Increasing batch size through instance repetition improves generalization

Anonymous authors
Paper under double-blind review

Abstract

Large-batch SGD is important for scaling training of deep neural networks. However, without fine-tuning hyperparameter schedules, the generalization of the model may be hampered. We propose to use batch augmentation: replicating instances of samples within the same batch with different data augmentations. Batch augmentation acts as a regularizer and an accelerator, increasing both generalization and performance scaling for a fixed budget of optimization steps. We analyze the effect of batch augmentation on gradient variance and show that it empirically improves convergence for a wide variety of networks and datasets. Our results show that batch augmentation reduces the number of necessary SGD updates to achieve the same accuracy as the state-of-the-art. Overall, this simple yet effective method enables faster training and better generalization by allowing more computational resources to be used concurrently.

1 Introduction

Deep neural network training is a computationally-intensive problem, whose performance is inherently limited by the sequentiality of the Stochastic Gradient Descent (SGD) algorithm. In a common variant of the algorithm, a batch of samples is used at each step for gradient computation, accumulating the results to compute the descent direction. Batch computation enables *data parallelism* (Ben-Nun & Hoefler, 2018), which is necessary to scale training to a large number of processing elements.

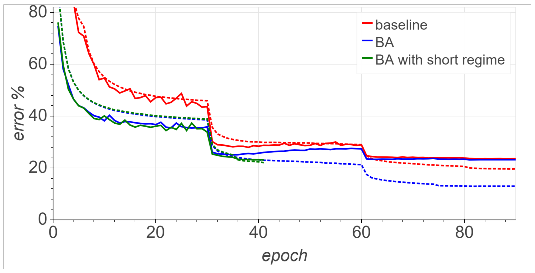


Figure 1: Impact of Batch Augmentation (BA, with four augmentations per sample) on ResNet-50 and ImageNet. Depicted – training (dashed) and validation (solid) errors.

Increasing batch size while mitigating accuracy degradation is actively researched in the ML and systems communities Goyal et al. (2017); Kurth et al. (2017); Jia et al. (2018); Mikami et al. (2018); Osawa et al. (2018); Ying et al. (2018). Shallue et al. (2018b) comprehensively study the relation between batch size and convergence, whereas other works focus on increasing parallelism for a specific setting or hardware. Using such techniques, it is possible to reduce the time to successfully train ResNet-50 (He et al., 2016) on the ImageNet (Deng et al., 2009) dataset down to 132 seconds (Ying et al., 2018), to the point where the performance bottleneck is reported to be input data processing (I/O) time.

The key to supporting large batch training often involves fine-tuning the base Learning Rate (LR), per-layer LR You et al. (2017), LR schedules Goyal et al. (2017); You et al. (2017), or the optimization step Krishnan et al. (2018); Hoffer et al. (2017); Osawa et al. (2018). These methods typically use higher LRs to account for the lower gradient variance in large batch updates. However, without fine-tuning, large batch training often results in degraded generalization. It was suggested (Keskar et al., 2017) that this is caused by a tendency of such low variance updates to converge to "sharp minima" .

In this work, we propose **Batch Augmentation** (BA), which enables to control the gradient variance while increasing batch size. Using larger augmented batches, we can better utilize the computational resources without the cost of additional I/O. In fact, it is even possible to achieve better generalization accuracy while adopting existing, standard LR schedules (see Figure 1).

Our main contributions are:

- Introducing BA and its possible usages.
- Empirical results for BA properties, resource utilization and gradient variance.
- Convergence results on multi-GPU nodes and a Cray supercomputer with 5,704 GPUs.

1.1 Large batch training of neural networks

Recent approaches by Hoffer et al. (2017), Goyal et al. (2017), You et al. (2017) and others show that by adapting the optimization regime (i.e., hyperparameter schedule), large batch training can achieve equally good (and sometimes even better) generalization as training with small batches.

Hoffer et al. (2017) argue that the quality of the optimized model stems from the number of SGD iterations, rather than the number of cycles through training data (epochs), and increase the number of steps w.r.t. the batch size. They then train ImageNet without accuracy degradation using additional epochs, adapting the points in which LR is reduced (Regime Adaptation), and normalizing subsets of the batch in a process called Ghost Batch Normalization (GBN).

Goyal et al. (2017) use a batch size of 8,192 and adopt a "gradual warmup" scheme, in which the LR linearly increases to the base LR after 5 epochs, after which the regime resumes normally. You et al. (2017) increases the batch size to 32,768 by using Layer-wise Adaptive Rate Scaling (LARS), as well as polynomial LR decay following warmup, with some reduction in accuracy. Ying et al. (2018) employ distributed batch normalization and gradient accumulation to retain validation accuracy on ImageNet with 32,768 images per batch and 1,024 TPU devices. Jia et al. (2018) make use of 16-bit floating point ("half-precision") and further tune hyperparameters (e.g., weight decay) to reduce communication and enable training with batches of size 65,536.

Other large-batch methods utilize second-order information during training. The Neumann optimizer Krishnan et al. (2018) uses a first-order approximation of the inverse Hessian using the Neumann Series, and is able to train up to batches of size 32,000 without accuracy degradation, albeit converging fastest when batches of 1,600 are used. The Kronecker Factorization (K-FAC) second-order approximation was also used to accelerate the convergence of deep neural network training Osawa et al. (2018), achieving 74.9% validation accuracy on ImageNet after 45 epochs, batch size of 32,768 on 1,024 nodes.

In contrast, Masters & Luschi (2018) suggested that small batch updates may still provide benefits over large batch ones, showing better results over several tasks, with higher robustness to hyperparameter selection. The training process in this case, however, is sequential and cannot be distributed over multiple processing elements. An extensive survey by Shallue et al. (2018a) showed that the ability to scale to large minibatch sizes is highly dependent on the model used. It was also noted that optimal values of training do not consistently follow any simple relation to the batch size. Specifically, it was shown that common learning rate heuristics do not hold across all tasks and batch sizes.

Batch Augmentation enables all benefits of large-batch training, while keeping the number of input examples constant and minimizing the number of hyperparameters. Furthermore, it improves generalization as well as hardware utilization. We now continue to discuss existing data augmentation techniques that we will later use for Batch Augmentation.

1.2 A primer on data augmentation

A common practice in training modern neural networks is to use data augmentation — applying different transformations to each input sample. For example, in image classification tasks, for any input image, a random crop of varying size and scale is applied to it, potentially together with rotation, mirroring and even color jittering (Krizhevsky et al., 2012). Data augmentations were repeatedly found to provide efficient and useful regularization, even in semi-supervised settings (Xie et al., 2019), often accounting for significant portion of the final generalization performance (Zagoruyko, 2016; DeVries & Taylor, 2017).

