Inefficiency of K-FAC for Large Batch Size Training

Linjian Ma,[1*] Gabe Montague,[1*] Jiayu Ye,[1*]
Zhewei Yao,[1] Amir Gholami,[1] Kurt Keutzer,[1] Michael W. Mahoney[1]
[1]University of California at Berkeley, Berkeley, USA
{linjian, gabe_montague, yejiayu, zheweiy, amirgh, keutzer, mahoneymw}@berkeley.edu

Abstract

There have been several recent work claiming record times for ImageNet training. This is achieved by using large batch sizes during training to leverage parallel resources to produce faster wall-clock training times per training epoch. However, often these solutions require massive hyper-parameter tuning, which is an important cost that is often ignored. In this work, we perform an extensive analysis of large batch size training for two popular methods that is Stochastic Gradient Descent (SGD) as well as Kronecker-Factored Approximate Curvature (K-FAC) method. We evaluate the performance of these methods in terms of both wall-clock time and aggregate computational cost, and study the hyper-parameter sensitivity by performing more than 512 experiments per batch size for each of these methods. We perform experiments on multiple different models on two datasets of CIFAR-10 and SVHN. The results show that beyond a critical batch size both K-FAC and SGD significantly deviate from ideal strong scaling behaviour, and that despite common belief K-FAC does not exhibit improved large-batch scalability behavior, as compared to SGD.

1 Introduction

As the boundaries of parallelism are pushed by modern hardware and distributed systems, researchers are increasingly turning their attention toward leveraging these advances for faster training of deep neural networks (DNNs). When using the prevailing Stochastic Gradient Descent (SGD) method, a batch of training data is split across computational processing units, which together compute a stochastic gradient used to update the parameters of the DNN.

To allow for efficient parallel scalability to a large number of processors, one would like to use a large batch of training data to compute the stochastic gradient estimate (Gholami et al. 2018). Using a larger batch size allows scaling of the training to more GPUs, thus reducing training time for a fixed number of epochs as opposed to small batch training. However, large batch training of DNNs often results in sub-optimal generalization performance as compared to training with small batch size (Keskar et al. 2016; Yao et al. 2018b). However, multiple different groups have provided solutions for large batch size, reducing training time of ImageNet from 720 hours on a Titan X (Iandola et al. 2016) down to minutes/seconds (Goyal et al. 2017a; Yao et al. 2018a; Devarakonda, Naumov, and Garland 2017; You, Gitman, and Ginsburg 2017; You et al. 2017; Jia et al. 2018; Yamazaki et al. 2019; Osawa et al. 2018; Ginsburg, Gitman, and You 2018; Mu et al. 2018). An important practical consideration in methods is the sensitivity to hyperparameters. While it is possible to train ImageNet in minutes, but often the cost of hyper-parameter tuning and some times ad-hoc rules has made large batch training not a feasible method in practice. Generally, we find that this sensitivity is quite strong, and the required tuning process is expensive in terms of both analyst time and in-search training time. If performing large batch training requires significant hyperparameter tuning for each batch-size, then one would not achieve any effective speed up in total training time (i.e., hyperparameter tuning time plus final training time).

Using a large batch size changes the dynamics of the training. It has been demonstrated both theoretically (Ma, Bassily, and Belkin 2017; Martin and Mahoney 2018; 2019) and empirically (Yao et al. 2018b; Golmant et al. 2018; McCandlish et al. 2018; Shallue et al. 2018; Keskar et al. 2016) that, in many cases, training with large-batch SGD comes with significant drawbacks. This includes degraded testing performance, worse implicit regularization, and diminishing returns in terms of training loss reduction. Among other things, there exists a critical batch size beyond which these effects are most acute. For practitioners operating on a data-driven computational budget, large batch size comes with the additional inconvenience of increased sensitivity to hyperparameters and thus increased tuning time and cost (Shallue et al. 2018).

There has been recent efforts in using approximate second-order optimization method known as Kronecker-Factored Approximate Curvature (K-FAC) (Martens and Grosse 2015) for large batch training. K-FAC views the parameter space as a manifold of distribution space, in which distance between parameter vectors is measured by a variant of the Kullback–Leibler divergence between their corresponding distributions. In certain circumstances, K-FAC has

[*]Equal contribution. Authors ordered alphabetically.

been demonstrated to attain comparable effectiveness with large batch size as small batch SGD (Osawa et al. 2018), but the cost for hyper-parameter tuning and often the need for performing many more iterations with K-FAC has not been considered.

In this work, we investigate these issues, and perform an extensive study of large batch training with both K-FAC and SGD. In particular, we focus on the following two questions regarding large batch training:

- What is the scalability behavior of K-FAC and SGD, and how does it compare with that of small batch SGD?
- How does increasing batch size affect the hyperparameter sensitivity?

