L2 regularizer to the training objective.
- Weight decay is widely used in networks with batch normalization (BN) (Ioffe & Szegedy, 2015). In principle, weight decay regularization should have no effect in this case, since one can scale the weights by a small factor without changing the network's predictions. Hence, it does not meaningfully constrain the network's capacity.

The effect of weight decay remains poorly understood, and we lack clear guidelines for which tasks and architectures it is likely to help or hurt. A better understanding of the role of weight decay would help us design more efficient and robust neural network architectures.

In order to better understand the effect of weight decay, we experimented with both weight decay and L2 regularization applied to image classifiers using three different optimization algorithms: SGD, Adam, and Kronecker-Factored Approximate Curvature (KFAC) (Martens & Grosse, 2015). (In this paper, weight decay refers to rescaling the weights by a fixed factor less than 1 after each weight update.) Consistent with the observations of Loshchilov & Hutter (2017), we found that weight decay consistently outperformed L2 regularization in cases where they differ. Weight decay gave an especially strong performance boost to the KFAC optimizer, and closed most of the generalization gaps between first- and second-order optimizers, as well as between small and large batches.

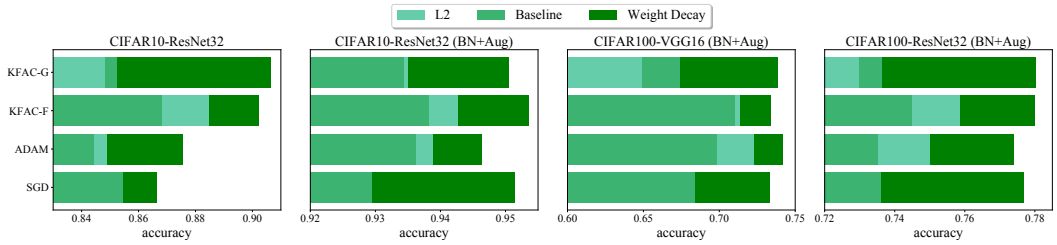


Figure 1: Comparison of test accuracy of the networks trained with different optimizers on both CIFAR10 and CIFAR100 datasets. We compare **Weight Decay** regularization to **L2** regularization and the **Baseline** (w/o regularization). To be noted, **BN+Aug** denotes with BN and data augmentation during training. **KFAC-G** and **KFAC-F** denote using Gauss-Newton and Fisher as the precondition matrix respectively. The results suggest that weight decay leads to improved performance over different optimizers and settings.

We then investigated the reasons for weight decay's performance boost. Surprisingly, we identified three distinct mechanisms by which weight decay has a regularizing effect, depending on the particular algorithm and architecture:

1. In our experiments with first-order optimization methods (SGD and Adam), we found that combined with BN, weight decay reduced the scale of the weights, hence increasing the effective learning rate, which increases the stochastic regularization effect (an effect previously observed by Srivastava et al. (2014); Neelakantan et al. (2015); Keskar et al. (2016)). As evidence, almost all of the generalization effect of weight decay was due to applying BN on layers (for which L2 regularization is meaningless). Furthermore, when we computed the effective learning rate for the network with weight decay, and applied the same effective learning rate to a network without weight decay, this captured the full regularization effect.

2. We show that when KFAC is applied to a linear network using the Gauss-Newton metric (KFAC-G), weight decay is equivalent to regularizing the squared Frobenius norm of the input-output Jacobian (which was shown by Novak et al. (2018) to improve generalization). Empirically, we found that the Gauss-Newton norm (which KFAC with weight decay is implicitly regularizing) is highly correlated with the Jacobian norm of real image classification networks, and that KFAC with weight decay significantly reduces the Jacobian norm.

3. Because the idealized, undamped version of KFAC is invariant to affine reparameterizations, the implicit learning rate effect described above should not apply. However, in practice the approximate curvature matrix is damped by adding a multiple of the identity matrix, and this damping is not scale-invariant. We show that without weight decay, the weights grow large, causing the effective damping term to increase, and may dominate the curvature term. Weight decay kept the effective damping term small, enabling KFAC to retain its second-order properties, and correspondingly improved generalization.

Hence, we have identified three distinct mechanisms by which weight decay improves generalization, depending on the optimization algorithm and network architecture. Our results underscore the subtlety and complexity of neural network training: the final performance numbers obscure a variety of complex interactions between phenomena. While more analysis and experimentation is needed to understand how broadly each of our three mechanisms applies (and to find additional mechanisms!), our work provides a starting point for understanding practical regularization effects in neural network training.

2 PRELIMINARIES

Natural gradient. For loss function $\mathcal{L}(\boldsymbol{\theta})$, its gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is the direction of change to $\boldsymbol{\theta}$ that causes $\mathcal{L}(\boldsymbol{\theta})$ to increase most rapidly, assuming that $\boldsymbol{\theta}$ resides in Euclidean space. The natural gradient generalizes the gradient to allow $\boldsymbol{\theta}$ to lie on a Riemannian manifold with metric tensor $\mathbf{C}(\boldsymbol{\theta})$.

For $\mathbf{C}$ to describe a Riemannian manifold, it must vary smoothly from point to point and must be positive definite for all $\boldsymbol{\theta}$, in which case $||\cdot||_{\mathbf{C}(\boldsymbol{\theta})}$ is a valid norm. Often, $\mathbf{C}(\boldsymbol{\theta})$ is taken to be Fisher

information matrix $\mathbf{F}$ for parameter distribution (Amari, 1998), where $\mathbf{F}$ is defined as[1]:

$$\mathbf{F} = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log p(y|\mathbf{x}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(y|\mathbf{x}, \boldsymbol{\theta})^\top\right] \tag{1}$$

In optimization, natural gradient can be derived by adapting the steepest descent formulation to use an alternative definition of (local) distance (Martens, 2014). For Fisher matrix, the particular distance function turns out to be

$$\mathrm{D}_{\mathrm{KL}}(p_{\boldsymbol{\theta}} \,\|\, p_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}) \approx \frac{1}{2}\Delta\boldsymbol{\theta}^\top \mathbf{F} \Delta\boldsymbol{\theta} \tag{2}$$

The distance function–KL divergence–is defined on the predictive distribution $p(y|\mathbf{x}, \boldsymbol{\theta})$, and therefore is invariant to the reparameterization of the neural network.

Gauss-Newton metric. Another sensible distance function is the L2 distance on the output of the neural network, i.e. $||f_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}} - f_{\boldsymbol{\theta}}||_2^2$. This is equivalent to using Gauss-Newton metric for the Riemannian manifold, which is defined as:

$$\mathbf{G} = \mathbb{E}\left[\mathbf{J}_{\boldsymbol{\theta}}^\top \mathbf{J}_{\boldsymbol{\theta}}\right] \tag{3}$$

where $\mathbf{J}_{\boldsymbol{\theta}}$ is the Jacobian of $f_{\boldsymbol{\theta}}(\mathbf{x})$ w.r.t the parameters $\boldsymbol{\theta}$. It is easy to see that $||f_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}} - f_{\boldsymbol{\theta}}||_2^2 \approx \Delta\boldsymbol{\theta}^\top \mathbf{G} \Delta\boldsymbol{\theta}$ locally. Heskes (2000) showed that the Fisher information matrix and the Gauss-Newton matrix are equivalent in the case of squared error loss. Later, Martens (2014) extended the equivalence to more general settings and showed that the Fisher information matrix is equivalent to *generalized* Gauss-Newton matrix when model prediction $p(y|\mathbf{x}, \boldsymbol{\theta})$ corresponds to exponential family model with natural parameters given by $f_{\boldsymbol{\theta}}(\mathbf{x})$, where the generalized Gauss-Newton matrix is given by

$$\mathbf{G} = \mathbb{E}\left[\mathbf{J}_{\boldsymbol{\theta}}^\top \mathbf{H}_\ell \mathbf{J}_{\boldsymbol{\theta}}\right] \tag{4}$$

where $\mathbf{H}_\ell$ is the Hessian of $\ell(y, z)$ w.r.t z, evaluated at $z = f_{\boldsymbol{\theta}}(\mathbf{x})$.