Several works attempt to learn how to generate good data augmentations. For example, Bayesian approaches based on the training set distribution (Tran et al., 2017), generative approaches based on GANs (Antoniou et al., 2017; Sixt et al., 2018) and search methods aim to find the best data augmentation policy (Cubuk et al., 2018). Our approach is orthogonal to those methods, and even better results can be obtained by combining them.

Other regularization methods, such as Dropout (Srivastava et al., 2014) or ZoneOut (Krueger et al., 2016), although not explicitly considered as data augmentation techniques, can be considered as such by viewing them as random transforms over inputs for intermediate layers. These methods were also shown to benefit models in various tasks. Another related regularization technique called "Mixup" was introduced by Zhang et al. (2018). Mixup uses mixed inputs from two separate samples with different classes, and uses their labels mixed by the same amount as the target.

2 Batch Augmentation

In this work, we suggest leveraging the merits of data augmentation together with large batch training, by using multiple instances of a sample in the same batch.

We consider a model with a loss function $\ell(\mathbf{w}, \mathbf{x}_n, \mathbf{y}_n)$ where $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N}$ is a dataset of N data sample-target pairs, where $x_n \in X$ and $T : X \to X$ is some data augmentation transformation applied to each example, e.g., a random crop of an image. The common training procedure for each batch consists of the following update rule (here using vanilla SGD with a learning-rate η and batch size of B, for simplicity):

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{B} \sum_{n \in \mathcal{B}(k(t))} \nabla_{\mathbf{w}} \ell\left(\mathbf{w}_t, T(\mathbf{x}_n), \mathbf{y}_n\right)$$

where $k\left(t\right)$ is sampled from $[N/B] \triangleq \{1, \dots, N/B\}$, $\mathcal{B}\left(t\right)$ is the set of samples in batch t, and we assume for simplicity that B divides N.

We suggest to introduce M multiple instances of the same input sample by applying the transformation T_i, here denoted by subscript $i \in [M]$ to highlight the fact that they are different from one another. We now use the slightly modified learning rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{M \cdot B} \sum_{i=1}^{M} \sum_{n \in \mathcal{B}(k(t))} \nabla_{\mathbf{w}} \ell\left(\mathbf{w}_t, T_i(\mathbf{x}_n), \mathbf{y}_n\right)$$

effectively using a batch of $M \cdot B$ composed of B samples augmented by M different transformations.

We note that this updated rule can be computed either by evaluating on the whole $M \cdot B$ batch or by accumulating M instances of the original gradient computation. Using large batch updates as part of batch augmentations does not change the number of SGD iterations that are performed per epoch.

Batch augmentation (BA) can also be used to transform over intermediate layers, rather than just the inputs. For example, we can use the common Dropout regularization method (Srivastava et al., 2014) to generate multiple instances of the same sample in a given layer, each with its Dropout mask.

Batch augmentation can be easily implemented in any framework with reference PyTorch and TensorFlow implementations[1]. To further highlight the ease of incorporating these ideas, we note that BA can be added to any training code by merely modifying the input pipeline – augmenting each batch that is fed to the model.

2.1 Countering large batch issues with data augmentation

Standard batch SGD averages the gradient over different samples, while BA additionally averages the gradient over several transformed instances $T\left(x_n\right)$ of the same samples. The augmented instances describe the same samples, typically with only small changes, and produce correlated gradients within the batch. BA can achieve variance reduction that is significantly lower than the $1/B$ reduction, which may occur with an uncorrelated sum of B samples.

[1] Available at `https://github.com/paper-submissions/batch-duplicates`

In order to achieve such decreased variance reduction, we must assume certain necessary conditions on T. Specifically, data augmentations should be designed to produce, in expectation, gradients that are more correlated with the original sample than other samples in the input dataset. More formally,

$$\mathbb{E}_{n\in[N]}\left[\mathrm{Corr}\left(\nabla_{\mathbf{w}}^{(n)},\nabla_{\mathbf{w}}\ell\left(\mathbf{w},T\left(x_n\right),y_n\right)\right)\right] > \mathbb{E}_{n,m\in[N],\,n\neq m}\left[\mathrm{Corr}\left(\nabla_{\mathbf{w}}^{(n)},\nabla_{\mathbf{w}}^{(m)}\right)\right]$$

for $\nabla_{\mathbf{w}}^{(n)} \triangleq \nabla_{\mathbf{w}}\ell\left(\mathbf{w},x_n,y_n\right)$. Later, in section 3, we measure the effects of data augmentations used in practice and show that this property is maintained for standard image classification datasets. Thus, BA reduces variance less, as it adds additional highly correlated samples to the averaging of gradients.

Such decreased variance reduction might be helpful in mitgating large-batch training issues, as we explain next. Previous works (Keskar et al., 2017; Nar & Sastry, 2018; Wu et al., 2018) suggested that large-batch training issues may result from an implicit bias in the SGD training process: with large batch sizes, SGD selects different ("new") minima with worse generalization than the original minima selected by small batch training. This issue can be partially mitigated by increasing the learning rate to specific value Hoffer et al. (2017); Goyal et al. (2017), which will make these new minima inaccessible again, while keeping the original minima accessible. However, Shallue et al. (2018a) observed there is no general effective rule on how to change the learning rate with the batch size — as its optimal scaling with batch size may change with models, datasets, or other hyperparameters. Moreover, merely changing the learning rate may not be sufficient for very large batch sizes, as eventually SGD may not be able to discriminate between the new and original minima. In Appendix A we give a formal treatment of these issues, and explain why the decreased variance reduction properties of BA might be helpful to counter such issues.

Therefore, compared to standard large batch training, batch-augmentations enable the model to train on more augmentations while modifying the optimization dynamics less.

3 Characterizing Batch Augmentation

We proceed to empirically study different aspects of Batch Augmentation, including measurements of gradient correlation and variance, and performance, and utilization analysis of augmented batches.

Data Augmentation To analyze the variance reduction of BA, we empirically show that data augmentations T fulfill the assumption that they create correlated gradients in expectation. Table 1 lists the validation accuracy and median correlations (100 samples) between gradients of ResNet-44 on the Cifar10 dataset, at initialization, after 5 epochs, and after convergence at 93 epochs. In the table, it is clear to see that augmentations produce gradients that are considerably more correlated than images in different classes, and even within the same class. Moreover, the Cutout augmentation slightly decreases the gap between augmented and different images of the same class. As for the network state, when using random weights, interestingly all gradients of the same class correlate with each other.