To answer these questions, we conduct a comprehensive investigation in the context of image classification on CIFAR-10 (Krizhevsky and Hinton 2009) and SVHN (Netzer et al. 2011). We investigate the performance of CIFAR-10 with Residual Networks (ResNet20 and ResNet32) classifier (He et al. 2016), and we investigate SVHN with an AlexNet classifier (Krizhevsky, Sutskever, and Hinton 2012). We investigate the problem of large-batch diminishing returns by measuring iteration speedup and comparing it to an ideal scaling scenario. Our key observations are as follows:

- **Performance.** Even with extensive hyperparameter tuning, K-FAC has comparable, but not superior, train/test performance to SGD (Fig. 1).
- **Speedup.** Both K-FAC and SGD exhibit diminishing returns with the increase of batch size. Increasing batch size for K-FAC yields lower, i.e., less prominent, speedup, as compared with SGD, when measured in terms of iterations, even when ignoring the important cost of matrix inversion in K-FAC (Fig. 2).
- **Hyperparameter Sensitivity.** Hyperparameter sensitivity depends on both batch size and epochs/iterations. For fixed epochs, i.e., running the same number of epochs, larger batch sizes result in greater hyperparameter sensitivity and smaller regions of hyperparameter space which result in "good convergence". For fixed iterations, i.e., running the same number of iterations, larger batch sizes result in less sensitivity and larger regions of hyperparameter space which result in "good convergence" (Fig. 3, Fig. 4).

We start with mathematical background and related work in Section 2, followed by a description of our experimental setup in Section 3. Our empirical results demonstrating the inefficiencies of both K-FAC and SGD with large batch sizes appear in Section 4. Our conclusions are in Section 5.

2 Background and Related Work

For a supervised learning framework, the goal is to minimize a loss function expressed as

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(x_i, y_i, \theta),$$

where $\theta \in \mathbb{R}^d$ is the vector of model parameters, and $l(x, y, \theta)$ is the loss for a datum $(x, y) \in (X, Y)$. Here, X is the input, Y is the corresponding label, and $N = |X|$ is the cardinality of the training set. SGD is typically used to optimize the loss by taking steps of the form:

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{|B|}\sum_{(x,y)\in B} \nabla_\theta l(x, y, \theta_t),$$

where B is a mini-batch of examples drawn randomly from $X \times Y$, and η_t is the learning rate at iteration t.

2.1 Kronecker-Factored Approximate Curvature

As opposed to SGD, which treats model parameter space as Euclidean, natural gradient descent methods (Amari 1998) for DNN optimization operate in the space of distributions defined by the model, in which the parameter distance between two vectors is defined using the KL-divergence between the two corresponding distributions. Denoting D_{KL} as our vector norm in this space, it can be shown that $D_{KL}(\Delta\theta) \approx \frac{1}{2}\Delta\theta^\top F \Delta\theta$, where F is the Fisher Information Matrix (FIM) defined as:

$$F = \mathbb{E}[\nabla_\theta \log p(y|x,\theta) \nabla_\theta \log p(y|x,\theta)^\top],$$

where the expectation is taken over both the model's training data space and target variable space (Martens and Grosse 2015). The update rule for natural gradient descent then becomes $\theta_{t+1} = \theta_t - \eta_t F^{-1} \nabla_\theta \ell(\theta_t)$. As noted by (Osawa et al. 2018) and others, the FIM is often poorly-conditioned for DNNs, leading to unstable training. To counter this effect, a damping term is often added (i.e., the FIM is *preconditioned*). Using the preconditioned FIM, the update rule then becomes:

$$\theta_{t+1} = \theta_t - \eta_t (F + \lambda I)^{-1} \nabla_\theta \ell(\theta_t), \tag{1}$$

with λ denoting a positive damping parameter. Due to the computational intractability of the true Fisher matrix, natural gradient methods typically rely on approximations to F. For example, (Martens and Grosse 2015) proposes an approximation, the K-FAC method, exploiting the assumption that (i) F is largely block-diagonal[1] and (ii) across the training distribution, the products of unit activations and products of unit output derivatives are statistically independent. While these assumptions are inexact, their accuracy has been empirically verified by the authors in several cases. The approximation can be written as:

$$F_i = \mathbb{E}[A_{i-1}A_{i-1}^\top \otimes G_i G_i^\top] \approx \mathbb{E}[A_{i-1}A_{i-1}^\top] \otimes \mathbb{E}[G_i G_i^\top],$$

where F_i represents the Fisher matrix of i-th layer, G_i is the gradient of the loss with respect to the i-th layer output before non-linear activation function, and A_{i-1} is the activation output of the previous layer. Note that this is the approximation form we use in our implementation.

[1] Although the K-FAC authors propose an alternative tridiagonal approximation that eases the strength of this assumption, we consider their block diagonal approximation of the Fisher, due to its demonstrated performance.

2.2 Difficulties of Large Batch Training

The problems of large batch training under SGD have been studied in detail through both analytical and empirical studies. (Ma, Bassily, and Belkin 2017) proves that for convex cases, increasing batch size by a factor f yields a no worse than a factor f speedup in the number of SGD iterations, so long as batch size is below a critical point. The batch sizes falling below this critical point are referred to collectively as the *linear scaling regime*. (Golmant et al. 2018) empirically investigates this in the context of non-convex training of DNNs for a variety of training workloads; and it finds evidence of a similar critical batch size for the non-convex case, before which f-fold increases of batch size yield f-fold reductions in total iterations needed to converge, and after which diminishing returns are observed, eventually leading to stagnation and no further benefit.