KFAC. Kronecker-factored approximate curvature (KFAC) Martens & Grosse (2015) uses a Kronecker-factored approximation to the curvature matrix to perform efficient approximate natural gradient updates. As shown in Luk & Grosse (2018), KFAC can be applied to general pullback metric. For convenience, we introduce KFAC here using the Fisher metric.

Considering l-th layer in the neural network whose input activations are $\mathbf{a}_l \in \mathbb{R}^{n_1}$, weight $\mathbf{W}_l \in \mathbb{R}^{n_1 \times n_2}$, and output $\mathbf{s}_l \in \mathbb{R}^{n_2}$, we have $\mathbf{s}_l = \mathbf{W}_l^\top \mathbf{a}_l$. Therefore, weight gradient is $\nabla_{\mathbf{W}_l}\mathcal{L} = \mathbf{a}_l(\nabla_{\mathbf{s}_l}\mathcal{L})^\top$. With this gradient formula, KFAC decouples this layer's fisher matrix $\mathbf{F}_l$ using mild approximations,

$$\begin{aligned}\mathbf{F}_l &= \mathbb{E}\left[\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\}\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\}^\top\right] = \mathbb{E}\left[\{\nabla_{\mathbf{s}_l}\mathcal{L}\}\{\nabla_{\mathbf{s}_l}\mathcal{L}\}^\top \otimes \mathbf{a}_l\mathbf{a}_l^\top\right] \\ &\approx \mathbb{E}\left[\{\nabla_{\mathbf{s}_l}\mathcal{L}\}\{\nabla_{\mathbf{s}_l}\mathcal{L}\}^\top\right] \otimes \mathbb{E}\left[\mathbf{a}_l\mathbf{a}_l^\top\right] = \mathbf{S}_l \otimes \mathbf{A}_l\end{aligned} \tag{5}$$

Where $\mathbf{A}_l = \mathbb{E}\left[\mathbf{a}\mathbf{a}^\top\right]$ and $\mathbf{S}_l = \mathbb{E}\left[\{\nabla_{\mathbf{s}}\mathcal{L}\}\{\nabla_{\mathbf{s}}\mathcal{L}\}^\top\right]$. Further, assuming between-layer independence, the whole fisher matrix $\mathbf{F}$ can be approximated as block diagonal consisting of layerwise fisher matrices $\mathbf{F}_l$. Decoupling $\mathbf{F}_l$ into $\mathbf{A}_l$ and $\mathbf{S}_l$ not only avoids the memory issue saving $\mathbf{F}_l$, but also provides efficient natural gradient computation.

$$\mathbf{F}_l^{-1}\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\} = \mathbf{S}_l^{-1} \otimes \mathbf{A}_l^{-1}\mathrm{vec}\{\nabla_{\mathbf{W}_l}\mathcal{L}\} = \mathrm{vec}[\mathbf{A}_l^{-1}\nabla_{\mathbf{W}_l}\mathcal{L}\mathbf{S}_l^{-1}] \tag{6}$$

As shown by equation 6, computing natural gradient using KFAC only consists of matrix transformations comparable to size of $\mathbf{W}_l$, making it very efficient.

3 The Effectiveness of Weight Decay

Our goal is to understand weight decay regularization in the context of training deep neural networks. Towards this, we first analyze the relationship between L2 regularization and weight decay in different optimizers, which has long been confused as being the same in the deep learning community.

The classical weight decay is defined by the following update rule:

$$\boldsymbol{\theta}^{t+1} \leftarrow (1 - \eta\beta)\boldsymbol{\theta}^t - \eta\nabla\mathcal{L}(\boldsymbol{\theta}^t) \tag{7}$$

[1]The underlying distribution for the expectation in equation 1 has been left ambiguous. Throughout the experiments, we sample the targets from the model's predictions, as done in Martens & Grosse (2015)

Table 1: Classification results on CIFAR-10 and CIFAR-100. **B** denotes BN while **D** denotes data augmentation including horizontal flip and random crop. **WD** denotes weight decay regularization.

Dataset	Network	B	D	SGD		ADAM		KFAC-F		KFAC-G	
					WD		WD		WD		WD
CIFAR-10	VGG16			83.20	84.87	83.16	84.12	85.58	89.60	83.85	**89.81**
		✓		86.99	88.85	88.45	88.72	87.97	89.02	88.17	**89.77**
		✓	✓	91.71	93.39	92.89	93.62	93.12	**93.90**	93.19	93.80
CIFAR-10	ResNet32			85.47	86.63	84.43	87.54	86.82	90.22	85.24	**90.64**
		✓		86.13	90.65	89.46	90.61	89.78	**91.24**	89.94	90.91
		✓	✓	92.95	95.14	93.63	94.66	93.80	**95.35**	93.44	95.04
CIFAR-100	VGG16	✓	✓	68.42	73.31	69.88	**74.22**	71.05	73.36	67.46	73.57
CIFAR-100	ResNet32	✓	✓	73.61	77.73	73.60	77.40	74.49	78.01	73.70	**78.02**

In the case of gradient descent update, weight decay is equivalent to L2 regularization, which adds an extra L2 norm $\frac{\beta}{2}||\boldsymbol{\theta}||_2^2$ to the loss $\mathcal{L}(\boldsymbol{\theta})$. However, it is not true for the preconditioned gradient update with the curvature matrix $\mathbf{C}$, e.g, natural gradient update. The preconditioned gradient update with L2 regularization is given by

$$\boldsymbol{\theta}^{t+1} \leftarrow (\mathbf{I} - \eta\beta\mathbf{C}^{-1})\boldsymbol{\theta}^t - \eta\mathbf{C}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^t) \tag{8}$$

whereas natural gradient with weight decay is slightly different

$$\boldsymbol{\theta}^{t+1} \leftarrow (1 - \eta\beta)\boldsymbol{\theta}^t - \eta\mathbf{C}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^t) \tag{9}$$

From equation 8 and equation 9, we can see that the main difference between L2 regularization and weight decay is whether the preconditioner $\mathbf{C}$ is applied to $\boldsymbol{\theta}^t$. Though the difference may appear subtle, we surprisingly find that it makes a noticeable difference in terms of generalization performance. We demonstrate the regularization effect of L2 and weight decay by conducting extensive experiments on image classification.