The results reaffirm that augmentations produce gradients that are considerably more correlated than images in different classes, and even within the same class. Moreover, the results indicate that, at first, there is a particular direction to descend in expectation in order to learn classifying a certain class of images, regardless of the actual sample. Correlation then decreases as training progresses.

Variance Reduction To empirically evaluate the effect of variance reduction in BA, we measured the L^2 norm of the weight gradients throughout the training for the setting described in Section 4.1. We use the L^2 norm as a proxy for variance reduction, as each gradient can be viewed as a random variable. As expected, the variance reduction is reflected in the norm values as can be seen in Figure 2.

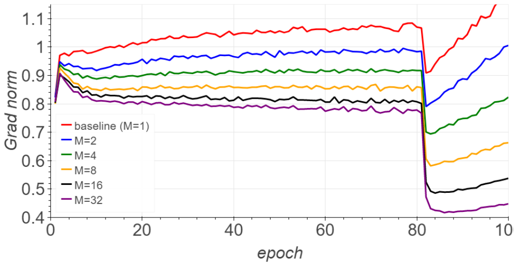


Figure 2: Comparison of gradient L^2 norm (ResNet44 + cutout, Cifar10, $B = 64$) between the baseline ($M = 1$) and batch augmentation with $M \in \{2, 4, 8, 16, 32\}$

Performance A theoretical understanding of the performance of parallel algorithms can be

Table 1: ResNet-44 Gradient correlation on Cifar10. We measure the Pearson correlation coefficient ρ between random images and augmented versions thereof $\rho\left(x, T\left(x\right)\right)$, as well as for random images of the same class $\rho\left(x, y\right)$ and different classes $\rho\left(z, w\right)$. Augmentation types: **RC**=Random Crop, **F**=flip, **CO**=Cutout.

Measure	Network State		
	Initial	Partially Trained	Fully Trained
Epoch	0	5	93
Validation Accuracy	9.63%	63.24%	95.43%
$\rho\left(x, T\left(x\right)\right)$ (RC,F)	0.99 ± 0	0.56 ± 0.09	0.13 ± 0.13
$\rho\left(x, T\left(x\right)\right)$ (RC,F,CO)	0.99 ± 0	0.51 ± 0.08	0.09 ± 0.08
$\rho\left(x, y\right)$	0.99 ± 0	0.42 ± 0.06	0.04 ± 0.03
$\rho\left(z, w\right)$	-0.11 ± 0.01	-0.04 ± 0.06	0 ± 0.02

derived from the overall number of operations and the longest dependency path between them, which is a measure of the sequential part that fundamentally constrains the computation time (i.e., a work-depth model Blumofe & Leiserson (1999)). In BA and standard large-batch training, the overall number of operations (*work*) increases proportionally to the overall batch size, i.e., $M \cdot B$. However, the sequential part (*depth*), which is proportional to the number of SGD iterations, decreases as a result of faster LR schedules in BA, or shorter epochs in standard large-batch training. In essence, serialization can be reduced at the expense of more work, which increases the average parallelism.

Factoring for I/O and communication, BA also poses an advantage over standard large-batch training. BA decreases the dependency on external data, as in each iteration every processor can read the inputs and decode them once, applying augmentations locally. This increases scalability in state-of-the-art implementations, where input processing pipeline is the current bottleneck Ying et al. (2018). Communication per iteration, on the other hand, is governed by the number of participating processing elements, in which the cost remains equivalent to standard large-batch training.

Our empirical results (e.g., Figure 1) show that in BA, the number of iterations may indeed be reduced as M increases. This indicates that the time to completion can remain constant with better generalization properties. Thus, BA, in conjunction with large batches, opens an interesting tradeoff space between the work and depth of neural network training.

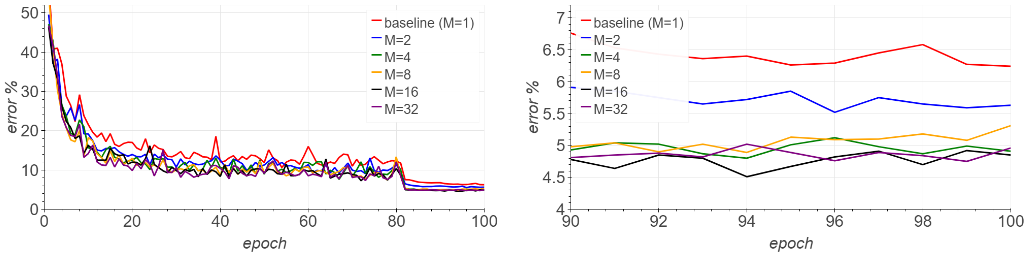
4 Convergence analysis

To evaluate the impact of Batch Augmentation (BA), we used several common datasets and neural network based models. For each one of the models, unless explicitly stated, we tested our approach using the original training regime and data augmentation described by its authors. To support our claim, *we neither change the learning rate nor the number of training steps* for BA. For each result, we compare BA to two separate baselines — one with the same number of training iterations and one, additionally, with the same number of seen samples (achieved by enlarging the used batch-size). For large batch cases, we also used alternative learning rates in our measurements, as suggested in previous works (Goyal et al., 2017; Shallue et al., 2018a).

4.1 Cifar10/100

We first used the popular image classification datasets Cifar10/100, introduced by Krizhevsky (2009). For both datasets, the common data augmentation technique is described by He et al. (2016). In this method, the input image is padded with 4 zero-valued pixels at each side, top, and bottom. A random 32×32 part of the padded image is then cropped and with a 0.5 probability flipped horizontally. This augmentation method has a rather small space of possible transforms ($9 \cdot 9 \cdot 2 = 162$), and so it is quickly exhausted by even a $M \approx$ 10s of simultaneous instances.

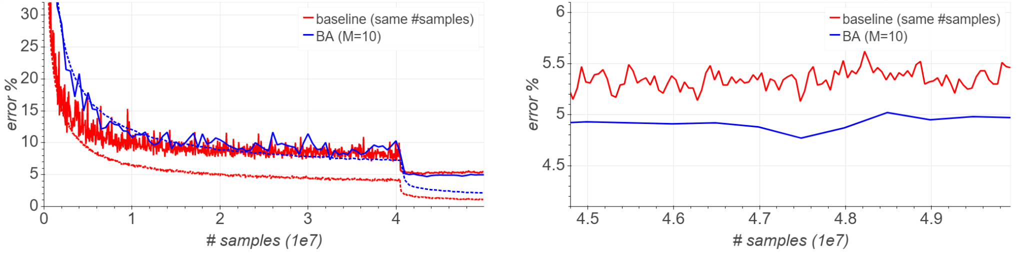
We therefore speculated that using a more aggressive augmentation technique, with larger option space, will yield more noticeable difference when batch augmentation is used. We chose to use the recently introduced "Cutout" (DeVries & Taylor, 2017) method, that was noted to improve the generalization of models on various datasets considerably. Cutout uses randomly positioned zero-valued squares within images, thus increasing the number of possible transforms by $\times 30^2$.