Subsequent to (Golmant et al. 2018), (Shallue et al. 2018; McCandlish et al. 2018) obtain broadly similar conclusions with more detailed studies. In particular, (McCandlish et al. 2018) goes further to predict the critical batch size to the nearest order of magnitude, demonstrating that critical batch size can be predicted from the *gradient noise scale*, representing the noise-to-signal ratio of the stochastic estimation of the gradient. The authors further find that gradient noise scale increases during the course of training. This principle motivates the success of techniques as in (Smith et al. 2017; Devarakonda, Naumov, and Garland 2017; Yao et al. 2018a), in which batch size is adaptively increased during training.

Apart from increasing batch size during training, effort has been undertaken to increase critical batch size and linear scaling throughout the entire training process. (Goyal et al. 2017b) attempts to improve SGD scalability by tuning hyperparameters more carefully using a linear batch-size to learning-rate relationship. While this proves effective for the authors' training setup, (Golmant et al. 2018) demonstrates that for a wide variety of other training workloads a linear scaling rule is ineffective to counter inefficiencies of large batch. Recent work has applied K-FAC to large-batch training settings, as in training ResNet50 on ImageNet reducing training time down to 10 minutes on 1024 V100 GPU machines of ABCI supercomputer (Osawa et al. 2018). This work actually observed that K-FAC requires more iterations to reach the same accuracy as small batch SGD.

3 Experimental Setup

We investigate the performances of both K-FAC and SGD on CIFAR-10 with ResNet20 and ResNet32, and on SVHN with AlexNet. To be comparable with state-of-art results for K-FAC (Zhang et al. 2019), we apply batch normalization to the models along with standard data augmentation during the training process. We further regularize with a weight decay parameter of 5×10^{-4}. We perform extensive hyperparameter tuning individually for each batch size ranging from 128 to 16,384.

3.1 Learning Rate Schedule

For experiments on CIFAR-10, we decay the learning rate twice by a factor of ten for each run over the course of training. These two learning rate decays separate the training process into three *stages*. Because training extends to a greater number of epochs for large batches under the adjusted epoch budget, for large batch runs we allow a proportionally greater number of epochs to pass before learning rate decay. For each run we therefore decay the learning rate at 40% and 80% of the total epochs[2]. We refer to this decay scheme as a *scaled learning rate schedule*. Similarly, for SVHN, we also choose the *scaled learning rate schedule* and decay the learning rate by a factor of 5 at 50% of the total epochs.

It has been shown that large batch size training often cannot reach the same accuracy as small batch for fixed number of epochs (Keskar et al. 2016; Yao et al. 2018b). Our initial experiments also found the same conclusion. Recent work has shown that large batch requires more iterations to converge to the same accuracy as compared to small batch size (Hoffer, Hubara, and Soudry 2017). To consider this, we allow larger batches to perform more iterations for an *adjusted epoch budget*, in which the epoch limit of training is extended proportionally to the log of the batch size. Specifically, we use the rule: for CIFAR-10, number of training epochs equals $(\log_2(\text{batch size}/128)+1)\times 100$; for SVHN, it equals $(\log_2(\text{batch size}/128)+1)\times 20$. This adjusted schedule allows larger-batch training runs more of a chance to converge by affording them a greater number of iterations than would normally be allowed under a traditional epoch budget.

3.2 Hyperparameter Tuning

For both K-FAC and SGD, in order to perform training, we must deal with hyperparameters, and we describe this here.

For K-FAC, we use the various techniques discussed in (Osawa et al. 2018). We precondition the Fisher matrix based on Eqn. (1) according to the methodology presented in (Grosse and Martens 2016). For hyperparameter tuning of our CIFAR-10 experiments, we conduct a log-space grid search over 64 configurations with learning rates ranging from 10^{-3} to 2.187, and with damping ranging from 10^{-4} to 0.2187. We also considered expanding grid search for cases where the optimal parameters were on the boundary of this search space. Similarly, for our SVHN experiments, we conduct a log-space grid search over 64 configurations with learning rates ranging from 10^{-5} to 0.02187, and with damping ranging from 10^{-4} to 0.2187. The decay rate for second-order statistics is held constant at 0.9 throughout training. We use update clipping as in (Ba, Grosse, and Martens 2017), with a constant parameter of 0.1.

To ensure a fair comparison between methods, we employ a similarly extensive hyperparameter tuning process for SGD. In particular, we conduct a similar log-space grid search over 64 hyperparameter configurations. For CIFAR-10 experiments, learning rates range from 0.05 to 9.62, and momentum range from 0.9 to 0.999. For SVHN experiments, learning rates range from 0.005 to 0.962, and mo-

[2]This schedule can loosely be regarded as a mixture of an epoch-driven schedule, as in (Hoffer, Hubara, and Soudry 2017), and an iteration-based schedule, as in (He et al. 2016).

mentum range from 0.9 to 0.999.