Initial Experiments. Our experiments were carried out on two different datasets: CIFAR-10 and CIFAR-100 (Krizhevsky & Hinton, 2009) with varied batch sizes. We test VGG16 (Simonyan & Zisserman, 2014) on CIFAR-10 and ResNet32 (He et al., 2016) on both CIFAR-10 and CIFAR-100. Moreover, we widen all convolutional layers in ResNet32 by a factor of 4 according to Zagoruyko & Komodakis (2016). To gain more insights, we also test ResNet32 on CIFAR-10 without both BN and data augmentation for the small batch setting (with a batch size of 128). In practice, we compute natural gradient using KFAC with different curvature matrix (see Appendix F for pseudo code of KFAC). For simplicity, we fix the γ and β in BN throughout all the experiments[2].

The results for small batch are summarized in Figure 1. Additionally, We plot the curves of test accuracies after tuning hyperparameters on held-out validation set. The final test accuracies are reported in Table 1. We also reported the final test accuracies of larger batch sizes in Table 3. Based on the initial results, we make the following observations regarding weight decay:

1. In all conducted experiments, weight decay regularization consistently improved the generalization performance and it is equally effective for different optimizers.
2. Interestingly, weight decay closed most of the generalization gaps between first- and second-order optimizers, as well as between small and large batches.
3. Finally, we notice that weight decay gave an especially strong performance boost to KFAC optimizer when BN is disabled.

In the following section, we seek to explain these phenomena. With further testing, we find that weight decay can work in unexpected ways, especially in the presence of BN.

4 Three Mechanisms of Weight Decay Regularization

4.1 Mechanism I: Higher Effective Learning Rate

Commonly, weight decay–when perceived to be similar to L2 regularization–is thought to improve generalization by regularizing the weights of neural networks to be small in magnitude. However,

[2]In practice, we found that fix these two parameters does not significantly impact results on image classification tasks.

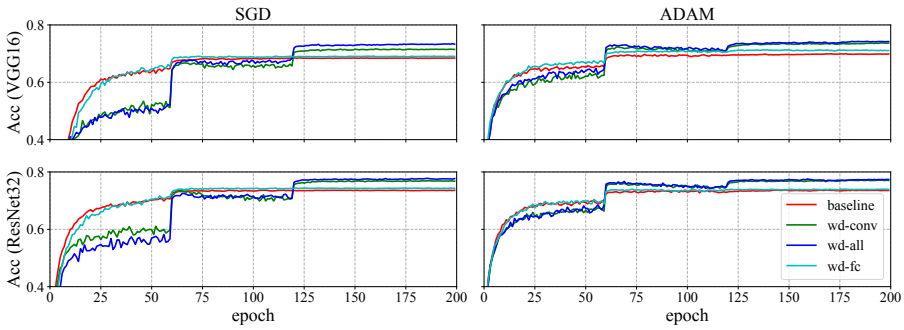


Figure 2: Test accuracy as a function of training epoch for SGD and Adam on CIFAR-100 with ResNet32 model and different weight decay regularization schemes. baseline is the model without weight decay; wd-conv is the model with weight decay applied to all convolutional layers; wd-all is the model with weight decay applied to all layers; wd-fc is the model with weight decay applied to last layer (fc).

since the introduction of BN (Ioffe & Szegedy, 2015), it is unclear why weight decay is still effective due to the fact that networks with BN is invariant to the scaling of weights. van Laarhoven (2017) made an interesting argument that weight decay has no regularization effect when combined with normalization, and instead has an influence on the effective learning rate.

In this section, we confirm this argument and show that in first-order optimization methods with BN networks, the role of weight decay is simply picking a higher learning rate, which happens to improve generalization. For simplicity, we first focus on Stochastic Gradient Descent (SGD).

To be precise, weight decay has no regularization effect on the layers followed by BN, but it is not true for the last layer where we don't use BN in practice. As pointed out by van Laarhoven (2017), weight decay doesn't limit the capacity of neural networks for those layers with BN, since it gives the same function by scaling down the weights

$$\mathrm{BN}(\mathbf{x}; \alpha \boldsymbol{\theta}_l) = \mathrm{BN}(\mathbf{x}; \boldsymbol{\theta}_l) \tag{10}$$

where $\boldsymbol{\theta}_l$ is the weight of l-th layer. However, the scale of $\boldsymbol{\theta}_l$ does influence the updates that are performed in first-order methods:

$$\frac{\partial \mathrm{BN}(\mathbf{x}; \alpha \boldsymbol{\theta}_l)}{\partial (\alpha \boldsymbol{\theta}_l)} = \frac{1}{\alpha} \frac{\partial \mathrm{BN}(\mathbf{x}; \boldsymbol{\theta}_l)}{\partial (\boldsymbol{\theta}_l)} \tag{11}$$

Due to its invariance to the scaling of the weights, the key feature of the weights becomes its direction. As shown in Hoffer et al. (2018), the weight direction $\hat{\boldsymbol{\theta}}_l = \boldsymbol{\theta}_l / ||\boldsymbol{\theta}_l||_2$ is updated according to

$$\hat{\boldsymbol{\theta}}_l^{t+1} \leftarrow \hat{\boldsymbol{\theta}}_l^t - \eta ||\boldsymbol{\theta}_l^t||_2^{-2} (\mathbf{I} - \hat{\boldsymbol{\theta}}_l^t \hat{\boldsymbol{\theta}}_l^{t\top}) \nabla_{\boldsymbol{\theta}_l} \mathcal{L}(\hat{\boldsymbol{\theta}}^t) + O(\eta^2) \tag{12}$$

Therefore, the effective learning rate is approximately proportional to $\eta / ||\boldsymbol{\theta}_l||_2^2$. Which means that by decreasing the scale of the weights, weight decay regularization is able to increase the effective learning rate. This is contrary to our intuition that weight decay results in a simple function.

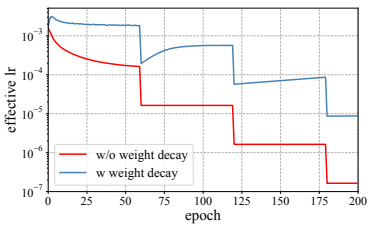


Figure 3: Effective learning rate of the first layer of ResNet32 on CIFAR-100.

We show empirically that weight decay only improves generalization by controlling the norm, and therefore the effective learning rate. Without weight decay, the norm is unbounded, leading to decreasing effective learning rate. Therefore, we conjecture that higher effective learning rate will result in better generalization performance.

In order to verify the above argument, we apply weight decay to different parts of the networks, e.g., the layers with BN, the last layer and all layers and compare them to the baseline in terms of test accuracy as shown in Figure 2.

As expected, if we only apply weight decay to layers with BN, we still observe comparable gain over the baseline as wd-conv and wd-all are nearly identical. On the other hand, if we only apply weight decay to the last layer, the improvement compared to the baseline is marginal and can be ignored, suggesting higher effective learning rate will improve the generalization performance.

To further verify the "effective learning rate" argument, we normalize the weights of the layers with BN at the end of each epoch according to norm statistics from a training with weight decay. Ideally, we are supposed to normalize the weights each iteration. Nevertheless, we found that amortizing the normalization to each epoch doesn't degrade the performance much. As shown in Figure 4, normalizing the layers with BN (wn-conv) causes much of the gap diminished and we can close the gap by further adding weight decay to the last layer (wd-fc+wn-conv).