(a) Validation error

(b) Final Validation error

Figure 3: Impact of batch augmentation (ResNet44, Cifar10). We used the original (red) training regime with $B = 64$, and compared to batch augmentation with $M \in \{2, 4, 8, 16, 32\}$.



(a) Training (dashed) and validation error

(b) Training (dashed) and validation final error

Figure 4: A comparison between (1) baseline with B=640 and 10x more epochs. (2) our batch augmentation (BA) method with M=10.

We tested batch augmentation using a ResNet44 (He et al., 2016) over the Cifar10 dataset (Krizhevsky, 2009) together with cutout augmentation (DeVries & Taylor, 2017). We used the original regime by He et al. (2016) with a batch of $B = 64$. We then compared the learning curve with training using batch augmentation with $M \in \{2, 4, 8, 16, 32\}$ different transforms for each sample in the batch, effectively creating a batch of $64 \cdot M$.

Figure 3 shows an improved validation convergence speed (in terms of epochs), with a significant reduction in final validation classification error (Figure 3b). This trend largely continues to improve as M is increased, consistent with our expectation. We verified these results using a variety of models (Simonyan & Zisserman, 2014; He et al., 2016; Zagoruyko, 2016; Liu et al., 2018; Real et al., 2018; Huang et al., 2017) using various values of M, depending on our ability to fit the $M \cdot B$ within our compute budget. Results appear on Table 4. Our best result was achieved using the AmoebaNet final Cifar10 model (Real et al., 2018).

In all our experiments we have observed *significant improvements* to the final validation accuracy, as well as faster convergence in terms of accuracy per epoch. Moreover, we managed to achieve high validation accuracy much quicker with batch augmentation. We trained a ResNet44 with Cutout on Cifar10 for half of the iterations needed for the baseline, using batch augmentation, larger learning rate, and faster learning rate decay schedule. We managed to achieve 94.15% accuracy in only 23 epochs for ResNet44, whereas the baseline achieved 93.07% with over four times the number of iterations (100 epochs). When the baseline is trained with the same shortened regime there is a significant accuracy degradation. This indicates not only an accuracy gain, but a potential runtime improvement for a given hardware. We note that for AmoebaNet with $M = 12$ we reach 94.46% validation accuracy after 14 epochs without any modification to the LR schedule.

We were additionally interested to verify that improvements gained with BA were not caused by simply viewing more sample instances during training. To make this distinction apparent, we compare with a training regime that guarantees a fixed number of seen examples. In this method, the number of epochs is increased so that the number of iterations is fixed when using a larger batch (by the same factor of M). This alternative baseline is comparable to BA with respect to the number of instances seen for each sample over the course of training. Using the same settings (ResNet44, Cifar10), we find an accuracy gain of 0.5% over the 93.07% result obtained using the fixed-number-of-samples baseline. Figure 4 shows these results, and additional comparisons appear in Table 4 (Baseline with fixed number of samples). We also note the fact that a moderate grid search for alternative learning rates (see Appendix C.1 for details) did not improve the baseline results, affirming the strong results of BA under a fixed steps budget.

Table 2: Validation accuracy (Top1) results for Cifar, ImageNet models. Bottom: test perplexity result and BLEU score on Penn-Tree-Bank (PTB) and WMT datasets. We compare BA to two baselines – (1) "Fixed #Steps" - original regime with same number of training steps as BA (2) "Fixed #Samples" - where in addition, the same number of samples as BA were observed (using $M \cdot B$ batch size).

Network	Dataset	M	Baseline		BA
			Fixed #Steps	+ Fixed #Samples	
ResNet44	Cifar10	40	93.70%	93.80%	**95.43%**
VGG16	Cifar10	32	93.82%	94.49%	**95.32%**
Wide-ResNet28-10	Cifar10	6	96.60%	96.60%	**97.15%**
DARTS	Cifar10	8	97.65%	97.63%	**97.85%**
AmoebaNet	Cifar10	8	98.16%	98.10%	**98.24%**
ResNet44	Cifar100	40	72.97%	70.30%	**74.13%**
VGG	Cifar100	32	73.03%	67.20%	**75.50%**
Wide-ResNet28-10	Cifar100	10	79.85%	80.12%	**83.45%**
DenseNet100-12	Cifar100	4	77.73%	75.35%	**78.80%**
AlexNet	ImageNet	8	58.25%	57.60%	**62.31%**
MobileNet	ImageNet	4	70.60%	69.50%	**71.40%**
ResNet50	ImageNet	4	76.30%	75.70%	**76.86%**
Word-level LSTM	PTB	10	58.8 ppl	58.8 ppl	**58.6** ppl
Transformer (base)	WMT En-De	4	26.88 BLEU	27.13 BLEU	**27.49** BLEU

4.2 IMAGENET

As a larger scale evaluation, we used the ImageNet dataset (Deng et al., 2009), containing more than 1.2 million images with 1,000 different categories. We evaluate three models — AlexNet(Krizhevsky et al., 2012), MobileNet(Howard et al., 2017) and ResNet50 (He et al., 2016). For details regarding training and hyper-parameters see Appendix C.2.

To fit within our time and compute budget constraints, we used a mild $M = 4$ batch augmentation factor for ResNet and MobileNet, and $M = 8$ for AlexNet. We again observe an improvement with all models in their final validation accuracy (Table 4). Using a linear scaling of learning rate as suggested by Goyal et al. (2017); Shallue et al. (2018a) also didn't improve the measured baseline accuracy for large batch training.

The AlexNet model had the most dramatic improvement – yielding more than 4% improvement in absolute validation accuracy compared to our baseline, and more than 2% than previously best published results (You et al., 2017). We also highlight the fact that models reached a high validation accuracy quicker. For example, the ResNet50 model, without modification, reached a 75.7% at epoch 35 – only 0.6% shy of the final accuracy achieved at epoch 90 with the baseline model (Figure 1). The increase in validation error between epochs $30 - 60$ suggests that either learning rate or weight-decay values should be altered as discussed by Zagoruyko (2016) who witnessed similar effects. This led us to believe that with careful hyperparameter tuning of the training regime, we can shorten the number of epochs needed to reach the desired accuracy and even improve it further.