3.3 Speedup Ratio

We use *speedup ratio* (Golmant et al. 2018) to measure the efficiency of large batch training based on iterations. We define the convergence rate $k_c(m)$ as the fewest number of iterations to reach a certain criteria c under the batch size m, where c is defined as attaining a target accuracy or loss threshold. Here, $k_c(m)$ is a minimum, as it is picked across all configurations of hyperparameters. We then define the *speedup ratio* $s_c(m; m_0)$ as $k_c(m_0)/k_c(m)$, in which we rely on some small batch size m_0 as our reference for convergence rate when comparing to larger batch sizes $m > m_0$. In an ideal scenario, the batch size has no effect on the performance increase per training observation, so in such cases $s_c(m; m_0) = \frac{m}{m_0}$.

It should be noted that for K-FAC speedup we solely measure the number of iterations and ignore the cost of computing the inversion of the Fisher matrix. In fact the latter can become very expensive, and multiple approaches such as stochastic low-rank approximation and/or inexact iterative solves can be used. But for fairness to K-FAC we ignore this cost, since Pytorch does not use the most optimal method for matrix inversion. However, as we will show, K-FAC speedup is far from ideal, even when ignoring this cost of performing more exact computations.

4 Experimental Results

We perform extensive experiments on CIFAR-10 and SVHN datasets with both K-FAC and SGD with ResNet-20/32 and AlexNet. Section 4.1 compares the training and test performances of K-FAC and SGD resulting from extensive hyperparameter tuning for each batch size. Section 4.2 then discusses the large-batch scaling behaviors of K-FAC and SGD and compares them to the ideal scaling scenario (Golmant et al. 2018; Shallue et al. 2018). Finally, Section 4.3 investigates the hyperparameter sensitivity of the K-FAC method.

4.1 Comparing Best Performance of K-FAC and SGD

We run K-FAC and SGD for multiple batch sizes. The highest test accuracy and the lowest training loss achieved for each batch size are plotted in Fig. 1.

In this training context, K-FAC minimizes training loss most effectively for medium-sized batches (around 2^{10}). Inspecting the training trajectories, we found that both the smallest (2^7) and largest batch sizes (2^{14}) needed even more epochs/iterations to converge. For SGD however, training loss is minimized prominently at larger batch sizes. When comparing training trajectories with K-FAC, we found that SGD made much more progress per-iteration in reducing loss, allowing it to minimize the objective with a smaller number of updates, as shown in Fig. 1. A more detailed comparison of the per-iteration progress of SGD versus K-FAC can be found in the following section. For CIFAR-10 experiments, the gap between SGD and K-FAC in large-batch training loss is also present in their generalization performance. SGD's greater efficiency in maximizing per-iteration accuracy allows it to attain a higher level of test performance with the same number of training epochs.

4.2 Large-Batch Scalability of K-FAC and SGD

Training efficiency was measured for each batch size in terms of iterations to a target training loss or test accuracy ($k_c(m)$). The speed up versus batch size relations are displayed in Fig. 2. Dotted lines denote the ideal scaling relationship between batch size and iterations. We normalize each method-target line independently, dividing by the iterations at the smallest batch size $k(2^7)$ so that each of the dotted ideal lines is aligned in the plots, and we take the reciprocal to obtain the speedup function $s(m; 2^7) = k(2^7)/k(m)$, where m is a given batch size. To ensure a fair comparison between batch sizes, similar to what is done in (Golmant et al. 2018), we select target loss values as follows: we wish to analyze how quickly using different batch sizes reach a given threshold. However, not all thresholds are feasible, since large batch sizes may never reach a low training loss, whereas small batches may reach it easily. Thus, for speed up comparison purposes, we set thresholds such that all the batch sizes can reach it. We choose the worst-performing batch size and method. This selection is made after loss-based hyperparameter tuning is finished.

We use the resulting target values in Fig. 2. For both K-FAC and SGD, diminishing return effects are present. In all examined cases, K-FAC deviates from ideal scaling (dotted lines) to a greater extent than SGD, as batch size increases. This difference explains why in Fig. 1 SGD is increasingly able to outperform K-FAC for large batches, for a fixed epoch budget. We note that for both SGD and K-FAC, the linear scaling regime is largely nonexistent, particularly for the highest-performance targets. Fig. 2 shows that K-FAC exhibits worse scaling than SGD.

4.3 Hyperparameter Sensitivity of K-FAC

The hyperparameter tuning spaces on all three models for K-FAC are laid out in Fig. 3, which relates the selected hyperparameters for damping and learning rate with test accuracy. All heatmaps we observe demonstrate a consistent trend in terms of: (i) A positive correlation between damping and learning rate. (ii) A shrinking of the high-accuracy region with increasing batch size. The second point suggests a relationship between batch size and hyperparameter sensitivity for K-FAC, which can be measured in terms of the volume of hyperparameter space corresponding to successful training. In evaluating hyperparameter sensitivity (or inversely robustness), we take the approach of (Shallue et al. 2018), distinguishing between two types of robustness, each corresponding to a different definition of "successful training": (i) Epoch-based robustness, in which success is defined by training to a desired accuracy or loss within a fixed number of epochs. (ii) Iteration-based robustness, in which success is defined by training to a desired accuracy or loss within a fixed number of iterations. It is important to note that a set of hyperparameters considered to be acceptable to a practitioner under an iteration budget may at the same time be considered unacceptable to a practitioner operating under an

Figure 1: From left to right: Best test accuracy / training loss versus batch size for SGD and K-FAC with ResNet20 on CIFAR-10, ResNet32 on CIFAR-10, and AlexNet on SVHN, respectively. Large-batch K-FAC does not achieve higher accuracy or lower losses than large-batch SGD, given the same number of training epochs.