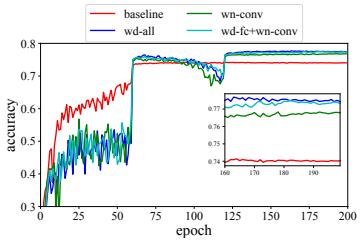


Figure 4: The curves of testing accuracies of ResNet32 on CIFAR-100. To be noted, we use **wd** and **wn** to denote weight decay and weight normalization respectively.

4.2 Mechanism II: Approximate Jacobian Regularization

Since we observed in Section 3 that weight decay regularization is more effective in KFAC when BN is disabled, we now discuss the effect of weight decay in this case by focusing on plain neural networks without BN.

While weight decay is well-understood in SGD to be equivalent to L2 regularizer, its effects in natural gradient have not been explored. A natural question would be *what's the underlying objective we are optimizing with the combination of natural gradient and weight decay?* Below, we answer this question formally:

Proposition 1 (For Natural Gradient, Weight Decay $\equiv$ Curvature Norm Regularization)**.** *In the case of natural gradient descent and base learning rate η, optimizing the loss function $\mathcal{L}$ with weight decay $\eta\beta$ is equivalent to optimizing $\mathcal{L}(\boldsymbol{\theta}) + \frac{\beta}{2}||\boldsymbol{\theta}||^2_{\mathbf{C}}$ without weight decay.*

With weight decay regularization, it is equivalent to adding curvature norm to the original loss function. We note that this interpretation holds only approximately[3], because we don't differentiate through $\mathbf{C}$, which changes as a function of $\boldsymbol{\theta}$. When the curvature matrix is taken to be Gauss-Newton matrix, we get Gauss-Newton norm which is defined as $||\boldsymbol{\theta}||^2_{\mathbf{G}} = \boldsymbol{\theta}^\top \mathbf{G}(\boldsymbol{\theta})\boldsymbol{\theta}$. We find that Gauss-Newton norm on the parameters is equivalent to L2 norm in the output space under some assumptions, which we summarize in the following theorem.

Theorem 1 (Gauss-Newton Norm & KFAC Gauss-Newton Norm)**.** *For a feed-forward neural network of depth L with rectified linear activation and no bias, we observe:*

$$||\boldsymbol{\theta}||^2_{\mathbf{G}} = (L+1)^2 \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right] \tag{13}$$

Furthermore, if we assume the network is linear, we have KFAC Gauss-Newton norm as follows

$$||\boldsymbol{\theta}||^2_{\mathbf{G}_{\text{KFAC}}} = (L+1) \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right] \tag{14}$$

According to Theorem 2, Gauss-Newton norm can be further rewritten as $(L+1)^2 \mathbb{E}\left[\mathbf{x}^\top \mathbf{J}_{\mathbf{x}}^\top \mathbf{J}_{\mathbf{x}} \mathbf{x}\right]$, therefore is equivalent to the squared Frobenius norm of input-output Jacobian norm under the assumptions.

Remark 1 (Approximate Jacobian norm)**.** *In the case that the distribution of $\mathbf{x}$ is spherical Gaussian with unit variance and $f_{\boldsymbol{\theta}}$ is a linear function of $\mathbf{x}$ (no nonlinear activation functions), $||\boldsymbol{\theta}||^2_{\mathbf{G}} = (L+1)^2 ||\mathbf{J}_{\mathbf{x}}||^2_{\text{Frob}}$. Furthermore, if $\mathbf{G}$ is approximated by KFAC, $||\boldsymbol{\theta}||^2_{\mathbf{G}_{\text{KFAC}}} = (L+1)||\mathbf{J}_{\mathbf{x}}||^2_{\text{Frob}}$.*

The intuition behind Remark 1 is that when the network is a linear function, the input-output Jacobian is independent of the input $\mathbf{x}$ and we can take it as constant. In that case, if the distribution of $\mathbf{x}$ is spherical Gaussian, $\mathbb{E}\left[\mathbf{x}^\top \mathbf{J}_{\mathbf{x}}^\top \mathbf{J}_{\mathbf{x}} \mathbf{x}\right] = \text{Trace}(\mathbf{J}_{\mathbf{x}}^\top \mathbf{J}_{\mathbf{x}}) = ||\mathbf{J}_{\mathbf{x}}||^2_{\text{Frob}}$. Moreover, the Frobenius norm of input-output Jacobian has been observed to be consistently coupled with generalization performance (Novak et al., 2018).

However, the connection between Gauss-Newton norm and Jacobian norm is not true for general deep non-linear networks. To test whether they are still correlated in practical neural networks, we train feed-forward networks with different optimizers on both MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky & Hinton, 2009). For MNIST, we use simple fully-connected networks with different depth and width. For CIFAR-10, we adopt VGG family (From VGG11 to VGG19). We highlight that the generalization gap is measured by the gap between training loss and testing loss.

[3] We show in Appendix E that this interpretation holds exactly in the case of Gauss-Newton norm.

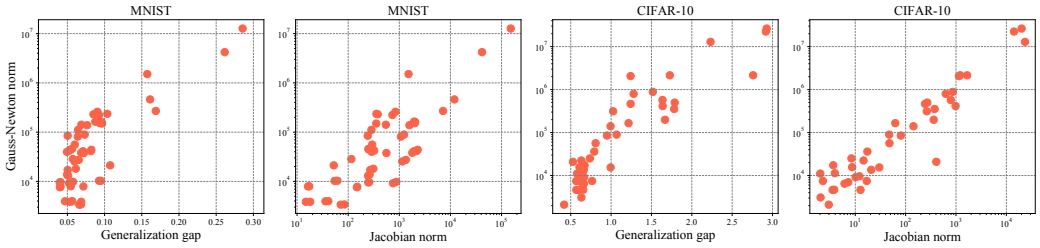


Figure 5: Correlation between Gauss-Newton norm and Jacobian norm in practical deep neural networks. Each point corresponds to a network trained to 100% training accuracy.

Our main result is presented in Figure 5. The left two figures show that Gauss-Newton norm correlates well with generalization performance and Jacobian norm, justifying its use as regularization term. On right side, we report similar results on CIFAR-10 with VGG networks. We conclude that in the settings without BN, weight decay improves generalization of natural KFAC-G by approximately regularizing the Jacobian norm.

Table 2: Squared Frobenius norm of input-output Jacobian matrix. **WD** denotes weight decay.

Optimizer	VGG16		ResNet32	
		WD		WD
SGD	564	142	2765	1074
KFAC-G	498	51.44	2115	64.16

We further compare the Jacobian norm of networks trained with and without weight decay after convergence. As shown in Table 2, weight decay, combined with KFAC-G, is able to reduce the Jacobian norm by order of magnitude. By contrast, the reduction in the case of SGD is much smaller.