By adapting the training regime to the improved convergence properties of BA, we were able to reduce the number of iterations needed to achieve the required accuracy. Using the same base LR (0.1), and reducing by a factor of 0.1 after epochs 30 and 35 allowed us to reach the same improved accuracy of 76.86% after only 40 epochs. An even faster schedule where the LR is reduced at epochs $15, 20$, and 22 yields the previous 75.7% at epoch 23.

4.3 DROPOUT AS INTERMEDIATE AUGMENTATION

We also tested the ability of batch augmentation to improve results in tasks where no explicit augmentations are performed on input data. An example for this kind of task is sequence modeling, where the input is fed in a deterministic fashion and noise is introduced in intermediate layers in the form of Dropout (Srivastava et al., 2014), DropConnect (Wan et al., 2013), or other forms of regularization (Krueger et al., 2016; Merity et al., 2017).

We used the base Transformer model by Vaswani et al. (2017) over WMT16 en-de task, along with the original hyper-parameters. We used our own implementation and trained the model for $100K$

iterations (details appear in Appendix C.3). The use of multiple sample instances within the batch caused each instance to be computed with a different random Dropout mask. Using BA with $M = 4$ and a batch-size of 4096 tokens, we find an improvement of 0.36 in BLEU score (see Table 4).

We also tested the language model described by Merity et al. (2017) and the proposed setting of an LSTM word-level language model over the Penn-Tree-Bank (PTB) dataset. We used a 3-layered LSTM of width 1,150 and embedding size of 400, with Dropout regularization on both input ($p = 0.4$) and hidden state ($p = 0.25$), with no fine-tuning. We used $M = 10$, increasing the effective batch-size from 20 to 200. We again observed a positive effect, yet more modest compared to the previous experiments, reaching a 0.2 improvement in final test perplexity compared to the baseline.

4.4 Regularization impact on BA

We were interested to see interaction between results obtained with BA together with recent regularization methods such as label-smoothing (Szegedy et al., 2016), mixup (Zhang et al., 2018) and manifold-mixup (Verma et al., 2018). We tested BA with mixup and find that the benefit in accuracy persists in batch-augmented training and can be used together with regularization to further improve generalization (see Table 4 in Appendix). We additionally observed that using test-time-augmentation (TTA), yields better relative improvement in models trained using BA (see Table 5 in the appendix). We speculate this is due to the fact that BA optimizes over several transforms of each input – which is more suited to a TTA scheme where classification is done over several instances of the same sample.

4.5 Distributed Batch Augmentation

To support large-scale clusters, we implement distributed BA over TensorFlow and Horovod Sergeev & Balso (2018). We test our implementation on CSCS Piz Daint, a Cray XC50 supercomputer with NVIDIA Tesla P100 GPUs. The full implementation and system details are detailed in Appendix D.

When distributing the computation, if we naively replicate a small batch M times on each node, we will degenerate the batch normalization process by normalizing a small set of images with multiple augmentations. Instead, our implementation ensures that every M nodes would load the same batch, so different images are normalized together. We achieve this effect by synchronizing the random seeds of the dataset samplers in every M nodes (but not the data augmentation seeds). This also allows the system to load the same files from the parallel filesystem once, followed by broadcasting.

The results in the supplementary material show that BA produces consistently higher validation accuracy on more nodes, successfully scaling to an effective batch size of 2,560 on 40 nodes, without tuning the LR schedule as Goyal et al. (2017) and exhibiting reduced communication cost due to I/O optimizations. When using the large-batch LR schedule Goyal et al. (2017) with $B = 8192$, running on 128 nodes results in an accuracy of 75.86%, whereas $M = 4$ and 512 nodes result in 76.51%.

5 Conclusion

In this work, we introduced "Batch Augmentation" (BA), a simple yet effective method to improve generalization performance of deep networks by training with large batches composed of multiple transforms of each sample. We have demonstrated significant improvements on various datasets and models, with both faster convergence per epoch, as well as better final validation accuracy.

We suggest a theoretical analysis to explain the advantage of BA over traditional large batch methods. We also show that BA causes a decrease in gradient variance throughout training, reflected in the gradient's ℓ_2 norm in each optimization step. This may be used in the future to search and adapt more suitable training hyperparameters, enabling faster convergence and even better performance.

Recent hardware developments allowed the community to use larger batches without increasing the wall clock time either by using data parallelism or by leveraging more advanced hardware. However, several papers claimed that working with large batch results in accuracy degradation (Masters & Luschi, 2018; Golmant et al., 2019). Here we argue that by using multiple instances of the same sample we can leverage the larger batch capability to increase accuracy. These findings give another reason to prefer training settings utilizing significantly larger batches than those advocated in the past.

REFERENCES

Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *arXiv preprint arXiv:1802.09941*, 2018.

R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent, 2019. URL `https://openreview.net/forum?id=S1en0sRqKm`.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

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

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

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

Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

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. In *ICLR*, 2017.

Shankar Krishnan, Ying Xiao, and Rif. A. Saurous. Neumann optimizer: A practical optimization algorithm for deep neural networks. In *International Conference on Learning Representations*, 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 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.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Thorsten Kurth, Jian Zhang, Nadathur Satish, Evan Racah, Ioannis Mitliagkas, Md. Mostofa Ali Patwary, Tareq Malas, Narayanan Sundaram, Wahid Bhimji, Mikhail Smorkalov, Jack Deslippe, Mikhail Shiryaev, Srinivas Sridharan, Prabhat, and Pradeep Dubey. Deep learning at 15pf: Supervised and semi-supervised classification for scientific data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pp. 7:1–7:11. ACM, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U.-Chupala, Yoshiki Tanaka, and Yuichi Kageyama. Imagenet/resnet-50 training in 224 seconds. *arXiv preprint arXiv:1811.05233*, 2018.

Kamil Nar and S Shankar Sastry. Step size matters tep size matters in deep learning deep learning. In *NIPS*, 2018.

Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs. *arXiv preprint arXiv:1811.12019*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

Christopher J Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018a.

Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018b.

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

Leon Sixt, Benjamin Wild, and Tim Landgraf. Rendergan: Generating realistic labeled data. *Frontiers in Robotics and AI*, 5:66, 2018.

Daniel Soudry and Elad Hoffer. Exponentially vanishing sub-optimal local minima in multilayer neural networks. *arXiv preprint arXiv:1702.05777*, 2017.

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

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

Toan Tran, Trung Pham, Gustavo Carneiro, Lyle Palmer, and Ian Reid. A bayesian data augmentation approach for learning deep models. In *Advances in Neural Information Processing Systems*, pp. 2797–2806, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Vikas Verma, Alex Lamb, Christopher Beckham, Aaron Courville, Ioannis Mitliagkis, and Yoshua Bengio. Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer. *arXiv preprint arXiv:1806.05236*, 2018.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. ICML'13, pp. III–1058–III–1066. JMLR.org, 2013.