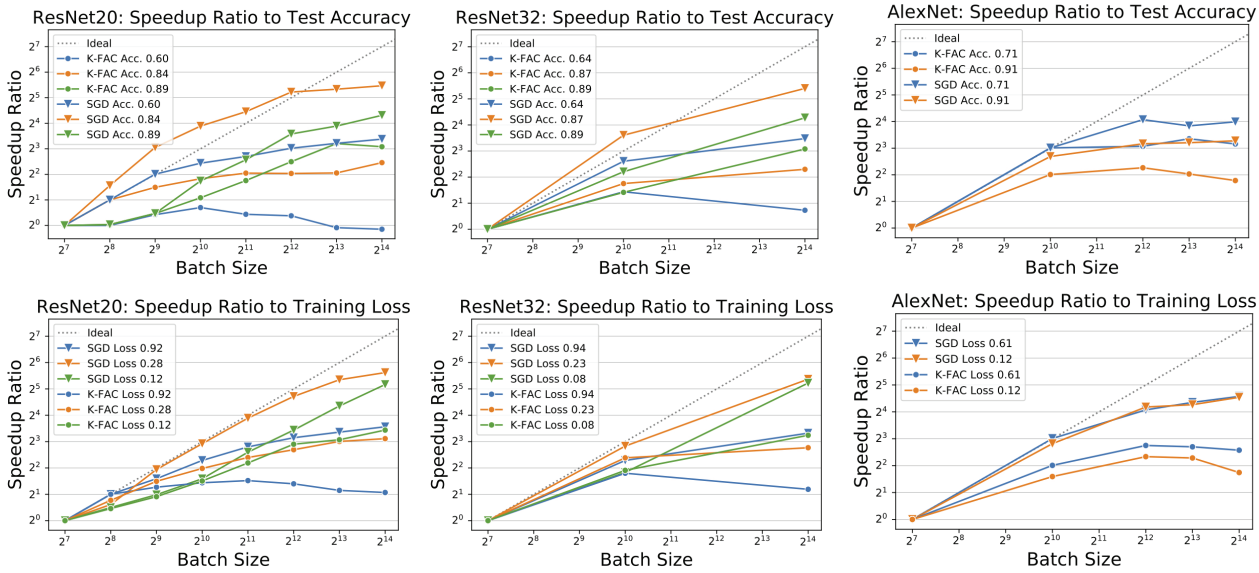

Figure 2: From left to right: speed up to a target training loss / test accuracy versus batch size for both SGD and K-FAC with ResNet20 on CIFAR-10, ResNet32 on CIFAR-10 and AlexNet on SVHN, respectively. The diminishing returns effect can be seen to be more prominent in K-FAC (circles) than in SGD (triangles).

epoch budget. It is for this reason that we make this distinction.

Through this lens, the robustness behavior of K-FAC with ResNet20 on CIFAR-10 is exhibited in Fig. 4. Distributions of training accuracy across hyperparameters are represented by box plots composed from the 64 hyperparameter configurations (8 damping parameters, 8 learning rate parameters). In Fig. 4a and 4d, we show the distributions of test accuracies and training losses for each batch size at the end of training under the adjusted epoch budget. The greater spread of accuracies observed for larger batch sizes indicates that given this budget we should consider batch sizes from 2^{12} to 2^{14} as more sensitive to hyperparameter tuning than batch sizes from 2^{8} to 2^{11}. Informally, if we draw a horizontal line at a desired test accuracy of, e.g., 0.8, then the batch sizes with boxplots containing the majority of their hyperparameter distribution above the 0.8 line should be favored as being more robust.

We can simulate stopping of training in terms of epochs and iterations to extract insight about robustness for other types of budgets than our own. Regardless of the stopping criteria, we expect that longer training will yield greater robustness (although at the cost of significantly higher computational/budget overhead). Figure 4b and 4e shows how the hyperparameter robustness of different batch sizes changes as a function of stopping epoch. Each group along the X-