Analogously to the case of Gauss-Newton matrix, we can obtain Fisher-Rao norm (Liang et al., 2017) by taking the curvature matrix as Fisher matrix, which is defined as $|\boldsymbol{\theta}||_{\mathbf{F}}^2 = \boldsymbol{\theta}^\top \mathbf{F} \boldsymbol{\theta}$. It has been shown that Fisher-Rao norm serves as an umbrella for the previously considered norm-based capacity measures (Neyshabur et al., 2015) and is a good complexity measure of neural networks. Therefore, weight decay in both KFAC-F and KFAC-G serves as a complexity regularizer, which can be used to account for the significant improvements on the generalization performance.

4.3 Mechanism III: Smaller Effective Damping Parameter

As we show in Section 4.2, weight decay approximately regularizes Jacobian norm of the networks. However, this argument may not hold once BN is applied. Similar to the first mechanism, we expect the effect of weight decay on intermediate layers is completely canceled out by BN, so weight decay should only affect the last layer of the network. Particularly, for first-order methods, the scaling of the weights does change the updates that are performed by adjusting the effective learning rate. However, KFAC update, preconditioned by approximated curvature matrix, is invariant to affine reparameterization (Luk & Grosse, 2018) and therefore not affected by the scaling of the weights (van Laarhoven, 2017). The key observation here is that, when with BN the curvature matrix, e.g., Fisher matrix or Gauss-Newton matrix, has the following property

$$\mathbf{C}(\boldsymbol{\theta}_l) = \mathbf{C}(||\boldsymbol{\theta}_l||_2 \hat{\boldsymbol{\theta}}_l) = \frac{1}{||\boldsymbol{\theta}_l||_2^2} \mathbf{C}(\hat{\boldsymbol{\theta}}_l) \tag{15}$$

which compensates the term of $\frac{\eta}{||\boldsymbol{\theta}_l||_2^2}$ in effective learning rate after inversion.

In this sense, weight decay regularization has no impact on the layers with BN. Thus, we may expect that the reason weight decay does not significantly improve generalization when BN is applied (see CIFAR-10 results in Table 1) is because the Jacobian regularization is only implicitly applied to the last layer.

This raises the question of *whether it is still necessary to reguarlize the layers with BN in KFAC*. To test this, we repeat the experiments in Figure 2 but with KFAC instead. The results are summarized in Figure 6. Contrary to our expectations, applying weight decay to layers with BN can still improve the performance significantly in KFAC-F. Overall, KFAC-F performs very similarly to first-order methods while the gain of weight decay in KFAC-G is mostly from regularizing the last layer.

To understand this phenomenon, we notice that previous analysis of natural gradient does not take into account the damping term, which is commonly used in second-order optimization to dampen the updates by adding a multiple of the identity matrix to the curvature before inversion. Analogous to

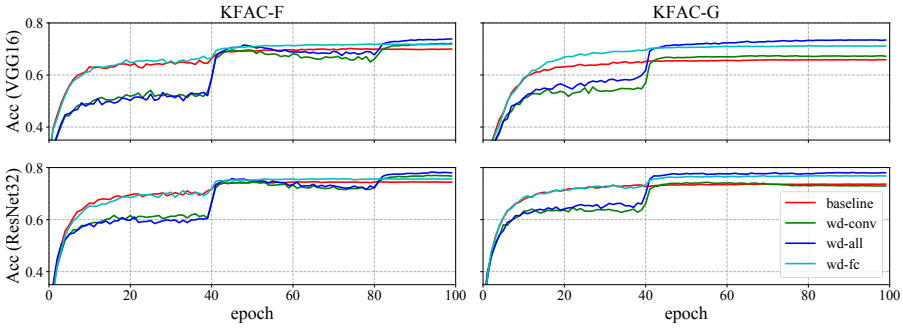


Figure 6: Test accuracy as a function of training epoch for KFAC on CIFAR-100 with ResNet32 model and different weight decay regularization schemes. baseline is the model without weight decay regularization; wd-conv is the model with weight decay applied to all convolutional layers; wd-all is the model with weight decay applied to all layers; wd-fc is the model with weight decay applied to last layer (fc).

equation 12, we can derive the update of weight direction by adding the fixed damping term to the curvature matrix before computing the inverse (see Appendix D for proof)

$$\hat{\boldsymbol{\theta}}_l^{t+1} \leftarrow \hat{\boldsymbol{\theta}}_l^t - \eta(\mathbf{I} - \hat{\boldsymbol{\theta}}_l^t \hat{\boldsymbol{\theta}}_l^{t\top})(\mathbf{C}(\hat{\boldsymbol{\theta}}_l^t) + ||\boldsymbol{\theta}_l^t||_2^2 \lambda \mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}_l} \mathcal{L}(\hat{\boldsymbol{\theta}}^t) + O(\eta^2) \tag{16}$$

From equation 16, we can see that natural gradient updates are not exactly invariant to the scaling of the weights. Particularly, KFAC-F and KFAC-G only differ in the choice of curvature matrix. We therefore compare the trends of the scale of the Fisher matrix and Gauss-Newton matrix by monitoring the trace norm of the first layer (see Figure 7). Surprisingly, the scale of Fisher matrix $\mathbf{F}(\hat{\boldsymbol{\theta}}_l)$ decreases as the training progresses while the scale of Gauss-Newton matrix is quite stable.

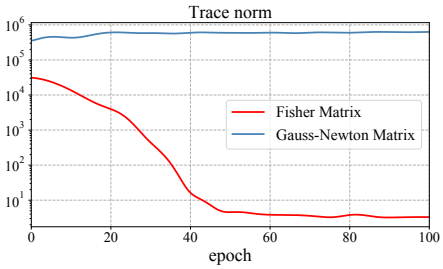


Figure 7: Trace norm of Fisher matrix and Gauss-Newton matrix of the first layer (Normalized) of ResNet32. The model is trained on CIFAR-10 with KFAC-F and BN is used. Note that the curves are smoothened for visual clarity.

In classification task with cross-entropy loss, the Fisher matrix is equivalent to generalized Gauss-newton matrix (Martens, 2014). Therefore, the only difference between Gauss-Newton matrix and the Fisher matrix is the Hessian of loss $\mathbf{H}_\ell$ (see Section 2), which is given by $\text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$. It is easy to see that the Hessian of cross-entropy loss would goes to zero matrix as the prediction $\mathbf{p}$ collapses to one class, which is often the case in neural network training. That explains the different trends of the scale of the Fisher and the Gauss-Newton well.

We notice that the equation 16 becomes exactly equation 12 as the Fisher matrix becomes smaller and smaller and therefore the damping term dominates, which explains why KFAC-F behaves like first-order methods. Although the scale of Gauss-Newton matrix is quite stable, the damping term will increase as the weight norm grows. That's why regularizing convolutional layers does make a small difference. Therefore, it is still necessary to apply weight decay regularization to control "effective damping parameter", which is proportional to $||\boldsymbol{\theta}_l||_2^2 \lambda \mathbf{I}$.

To summarize, weight decay helps control a small effective damping parameter in KFAC with BN enabled, enabling KFAC to retain its second-order properties, and correspondingly improves generalization.

5 Conclusion

Weight decay has been exclusively used in deep learning community, but the reasons for its regularization effect are poorly understood, and recent results have cast doubt on the traditional interpretation in terms of L2 regularization. Through extensive systematic experiments, we show weight decay consistently outperforms L2 regularization in cases where they differ. We empirically investigate weight decay for three optimization algorithms (SGD, Adam, and KFAC) and a variety of network architectures. We identify three distinct mechanisms by which weight decay improves generalization. Our results provide insight into how to improve the regularization of neural networks.