Lei Wu, Chao Ma, and Weinan E. How SGD Selects the Global Minima in Over-parameterized Learning : A Dynamical Stability Perspective. In *NeurIPS*, 2018.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised Data Augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992*, 2018.

Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017.

Komodakis Zagoruyko. Wide residual networks. In *BMVC*, 2016.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=r1Ddp1-Rb`.

APPENDIX

A HYPOTHESIS: LARGE BATCH TRAINING ISSUES

Previous works (Keskar et al., 2017; Nar & Sastry, 2018; Wu et al., 2018) suggested that large-batch training issues may result from an implicit bias in the SGD training process: with large batch sizes, SGD selects minima with worse generalization. We examine the dynamics of SGD to find how such a selection mechanism might work, and suggest why BA has less of these issues, in comparison to standard large batch.

Consider the optimization of non-augmented datasets, using loss functions of the form

$$f\left(\mathbf{w}\right)=\frac{1}{N}\sum_{n=1}^{N}\ell\left(\mathbf{w},\mathbf{x}_{n},\mathbf{y}_{n}\right)\ , \tag{1}$$

where we recall $\left\{\mathbf{x}_{n},\mathbf{y}_{n}\right\}_{n=1}^{N}$ is a dataset of N data sample-target pairs and ℓ is the loss function. We use SGD with batch of size B, where the update rule is given by

$$\mathbf{w}_{t+1}=\mathbf{w}_{t}-\eta\frac{1}{B}\sum_{n\in\mathcal{B}(k(t))}\nabla_{\mathbf{w}}\ell\left(\mathbf{w}_{t},\mathbf{x}_{n},\mathbf{y}_{n}\right)\ . \tag{2}$$

Here, we assume for simplicity that the indices are sampled with replacement, B divides N, and that $k\left(t\right)$ is sampled uniformly from $\{1,\dots,N/B\}$. When our model is sufficiently rich and over-parameterized (e.g., deep networks), we typically converge to a minimum $\mathbf{w}^{*}$ which is a global minimum on all data points in the training set (Zhang et al., 2017; Soudry & Hoffer, 2017). This means that $\forall n:\nabla_{\mathbf{w}}\ell\left(\mathbf{w}^{*},\mathbf{x}_{n},\mathbf{y}_{n}\right)=0$. We linearize the dynamics of Eq. 2 near $\mathbf{w}^{*}$ to obtain

$$\mathbf{w}_{t+1}=\mathbf{w}_{t}-\eta\frac{1}{B}\sum_{n\in\mathcal{B}(k(t))}\mathbf{H}_{n}\mathbf{w}_{t}\ , \tag{3}$$

where we assume (without loss of generality) that $\mathbf{w}^{*}=0$, and denote $\mathbf{H}_{n}\triangleq\nabla_{\mathbf{w}}^{2}\ell\left(\mathbf{w},\mathbf{x}_{n},\mathbf{y}_{n}\right)$ as the per-sample Hessian. Since we are at a global minimum, all $\mathbf{H}_{n}$ are symmetric PSD (there are no descent directions). However, recall that there can be many different global minima (on the training set). SGD selects only certain minima. As we shall see this selection depends on the batch sizes and learning rate, through the following quantities: the averaged Hessian over batch k

$$\left\langle\mathbf{H}\right\rangle_{k}\triangleq\frac{1}{B}\sum_{n\in\mathcal{B}(k)}\mathbf{H}_{n}$$

and the maximum over the maximal eigenvalues of $\{\left\langle\mathbf{H}\right\rangle_{k}\}_{k=1}^{N/B}$

$$\lambda_{\max}=\max_{k\in[N/B]}\max_{\forall\mathbf{v}:\|\mathbf{v}\|=1}\mathbf{v}^{\top}\left\langle\mathbf{H}\right\rangle_{k}\mathbf{v}. \tag{4}$$

This $\lambda_{\max}$ affects SGD through the following Theorem (proof in Appendix A.1):

Theorem 1 *The iterates of SGD (Eq. 3) will converge if*

$$\lambda_{\max}<\frac{2}{\eta}\,. \tag{5}$$

In addition, this bound is tight in the sense that it is also a necessary condition for certain datasets.

According to the Theorem, SGD with high learning rate will prefer to converge to minima with low $\lambda_{\max}$, thus selecting them from all (global) minima. Such minima, with low $\lambda_{\max}$, tend to have low variability of $\mathbf{H}_{n}$ (as high variability usually results in larger maximal values).

Next, when increasing the batch size, we typically *decrease* $\lambda_{\max}$, as we *decrease* the variability of $\left\langle\mathbf{H}\right\rangle_{k}$ and replace max operations with averaging. Therefore, certain minima with high variability in $\mathbf{H}_{n}$ will thus become accessible to SGD. Now SGD may converge to these high variability minima, which were suggested to exhibit worse generalization performance than the original minima (Wu et al., 2018).

This issue can be partially mitigated by increasing the learning rate Hoffer et al. (2017); Goyal et al. (2017), in a way which will make these new minima inaccessible again, while keeping the original minima accessible. However, merely changing the learning rate may not be sufficient for very large batch sizes, when some minima with high variability and low variability will eventually have similar $\lambda_{\max}$, so SGD will not be able to discriminate between these minima. For example, in the limit of full batch (GD), the variability of $\mathbf{H}_n$ will not affect $\lambda_{\max}$ (only their mean).

Now, recall BA can achieve variance reduction that is significantly lower than the $1/B$ reduction, which may occur with an uncorrelated sum of B samples. This implies that the $\lambda_{\max}$ (Eq. 5) would change less in BA than standard large-batch training, allowing the model to exhibit less of the aforementioned SGD convergence issues.

A.1 Proof of Theorem 1

We examine the first moment dynamics of Eq. 3, by taking its expectation

$$\mathbb{E}\mathbf{w}_{t+1} = \left(\mathbf{I} - \eta \left\langle \mathbf{H} \right\rangle\right) \mathbb{E}\mathbf{w}_t \,, \tag{6}$$

where

$$\left\langle \mathbf{H} \right\rangle \triangleq \frac{1}{N} \sum_{n=1}^{N} \mathbf{H}_n$$

it is easy to see that a necessary and sufficient condition for convergence of Eq. 6

$$\bar{\lambda}_{\max} < \frac{2}{\eta} \,, \tag{7}$$

where $\bar{\lambda}_{\max}$ is the maximal eigenvalue of $\left\langle \mathbf{H} \right\rangle$. This is the standard convergence condition for full batch SGD, i.e., gradient descent.