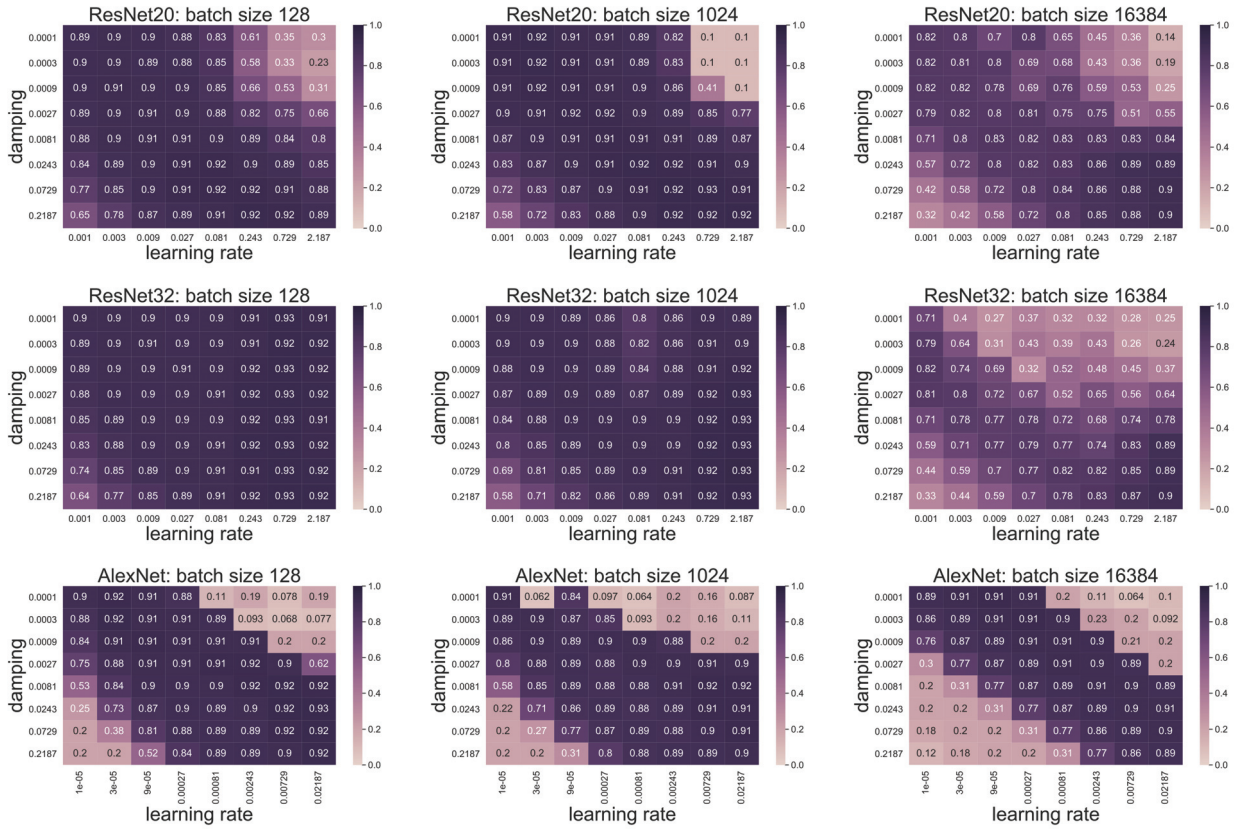

Figure 3: From top to bottom: accuracy at end of training under adjusted-epoch budget versus damping and learning rate for batch sizes 128, 1,024 and 16,384 for CIFAR-10 with ResNet20, CIFAR-10 with ResNet-32, SVHN with AlexNet, respectively. A positive correlation between damping and learning rate is exhibited, as well as a shrinking of the high-accuracy region for large batch sizes.

axis corresponds to a hypothetical epoch budget. The relationship demonstrates that for K-FAC, *robustness increases with amount of training, but more interestingly it decreases with batch size*. This can be observed by noting that for any fixed epoch, the distributions of accuracies corresponding to larger batch sizes fall lower than their smaller-batch counterparts, meaning fewer hyperparameter configurations will fall above a desired accuracy threshold. A similar robustness trend is observed with the distributions of training losses. We perform a similar analysis for iteration budgets in Fig. 4c and 4f, and we find as expected that robustness increases with training. Unlike the case of an epoch budget however, we find that for iteration budgets *robustness increases with larger batch size*. This is observed by noting that the distributions corresponding to large batch are more concentrated towards higher accuracy and lower loss, although the effect is not as pronounced as in the fixed epoch case.

Together, the results show that (i): epoch-based robustness is inversely related to batch size, and (ii): iteration-based robustness is *directly* related to batch size. This is analogous to the findings of (Shallue et al. 2018) for SGD.

5 Conclusions

Despite a number of work claiming record training times of ImageNet using large batch size training, an often overlooked cost is additional hyper-parameter tuning for large batch training. In this work, we performed extensive experimentation and on both CIFAR-10 and SVHN datasets, and find that both K-FAC and SGD exhibit diminishing returns with large batch training. We considered more than 512 hyperparmater values per experiment by considering three models of ResNet20, ResNet32 and AlexNet. We find that K-FAC does not have better training or testing performance than SGD given the same level of training and tuning. This is the case even though we ignored the cost of matrix inversion in K-FAC. Comparing the scalability behaviors of the two methods, we find that K-FAC exhibits a smaller regime of ideal scaling than SGD, suggesting that K-FAC's scalability to large batch training is not better than SGD. Finally, we find that K-FAC exhibits a similar relationship between budget and robustness as SGD, in which K-FAC is less robust to tuning under epoch budgets, but more robust to tuning under iteration budgets, mirroring the findings of similar work in literature for SGD (Shallue et al. 2018).