REFERENCES

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. 2016.

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 *Neural Networks, 1996., IEEE International Conference on*, volume 1, pp. 241–246. IEEE, 1996.

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.

Tom Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pp. 1731–1741, 2017.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. *arXiv preprint arXiv:1803.01814*, 2018.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. 2017.

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

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

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

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

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.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. *arXiv preprint arXiv:1711.01530*, 2017.

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

Kevin Luk and Roger Grosse. A coordinate-free construction of scalable natural gradient. *arXiv preprint arXiv:1808.10340*, 2018.

James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pp. 1376–1401, 2015.

Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

A Experiments Details

Throughout the paper, we perform experiments on image classification with three different datasets, MNIST (LeCun et al., 1998), CIFAR-10 and CIFAR-100 (Krizhevsky & Hinton, 2009). For MNIST, we use simple fully-connected networks with different depth and width. For CIFAR-10 and CIFAR-100, we use VGG16 (Simonyan & Zisserman, 2014) and ResNet32 (He et al., 2016). To make the network more flexible, we widen all convolutional layers in ResNet32 by a factor of 4, according to Zagoruyko & Komodakis (2016).

We investigate three different optimization methods, including Stochastic Gradient Descent (SGD), Adam (Kingma & Ba, 2014) and KFAC (Martens & Grosse, 2015). In KFAC, two different curvature matrices are studied, including Fisher matrix and Gauss-Newton matrix.

In default, batch size 128 is used unless stated otherwise. In SGD and Adam, we train the networks with a budge of 200 epochs and decay the learning rate by a factor of 10 every 60 epochs for batch sizes of 128 and 640, and every 80 epochs for the batch size of 2K. Whereas we train the networks only with 100 epochs and decay the learning rate every 40 epochs in KFAC. Additionally, the curvature matrix is updated by running average with re-estimation every 10 iterations and the inverse operator is amortized to 100 iterations. For KFAC, we use fixed damping term $1e^{-3}$ unless state otherwise. For each algorithm, best hyperparameters (learning rate and regularization factor) were selected using grid search on held-out 5k validation set. For the large batch setting, we adopt the same strategies in Hoffer et al. (2017) for adjusting the search range of hyperparameters. Then, we retrain the model with both training data and validation data.

B Gradient Structure in Neural Networks

Lemma 1 (Gradinet structure)**.** *For a feed-forward rectified neural network without bias, one has the following property:*

$$\begin{aligned} f_{\boldsymbol{\theta}}(\mathbf{x}) &= \nabla_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x})^{\top}\mathbf{x} = \mathbf{J}_{\mathbf{x}}\mathbf{x} \\ &= \frac{1}{L+1}\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x})^{\top}\boldsymbol{\theta} = \frac{1}{L+1}\mathbf{J}_{\boldsymbol{\theta}}\boldsymbol{\theta} \end{aligned} \tag{17}$$

The key observation of Lemma 1 is that rectified neural networks are piecewise linear up to the output $f_{\boldsymbol{\theta}}(\mathbf{x})$. And ReLU activation function satisfies the property $\sigma(\mathbf{z}) = \sigma'(\mathbf{z})\mathbf{z}$.

Proof. For convenience, we introduce some notations here. Let $\mathbf{z}_{L+1}$ denotes output logits $f_{\boldsymbol{\theta}}(\mathbf{x})$, $\mathbf{z}_l$ the output l-th layer. Similarly, we define $\mathbf{a}_l = \sigma(\mathbf{z}_l)$ and $\mathbf{a}_0 = \mathbf{x}$. By definition, it is easy to see that

$$\begin{aligned}\mathbf{z}_{l+1} = \mathbf{W}_l \mathbf{a}_l &= \frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{a}_l^\top} \mathbf{a}_l \\ &= \frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{a}_l^\top} \frac{\partial \mathbf{a}_l}{\partial \mathbf{z}_l^\top} \mathbf{z}_l \\ &= \frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{z}_l^\top} \mathbf{z}_l\end{aligned}$$

By induction, we conclude that $f_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x})^\top \mathbf{x}$.

On the other side, we have

$$\mathbf{z}_{l+1} = \mathbf{W}_l \mathbf{a}_l = \sum_{i,j} \frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{W}_l^{i,j}} \mathbf{W}_l^{i,j}$$

According to equation B, $\mathbf{z}_{L+1} = \frac{\partial \mathbf{z}_{L+1}}{\partial \mathbf{z}_{l+1}} \mathbf{z}_{l+1}^\top$, therefore we get

$$\mathbf{z}_{L+1} = \sum_{i,j} \frac{\partial \mathbf{z}_{L+1}}{\partial \mathbf{W}_l^{i,j}} \mathbf{W}_l^{i,j}$$

Summing over all the layers, we conclude the following equation eventually:

$$(L+1) f_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_l \sum_{i,j} \frac{\partial \mathbf{z}_{L+1}}{\partial \mathbf{W}_l^{i,j}} \mathbf{W}_l^{i,j} = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x})^\top \boldsymbol{\theta}$$

□

C PROOF OF THEMREM 2

Theorem 2 (Gauss-Newton Norm & KFAC Gauss-Newton Norm)**.** *For a feed-forward neural network of depth L with rectified linear activation and no bias, we observe:*

$$||\boldsymbol{\theta}||_{\mathbf{G}}^2 = (L+1)^2 \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right] \tag{18}$$

Furthermore, if we assume the network is linear, we have KFAC Gauss-Newton norm as follows

$$||\boldsymbol{\theta}||_{\mathbf{G}_{\text{KFAC}}}^2 = (L+1) \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right] \tag{19}$$

Proof. We first prove the equaility $||\boldsymbol{\theta}||_{\mathbf{G}}^2 = (L+1)^2 \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right]$. Using the definition of the Gauss-Newton norm, we have

$$\begin{aligned}||\boldsymbol{\theta}||_{\mathbf{G}}^2 &= \mathbb{E}\left[\boldsymbol{\theta}^\top \mathbf{J}_{\boldsymbol{\theta}}^\top \mathbf{J}_{\boldsymbol{\theta}} \boldsymbol{\theta}\right] \\ &= \mathbb{E}\left[\langle \mathbf{J}_{\boldsymbol{\theta}} \boldsymbol{\theta}, \mathbf{J}_{\boldsymbol{\theta}} \boldsymbol{\theta}\rangle\right]\end{aligned}$$

By Lemma 1,

$$\mathbf{J}_{\boldsymbol{\theta}} \boldsymbol{\theta} = (L+1) f_{\boldsymbol{\theta}}(\mathbf{x}) = (L+1) \mathbf{J}_{\mathbf{x}} \mathbf{x}$$

Combining above equalities, we arrive at the conclusion.