First, to see Eq. 5 is a necessary condition for certain datasets, suppose we have $\mathbf{H}_n = 0$ in all samples, except, in a single batch k, for which we have

$$\lambda_{\max} = \max_{\forall \mathbf{v}: \|\mathbf{v}\|=1} \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle_k \mathbf{v} \,,$$

In this case, the weights are updated only when we are at batch k. Therefore, ignoring all the batches, the dynamics are equivalent to full batch gradient descent with the dataset restricted to batch k. Therefore, $\bar{\lambda}_{\max} = \lambda_{\max}$, and we only have first order dynamics (with no noise). Thus, the necessary and sufficient condition for stability is Eq. 7 with $\bar{\lambda}_{\max} = \lambda_{\max}$, which is Eq. 5.

Next, to show Eq. 5 is also a sufficient condition (for all data sets) we examine the second moment dynamics. First we observe that

$$\begin{aligned}\mathbf{w}_{t+1}^\top \mathbf{w}_{t+1} &= \mathbf{w}_t^\top \left(\mathbf{I} - \eta \left\langle \mathbf{H} \right\rangle_{k(t)}\right)^\top \left(\mathbf{I} - \eta \left\langle \mathbf{H} \right\rangle_{k(t)}\right) \mathbf{w}_t \,. \\ &= \mathbf{w}_t^\top \left(\mathbf{I} - 2\eta \left\langle \mathbf{H} \right\rangle_{k(t)} + \eta^2 \left\langle \mathbf{H} \right\rangle_{k(t)} \left\langle \mathbf{H} \right\rangle_{k(t)}\right) \mathbf{w}_t \,.\end{aligned}$$

Denoting

$$\left\langle \mathbf{H}^2 \right\rangle \triangleq \frac{1}{N/B} \sum_{k=0}^{N/B} \left\langle \mathbf{H} \right\rangle_k \left\langle \mathbf{H} \right\rangle_k \,.$$

Thus, we obtain

$$\mathbb{E}\left\|\mathbf{w}_{t+1}\right\|^2 = \mathbb{E}\left[\mathbf{w}_{t+1}^\top \left(\mathbf{I} - 2\eta \left\langle \mathbf{H} \right\rangle + \eta^2 \left\langle \mathbf{H}^2 \right\rangle\right) \mathbf{w}_t\right] \,. \tag{8}$$

Since $\mathbf{H}_n$ are all PSDs it is easy to see that if $\mathbf{z}$ is a zero eigenvector of $\left\langle \mathbf{H} \right\rangle$ or $\left\langle \mathbf{H}^2 \right\rangle$ then it must be a zero vector eigenvector of other matrix, and also of all $\mathbf{H}_n$, $\forall n$. We denote the null space

$$\mathcal{V} \triangleq \left\{\mathbf{v} \in \mathbb{R}^d \middle| \, \|\mathbf{v}\| = 1, \left\langle \mathbf{H} \right\rangle \mathbf{z} = 0\right\}$$

and its complement $\bar{\mathcal{V}}$. From Eq. 8 a necessary and sufficient condition for convergence of this equation is

$$\max_{\mathbf{v}\in\bar{\mathcal{V}}} \mathbf{v}^\top \left(\mathbf{I} - 2\eta \left\langle \mathbf{H} \right\rangle + \eta^2 \left\langle \mathbf{H}^2 \right\rangle\right) \mathbf{v} < 1\,. \tag{9}$$

To complete the proof we will show that Eq. 5 also implies Eq. 9, for any B.

First we notice that Eq. 4 implies that $\forall \mathbf{v} \in \bar{\mathcal{V}}$:

$$\begin{aligned} \mathbf{v}^\top \left\langle \mathbf{H}^2 \right\rangle \mathbf{v} &= \frac{1}{N} \sum_{k=0}^{N/B} \sum_{n\in\mathcal{B}(k)} \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle_k \mathbf{H}_m \mathbf{v} \\ &\le \frac{1}{N} \sum_{n=1}^{N} \lambda_{\max} \mathbf{v}^\top \mathbf{H}_n \mathbf{v} \\ &= \lambda_{\max} \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle \mathbf{v}\,. \end{aligned} \tag{10}$$

Also, since $\lambda_{\max} > \bar{\lambda}_{\max}$, we have

$$\mathbf{v}^\top \left\langle \mathbf{H} \right\rangle^2 \mathbf{v} \le \lambda_{\max} \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle \mathbf{v}\,. \tag{11}$$

We combine the above results to prove the Lemma, and $\forall \mathbf{v} \in \bar{\mathcal{V}}$:

$$\begin{aligned} &\mathbf{v}^\top \left[(\mathbf{I} - 2\eta \left\langle \mathbf{H} \right\rangle) + \eta^2 \left\langle \mathbf{H}^2 \right\rangle \right] \mathbf{v} \\ =&1 - 2\eta \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle \mathbf{v} + \eta^2 \mathbf{v}^\top \left\langle \mathbf{H}^2 \right\rangle \mathbf{v} \\ \overset{(1)}{\le}&1 - 2\eta \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle \mathbf{v} + \eta^2 \lambda_{\max} \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle \mathbf{v} \\ =&1 - \eta \left(2 - \eta \lambda_{\max}\right) \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle \mathbf{v}\,, \end{aligned}$$

where in (1) we used Eqs. 10 and 11. Given the condition in Eq. 5 this is smaller than 1, so Eq. 9 holds, so this proves the Theorem.

As a side note, we can bound the convergence rate using the last equation. To see this, we denote $\mathcal{P}_{\bar{\mathcal{V}}}$ as the projection to $\bar{\mathcal{V}}$, and

$$\lambda_{\min} \triangleq \min_{\forall \mathbf{v}\in\bar{\mathcal{V}}} \mathbf{v}^\top \left\langle \mathbf{H} \right\rangle \mathbf{v}$$

as the smallest non-zero eigenvalue of $\left\langle \mathbf{H} \right\rangle$. iterating the recursion we obtain that the convergence rate is linear

$$\mathbb{E} \left\| \mathcal{P}_{\bar{\mathcal{V}}} \mathbf{w}_t \right\|^2 \le \left(1 - \eta \left(2 - \eta \lambda_{\max}\right) \lambda_{\min}\right)^t \mathbb{E} \left\| \mathcal{P}_{\bar{\mathcal{V}}} \mathbf{w}_0 \right\|^2\,. \tag{12}$$

However, note this bound is not necessarily tight.