Taken as a whole, our results suggest that, although K-

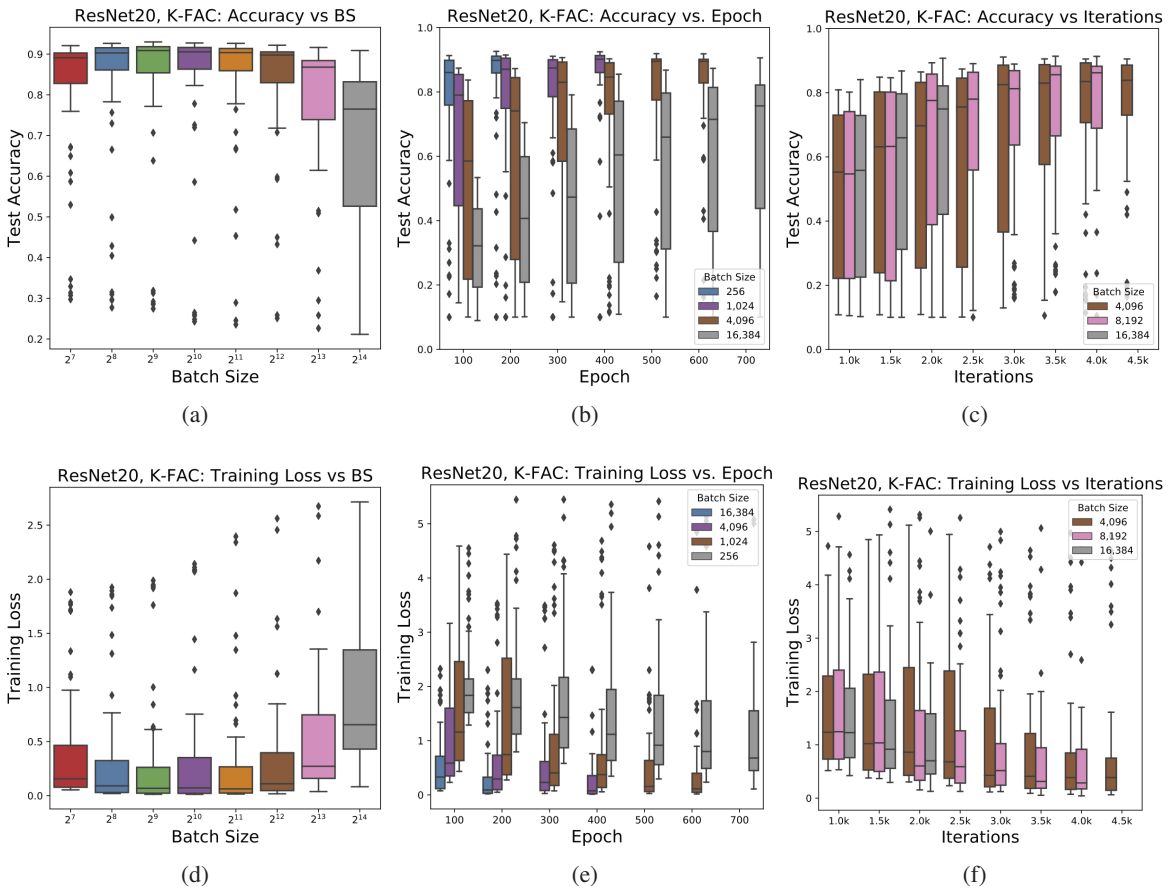

Figure 4: (a)(d): Test accuracy / training loss distribution vs. batch size for K-FAC at the end of training under an adjusted epoch budget. Larger batch sizes result in lower accuracies and higher training losses that are more sensitive to hyperparameter choice. (b)(e): Comparison of test accuracy / training loss distributions over various epochs. Smaller batch sizes provide better solutions that are less sensitive to choice of hyperparameters. (c)(f): Comparison of test accuracy / training loss distributions over various iteration numbers. Large batch sizes exhibit a trend of providing better solutions that are less sensitive to choice of hyperparameters.

FAC has been applied to large batch training scenarios, it encounters the same large-batch issues to an equal or greater extent as SGD, and that the problem of large batch training is still largely unresolved. It remains to be seen whether other variants of sub-sampled Newton methods (Roosta-Khorasani and Mahoney 2016a; 2016b) can lead to improved results or whether this is a more ubiquitous aspect of stochastic optimization algorithms applied to non-convex optimization problems of interest in machine learning.

We believe that it is important for every work to state its limitations (in general, but in particular in this area). We were particularly careful to perform extensive experiments and did some initial testing for each dataset to choose the right hyper-parameters that were not tuned. For the two hyperparameters that we tuned, we considered the same range for all batch sizes. It is possible to augment this search space using rules for large batch training such as scaling learning rate. However, we found that for large batch size this technique sometimes leads to divergence and poor results as compared to our fixed-grid search space. Here we solely looked at pure SGD and K-FAC, and did not consider other approaches such as LARS (You, Gitman, and Ginsburg 2017), or adaptive batch size methods (Smith et al. 2017; Devarakonda, Naumov, and Garland 2017; Yao et al. 2018a; Mu et al. 2018). We leave this as part of future work.