For second part $||\boldsymbol{\theta}||_{\mathbf{G}_{\text{KFAC}}}^2 = (L+1) \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right]$, we note that kronecker-product is exact under the condition that the network is linear, which means $\mathbf{G}_{\text{KFAC}}$ is the diagonal block version of Gauss-Newton matrix $\mathbf{G}$. Therefore

$$||\boldsymbol{\theta}||_{\mathbf{G}_{\text{KFAC}}}^2 = \sum_l \mathbb{E}\left[\boldsymbol{\theta}_l^\top \mathbf{J}_{\boldsymbol{\theta}_l}^\top \mathbf{J}_{\boldsymbol{\theta}_l} \boldsymbol{\theta}_l\right]$$

According to Lemma 1, we have $\mathbb{E}\left[\boldsymbol{\theta}_l^\top \mathbf{J}_{\boldsymbol{\theta}_l}^\top \mathbf{J}_{\boldsymbol{\theta}_l} \boldsymbol{\theta}_l\right] = \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right]$, therefore we conclude that

$$||\boldsymbol{\theta}||_{\mathbf{G}_{\text{KFAC}}}^2 = (L+1) \mathbb{E}\left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x})\rangle\right]$$

□

D DERIVATION OF EQUATION 16

Claim. During training, the weight direction $\hat{\boldsymbol{\theta}}_l^t = \boldsymbol{\theta}_l^t / ||\boldsymbol{\theta}_l^t||_2$ is updated according to

$$\hat{\boldsymbol{\theta}}_{t+1} \leftarrow \hat{\boldsymbol{\theta}}_t - \eta(\mathbf{I} - \hat{\boldsymbol{\theta}}_t \hat{\boldsymbol{\theta}}_t^\top)(\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + ||\boldsymbol{\theta}_t||_2^2 \lambda \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t) + O(\eta^2)$$

Proof. Natural gradient update is given by

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta(\mathbf{C}(\boldsymbol{\theta}_t) + \lambda \mathbf{I})^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

Denote $\rho_t = ||\boldsymbol{\theta}_t||_2$. Then we have

$$\rho_{t+1}^2 = \rho_t^2 - 2\eta \rho_t^2 \hat{\boldsymbol{\theta}}_t^\top (\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t) + \eta^2 \rho_t^2 ||(\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t)||_2^2$$

and therefore

$$\begin{aligned}
\rho_{t+1} &= \rho_t \sqrt{1 - 2\eta \hat{\boldsymbol{\theta}}_t^\top (\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t) + \eta^2 ||(\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t)||_2^2} \\
&= \rho_t (1 - \eta \hat{\boldsymbol{\theta}}_t^\top (\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t)) + O(\eta^2)
\end{aligned}$$

Additionally, we can rewrite the natural gradient update as follows

$$\rho_{t+1} \hat{\boldsymbol{\theta}}_{t+1} = \rho_t \hat{\boldsymbol{\theta}}_t - \eta \rho_t (\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t)$$

And therefore,

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{t+1} &= \frac{\rho_t}{\rho_{t+1}} \left(\hat{\boldsymbol{\theta}}_t - \eta (\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t) \right) \\
&= \left(1 + \eta \hat{\boldsymbol{\theta}}_t^\top (\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t) \right) \left(\hat{\boldsymbol{\theta}}_t - \eta (\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + \lambda \rho_t^2 \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t) \right) + O(\eta^2) \\
&= \hat{\boldsymbol{\theta}}_t - \eta (\mathbf{I} - \hat{\boldsymbol{\theta}}_t \hat{\boldsymbol{\theta}}_t^\top)(\mathbf{C}(\hat{\boldsymbol{\theta}}_t) + ||\boldsymbol{\theta}_t||_2^2 \lambda \mathbf{I})^{-1} \nabla \mathcal{L}(\hat{\boldsymbol{\theta}}_t) + O(\eta^2)
\end{aligned}$$

□

E THE GRADIENT OF GAUSS-NEWTON NORM

For Gauss-Newton norm $||\boldsymbol{\theta}||_{\mathbf{G}}^2 = (L+1)^2 \mathbb{E}_{\mathbf{x}} \left[\langle f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{x}) \rangle \right]$, its gradient has the following form:

$$\frac{\partial ||\boldsymbol{\theta}||_{\mathbf{G}}^2}{\partial \boldsymbol{\theta}} = 2(L+1)^2 \mathbb{E} \left[\mathbf{J}_{\boldsymbol{\theta}}^\top f_{\boldsymbol{\theta}}(\mathbf{x}) \right] \tag{20}$$

According to Lemma 1, we have $f_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{L+1} \mathbf{J}_{\boldsymbol{\theta}} \boldsymbol{\theta}$, therefore we can rewrite equation 20

$$\begin{aligned}
\frac{\partial ||\boldsymbol{\theta}||_{\mathbf{G}}^2}{\partial \boldsymbol{\theta}} &= 2(L+1) \mathbb{E} \left[\mathbf{J}_{\boldsymbol{\theta}}^\top \mathbf{J}_{\boldsymbol{\theta}} \boldsymbol{\theta} \right] \\
&= 2(L+1) \mathbf{G} \boldsymbol{\theta}
\end{aligned} \tag{21}$$

Surprisingly, the resulting gradient has the same form as the case where we take Gauss-Newton matrix as a constant of $\boldsymbol{\theta}$ up to a constant $(L+1)$.

F PSEUDO CODE OF KFAC WITH WEIGHT DECAY REGULARIZATION

G ADDITIONAL RESULTS

G.1 LARGE-BATCH TRAINING

It has been shown that KFAC scales very favorably to larger mini-batches compared to SGD, enjoying a nearly linear relationship between mini-batch size and per-iteration progress for medium-to-large sized mini-batches (Martens & Grosse, 2015; Ba et al., 2016). However, Keskar et al. (2016) showed that large-batch methods converge to sharp minima and generalize worse. In this subsection, we

Algorithm 1 K-FAC with L2 regularization and K-FAC with weight decay. Subscript l denotes layers, $\mathbf{w}_l = \mathrm{vec}(\mathbf{W}_l)$. We assume zero momentum for simplicity.

Require: η: stepsize
Require: β: weight decay
Require: stats and inverse update intervals T_{stats} and T_{inv}
  $k \leftarrow 0$ and initialize $\{\mathbf{W}_l\}_{l=1}^{L}, \{\mathbf{S}_l\}_{l=1}^{L}, \{\mathbf{A}_l\}_{l=1}^{L}$
  while stopping criterion not met **do**
    $k \leftarrow k + 1$
    if $k \equiv 0 \pmod{T_{\mathrm{stats}}}$ **then**
      Update the factors $\{\mathbf{S}_l\}_{l=1}^{L}, \{\mathbf{A}_l\}_{l=0}^{L-1}$ with moving average
    end if
    if $k \equiv 0 \pmod{T_{\mathrm{inv}}}$ **then**
      Calculate the inverses $\{[\mathbf{S}_l]^{-1}\}_{l=1}^{L}, \{[\mathbf{A}_l]^{-1}\}_{l=0}^{L-1}$
    end if
    $\mathbf{V}_l = \nabla_{\mathbf{W}_l} \log p(y|\mathbf{x}, \mathbf{w}) + \beta \cdot \mathbf{W}_l$
    $\mathbf{W}_l \leftarrow \mathbf{W}_l - \left(\eta [\mathbf{A}_l]^{-1} \mathbf{V}_l [\mathbf{S}_l]^{-1} + \beta \cdot \mathbf{W}_l\right)$
  end while

measure the generalization performance of KFAC with large batch training and analyze the effect of weight decay.