B Image Throughput with Batch Augmentation

In Table 3, we use one NVIDIA P100 GPU and the parallel filesystem of a Cray supercomputer to train the ImageNet dataset on ResNet-50 over all feasible batch sizes (limited by the device memory). We list the median values over 200 experiments of images processed per second, as well as standard deviation. As expected, increasing the batch size starts by scaling nearly linearly (1.8× between 1 and 2 images per batch), but slows scaling as we reach device capacity, with only 5.7% utilization increase between batch sizes of 64 and 128. This indicates that, when using data parallelism in training, the local batch size should be increased as much as possible to maximize device utilization.

C Experiments Implementation Details

C.1 CIFAR

For CIFAR models under a large batch regime (Fixed Samples) we tried to verify that the gap from BA persists even under learning rate modification. We multiplied the original learning rate by a factor α and used a grid search following a logarithmic scale of 4 additional values $\alpha \in \{M^{0.25}, M^{0.5}, M, M^2\}$ where M is the batch-scaling factor. This choice was made to reflect the linear and sqrt learning rate rules suggested by Goyal et al. (2017) and Hoffer et al. (2017) respectively.

Table 3: ResNet-50 Image Throughput on ImageNet

Batch Size	Throughput [images/sec]	Standard Deviation
1	29.9	0.07
2	53.9	0.71
4	87.8	0.31
8	126.9	0.48
16	172.5	0.29
32	210.1	2.40
64	234.4	0.12
128	247.9	0.12

C.2 IMAGENET

For ResNet50 (He et al., 2016), we used the data augmentation method advocated by Szegedy et al. (2015) that employed various sized patches of the image with size distributed evenly between 8% and 100% and aspect ratio constrained to the interval $[3/4, 4/3]$. The images were also flipped horizontally with $p = 0.5$, and no additional color jitter was performed. For the MobileNet model (Howard et al., 2017), we used a less aggressive augmentation method, as described in the original paper. In the AlexNet model (Krizhevsky et al., 2012), we used the original augmentation regime.

For all ImageNet models, we followed the training regime by Goyal et al. (2017) in which an initial learning rate of 0.1 is decreased by a factor of 10 in epochs $30, 60$, and 80 for a total of 90 epochs. We applied a weight decay factor of 10^{-4} to every parameter in the network except for those of batch-norm layers.

The ResNet50 model was trained using multiple feed-forwards and gradient accumulations, creating a "Ghost batch normalization" (Hoffer et al., 2017) effect, where subsets of 32 images in the batch are normalized separately

C.3 TRANSFORMER – WMT

We used the base Transformer model by Vaswani et al. (2017) over WMT16 en-de task, along with the original hyper-parameters. We used our own implementation and trained the model for $100K$ iterations. Evaluation was performed without checkpoint averaging and beam-search of width 4. We used BA with $M = 4$ and a batch-size of 4096 tokens.

D DISTRIBUTED BATCH AUGMENTATION

We test our implementation on CSCS Piz Daint, a Cray XC50 supercomputer. Each XC50 compute node contains a 12-core HyperThreading-enabled Intel Xeon E5-2690 CPU with 64 GiB RAM, and one NVIDIA Tesla P100 GPU. The nodes communicate using a Cray Aries interconnect. The implementation uses decentralized (i.e., without a parameter server) synchronous SGD, and communication is performed using the Cray-optimized Message Passing Interface (MPI) v7.7.2. We use the maximal number of images per batch per-node, as it provides the best utilization (see Table 3).

In Figure 5, we plot the training runtime of two experiments on ImageNet with ResNet-50 for 40 epochs. We test with $B = 256$ $M = 4$ (16 nodes) and $M = 10$ (40 nodes), where each node processes a batch of 64 images. The plot shows that the difference in runtime for $M = 4$ and $M = 10$ is negligible, where the larger augmented batch consistently produces increased validation accuracy. The training process uses Ghost Batch Normalization (Hoffer et al., 2017) of 32 images and a standard, but shorter regime (i.e., without adding gradual warmup).

E RESULTS FOR REGULARIZATION IMPACT ON BA

Tables 4 and 5 show the validation accuracy of various networks using Mixup regularization Zhang et al. (2018) and Test-Time Augmentation (TTA), respectively.

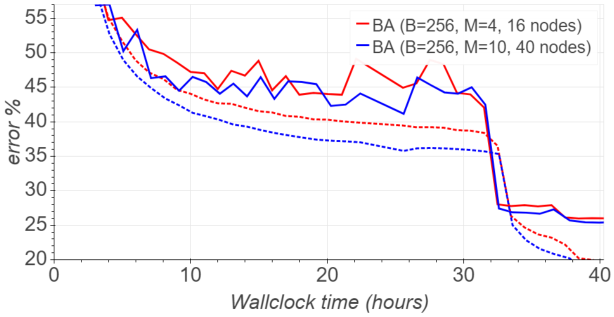


Figure 5: Training (dashed) and validation error over time (in hours) of ResNet50 with $B = 256$ and $M = 4$ (Red) vs $M = 10$ (Blue). Difference in runtime is negligible, while higher batch augmentation reaches lower error. Runtime for Baseline ($M = 1$): 1.43 ± 0.13 steps/second, $M = 4$: 1.47 ± 0.13 steps/second, $M = 10$: 1.46 ± 0.14 steps/second.

Table 4: Validation accuracy (Top1) results for Cifar, ImageNet models with mixup regularization ($\alpha = 0.2$). Training time was enlarged for ResNet44 to 200 epochs instead of 100 in previous results. ImageNet models was trained for 90 epochs.

Network	Dataset	M	Baseline	BA
ResNet44 He et al. (2016)	Cifar10	10	94.60%	**95.55%**
Wide-ResNet28-10 Zagoruyko (2016)	Cifar10	10	97.30%	**97.80%**
Wide-ResNet28-10	Cifar100	10	82.5%	**84.3%**
ResNet50 He et al. (2016)	ImageNet	4	76.70%	**77.04%**

Table 5: Validation accuracy (Top1) improvments of test-time-augmentation (TTA) vs a single-crop evaluation. Results for Cifar, ImageNet models under a 10-samples TTA. BA was trained with $M = 10$

		Single-Crop		TTA		Improvement	
Network	Dataset	Baseline	BA	Baseline	BA	Baseline	BA
ResNet44	Cifar10	93.41%	95.03%	94.33%	96.00%	13.96%	**19.52%**
DARTS	Cifar10	97.63%	97.85%	97.73%	98.13%	4.22%	**13.02%**
AmoebaNet (width=128)	Cifar10	98.16%	98.24%	98.15%	98.42%	-0.54%	**10.23%**
Wide-Resnet28-10	Cifar100	79.85%	83.45%	80.29%	84.88%	2.18%	**8.64%**
ResNet50	ImageNet	76.30%	77.85%	77.10%	77.9%	3.38%	**4.49%**