References

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

Ba, J.; Grosse, R.; and Martens, J. 2017. Distributed second-order optimization using Kronecker-factored Approximations. In *International Conference on Learning Representations*.

Devarakonda, A.; Naumov, M.; and Garland, M. 2017. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*.

Gholami, A.; Azad, A.; Jin, P.; Keutzer, K.; and Buluc, A.

2018. Integrated model, batch and domain parallelism in training neural networks. *ACM Symposium on Parallelism in Algorithms and Architectures(SPAA'18)*.

Ginsburg, B.; Gitman, I.; and You, Y. 2018. Large batch training of convolutional networks with layer-wise adaptive rate scaling.

Golmant, N.; Vemuri, N.; Yao, Z.; Feinberg, V.; Gholami, A.; Rothauge, K.; Mahoney, M. W.; and Gonzalez, J. 2018. On the computational inefficiency of large batch sizes for stochastic gradient descent. *CoRR* abs/1811.12941.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017a. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Goyal, P.; Dollár, P.; Girshick, R. B.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017b. Accurate, large minibatch SGD: Training imagenet in 1 hour. *CoRR* abs/1706.02677.

Grosse, R., and Martens, J. 2016. A Kronecker-factored Approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning*, 573–582.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hoffer, E.; Hubara, I.; and Soudry, D. 2017. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, 1731–1741.

Iandola, F. N.; Moskewicz, M. W.; Ashraf, K.; and Keutzer, K. 2016. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2592–2600.

Jia, X.; Song, S.; He, W.; Wang, Y.; Rong, H.; Zhou, F.; Xie, L.; Guo, Z.; Yang, Y.; Yu, L.; et al. 2018. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*.

Keskar, N. S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; and Tang, P. T. P. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

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

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Ma, S.; Bassily, R.; and Belkin, M. 2017. The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. *arXiv preprint arXiv:1712.06559*.

Martens, J., and Grosse, R. B. 2015. Optimizing neural networks with Kronecker-factored Approximate Curvature. *CoRR* abs/1503.05671.

Martin, C. H., and Mahoney, M. W. 2018. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *arXiv preprint arXiv:1810.01075*.

Martin, C. H., and Mahoney, M. W. 2019. Traditional and heavy-tailed self regularization in neural network models. In *Proceedings of the 36th International Conference on Machine Learning*, 4284–4293.

McCandlish, S.; Kaplan, J.; Amodei, D.; and Team, O. D. 2018. An empirical model of large-batch training. *CoRR* abs/1812.06162.

Mu, N.; Yao, Z.; Gholami, A.; Keutzer, K.; and Mahoney, M. W. 2018. Parameter re-initialization through cyclical batch size schedules. *arXiv preprint arXiv:1812.01216*.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading digits in natural images with unsupervised feature learning.

Osawa, K.; Tsuji, Y.; Ueno, Y.; Naruse, A.; Yokota, R.; and Matsuoka, S. 2018. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs. *arXiv preprint arXiv:1811.12019*.

Roosta-Khorasani, F., and Mahoney, M. W. 2016a. Sub-sampled Newton methods I: globally convergent algorithms. *arXiv preprint arXiv:1601.04737*.

Roosta-Khorasani, F., and Mahoney, M. W. 2016b. Sub-sampled Newton methods II: Local convergence rates. *arXiv preprint arXiv:1601.04738*.

Shallue, C. J.; Lee, J.; Antognini, J. M.; Sohl-Dickstein, J.; Frostig, R.; and Dahl, G. E. 2018. Measuring the effects of data parallelism on neural network training. *CoRR* abs/1811.03600.

Smith, S. L.; Kindermans, P.-J.; Ying, C.; and Le, Q. V. 2017. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.

Yamazaki, M.; Kasagi, A.; Tabuchi, A.; Honda, T.; Miwa, M.; Fukumoto, N.; Tabaru, T.; Ike, A.; and Nakashima, K. 2019. Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds. *arXiv preprint arXiv:1903.12650*.

Yao, Z.; Gholami, A.; Keutzer, K.; and Mahoney, M. W. 2018a. Large batch size training of neural networks with adversarial training and second-order information. *arXiv preprint arXiv:1810.01021*.

Yao, Z.; Gholami, A.; Lei, Q.; Keutzer, K.; and Mahoney, M. W. 2018b. Hessian-based analysis of large batch training and robustness to adversaries. *arXiv preprint arXiv:1802.08241*.

You, Y.; Zhang, Z.; Hsieh, C.; and Demmel, J. 2017. 100-epoch imagenet training with alexnet in 24 minutes. *CoRR* abs/1709.05011.

You, Y.; Gitman, I.; and Ginsburg, B. 2017. Scaling SGD batch size to 32k for ImageNet training. *arXiv preprint arXiv:1708.03888*.

Zhang, G.; Wang, C.; Xu, B.; and Grosse, R. 2019. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*.