In Table 3, we compare KFAC with SGD using different batch sizes. In particular, we interpolate between small-batch (BS128) and large-batch (BS2000). We can see that in accordance with previous works (Keskar et al., 2016; Hoffer et al., 2017) the move from a small-batch to a large-batch indeed incurs a substantial generalization gap. However, adding weight decay regularization to KFAC almost close the gap on CIFAR-10 and cause much of the gap diminish on CIFAR-100. Surprisingly, the generalization gap of SGD also disappears with well-tuned weight decay regularization. Besides, we observe that the training loss cannot decrease to zero if weight decay is not used, indicating weight decay may also speed up the training.

Table 3: Classification results with different batch sizes. **WD** denotes weight decay regularization. We tune weight decay factor and learning rate using held-out validation set.

Dataset	Network	Method	BS128		BS640		BS2000	
				WD		WD		WD
CIFAR10	VGG16	SGD	91.71	93.39	90.46	93.09	88.50	92.24
		KFAC-F	93.12	93.90	92.93	93.55	92.17	93.31
		KFAC-G	93.41	93.67	92.98	93.74	90.78	93.46
CIFAR10	ResNet32	SGD	92.95	95.14	91.68	94.45	89.70	94.68
		KFAC-F	93.80	95.35	92.30	94.79	91.15	94.43
		KFAC-G	93.44	95.04	91.80	94.73	90.02	94.85
CIFAR100	ResNet32	SGD	74.03	77.73	71.74	76.67	65.38	76.87
		KFAC-F	75.42	78.01	73.54	77.34	71.64	77.13
		KFAC-G	73.70	78.02	71.13	77.40	65.41	76.93

G.2 The curves of test accuracies

G.3 Optimization performance of different optimizers

While this paper mostly focus on generalization, we also report the convergence speed of different optimizers in deep neural networks; we report both per-epoch performance and wall-clock time performance.

We consider the task of image classification on CIFAR-10 (Krizhevsky & Hinton, 2009) dataset. The models we use consist of VGG16 (Simonyan & Zisserman, 2014) and ResNet32 (He et al., 2016). We compare our KFAC-G, KFAC-F with SGD, Adam (Kingma & Ba, 2014).

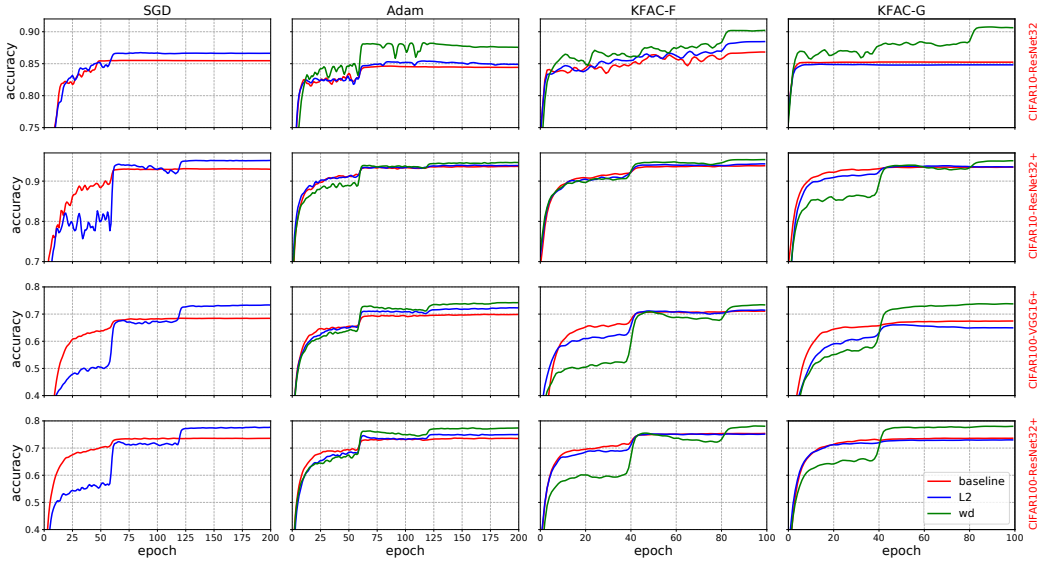
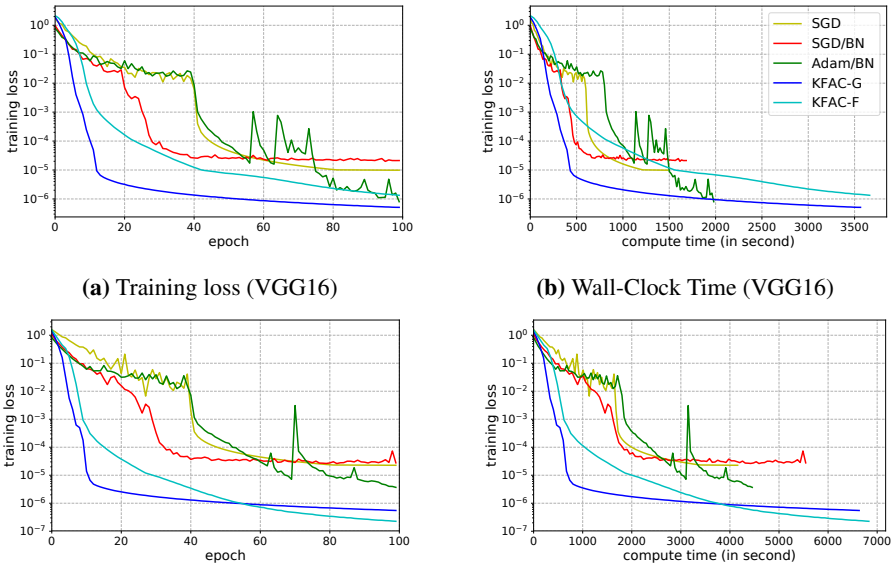


Figure 8: Test accuracy as a function of training epoch. We plot baseline vs L2 regularization vs weight decay regularization on CIFAR-10 and CIFAR-100 datasets. The '+' denotes with BN and data augmentation. Note that training accuracies of all the models are 100% in the end of the training. We smooth all the curves for visual clarity.

We experiment with constant learning for KFAC-G and KFAC-F. For SGD and Adam, we set batch size as 128. For KFAC, we use batch size of 640, as suggested by Martens & Grosse (2015).

In Figure 9, we report the training curves of different algorithms. Figure 9a show that KFAC-G yields better optimization than other baselines in training loss per epoch. We highlight that the training loss decreases to 1e-4 within 10 epochs with KFAC-G. Although KFAC based algorithms take more time for each epoch, Figure 9b still shows wall-clock time improvements over the baselines.

In Figure 9c and 9d, we report similar results on the ResNet32. Note that we make the network wider with a widening factor of 4 according to Zagoruyko & Komodakis (2016). KFAC-G outperforms both KFAC-F and other baselines in term of optimization per epoch, and compute time.



(a) Training loss (VGG16)

(b) Wall-Clock Time (VGG16)

(c) Training loss (ResNet32)

(d) Wall-Clock Time (ResNet32)

Figure 9: CIFAR-10 image classification task.