Scaling SGD Batch Size to 32K for ImageNet Training

Yang You [◇] **Igor Gitman** [*] **Boris Ginsburg**[†]
[◇] UC Berkeley [*] CMU [†] NVIDIA
youyang@cs.berkeley.edu igitman@andrew.cmu.edu bginsburg@nvidia.com

Abstract

The most natural way to speed-up the training of large networks is to use data-parallelism on multiple GPUs. To scale Stochastic Gradient (SG) based methods to more processors, one need to increase the batch size to make full use of the computational power of each GPU. However, keeping the accuracy of network with increase of batch size is not trivial. Currently, the state-of-the art method is to increase Learning Rate (LR) proportional to the batch size, and use special learning rate with "warm-up" policy to overcome initial optimization difficulty.

By controlling the LR during the training process, one can efficiently use large-batch in ImageNet training. For example, Batch-1024 for AlexNet and Batch-8192 for ResNet-50 are successful applications. However, for ImageNet-1k training, state-of-the-art AlexNet only scales the batch size to 1024 and ResNet50 only scales it to 8192. The reason is that we can not scale the learning rate to a large value. To enable large-batch training to general networks or datasets, we propose Layer-wise Adaptive Rate Scaling (LARS). LARS LR uses different LRs for different layers based on the norm of the weights ($||w||$) and the norm of the gradients ($||\nabla w||$). By using LARS algoirithm, we can scale the batch size to 32768 for ResNet50 and 8192 for AlexNet. Large batch can make full use of the system's computational power. For example, batch-4096 can achieve $3\times$ speedup over batch-512 for ImageNet training by AlexNet model on a DGX-1 station (8 P100 GPUs).

1 Introduction

Deep Neural Networks (DNN) perform significantly better than the traditional machine learning methods for the complicated applications like computer vision and natural language processing. However, the time-consuming DNN training process greatly limits the efficiency of DNN research and development. For example, using a eight-core CPU to train CIFAR-10 dataset needs 8.2 hours to achieve 0.8 accuracy. CIFAR-10 is only a 170 MB tiny dataset. Training ImageNet dataset [5] by AlexNet model [14] on one NVIDIA K20 GPUs will need 6 days to achieve 58% top-1 accuracy [10]. Scaling up and speeding up DNN training is highly important for the application of deep learning.

We focus on data-parallelism mini-batch Stochastic Gradient Descent (SGD) training [4], which is a state-of-the-art method used in many popular libraries like Caffe [11] and Tensorflow [1]. We use NVIDIA GPUs in our study. To speed up DNN training, we need to scale the algorithm to more computational processors. To scale the data-parallelism SGD method to more processors, we need to increase the batch size. Increasing the batch size as we increase the number of GPUs can keep the per-GPU work constant, which avoids reducing the singe GPU efficiency.

However, increasing batch size often leads to significant loss in test accuracy [7], [9], [12], [16]. Achieving the same accuracy with the baseline (e.g. batch size = 256) is the highest priority for large-batch training. By controlling the SGD learning rate (LR) during the training process, people can efficient large-batch trainning in a couple of notable applications. For example, Krizhevsky [13] used linear scaling LR rule in ImageNet training by AlexNet, which only lost 1 percent accuracy

when he increased the batch size from 128 to 1024. In ImageNet training by ResNet-152, Li [15] managed to achieve the same 77.8% accuracy when he increased the batch size from 256 to 5120 by linear scaling rule. Goyal et al [7] used linear scaling rule and warmup scheme in ImageNet training by ResNet-50. These methods helped Goyal et al to achieve 76.26% accuracy when they increased the learning rate from 256 to 8192. The accuracy of the baseline is 76.40%.

However, exsiting methods do not work when we increase the batch size beyond 1024 for using ImageNet dataset to train AlexNet. For example, by using linear scaling (or sqrt scaling) and warmup scheme, Batch-4096 only achieves 53.1% accuracy, which is 5.7% percent lower than Batch-512. Batch-8192 only achieves 44.8% accuracy, which is 14% percent lower than Batch-512. The target accuracy for using ImageNet dataset to train AlexNet is 58% in 100 epochs. To solve this problem, we tried many things such tune momentum, weight decay, minimum LR, and poly power. These tricks do not work for us. Then we add Batch Normalization to AlexNet model and we get AlexNet-BN model. AlexNet-BN improves Batch-4096's accuracy from 53.1% to 58.9% and Batch-8192's accuracy from 44.8% to 58.0%. Although AlexNet-BN achieves our target accuracy (58%) in 100 epochs, we still lost 1 percent accuracy by Batch-4096. We lost 2 percent accuracy by Batch-8192.

To further improve large-batch AlexNet's test accuracy and enable large-batch training to general networks or datasets, we propose Layer-wise Adaptive Rate Scaling (LARS) . LARS uses different LRs for different layers based on the norm of the weights ($||w||$) and the norm of the gradients ($||\nabla w||$). The reason behind this is that we observe that the ratio of weights and gradients ($||w||/||\nabla w||$) varies significantly for different layers. The optimal LR of large-ratio layer may lead to the divergence of small-ratio layer. By using LARS, we can achieve the same accuracy when we increase the batch size from 128 to 8192 for AlexNet model. For ResNet50 model, we successfully scaled the batch size to 32768 in ImageNet training.

Large batch can make full use the system's computational power. For example, batch-4096 can achieve 3$\times$ speedup over batch-512 for ImageNet training by AlexNet model on a DGX-1 station (8 P100 GPUs). All the accuracy mentioned in this paper means Top-1 test accuracy. The ImageNet dataset in this paper means ImageNet-1k. Because we did not use data augmentation, our ResNet-50 baseline's accuracy is lower than Goyal et al's baseline [7] (73% vs 76.4%).

2 Background and Related Work

2.1 Data-Parallelism Mini-Batch SGD

Let us refer to w as the DNN weights, X as the training data, n is the number of samples in X, and Y as the labels of X. Let us denote x_i as a sample of X and $l(x_i, w)$ as the loss computed by x_i and its lable y_i ($i \in \{1, 2, ..., n)$. Typically, people use the loss function like cross-entropy loss. The goal of DNN training is to minimize the loss function in Equation (1).

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} l(x_i, y_i, w) \tag{1}$$

At t-th iteration, we use forward and backward propagation to get the gradients of weights based on the loss. Then we use the gradients to update the weights, which is shown in Equation (2):

$$w_{t+1} = w_t - \eta \nabla l(x_i, y_i, w) \tag{2}$$

where η is the learning rate. This method is called as Stochastic Gradient Descent (SGD). Usually, people do not use a single sample to compute the loss and the gradients. they use a batch of samples at each iteration. Let us refer to the batch of sample at t-th iteration as B_t. The size of B_t is b. Then we update the weights based on Equation (3).

$$w_{t+1} = w_t - \frac{\eta}{b}\sum_{x \in B_t} \nabla l(x, y, w) \tag{3}$$

This method is called as Mini-Batch SGD. To simplify the notation, we ignore the momentum and weight decay. We denote the updating rule in Equation (4), which means that we use the gradients of weights ∇w_t to update the weights w_t.

$$w_{t+1} = w_t - \eta \nabla w_t \tag{4}$$

2.2 Large-Batch Training Difficulty

In order to improve the efficiency of data-parallelism DNN training when we have more computational processors, we need to increase the batch size. Increasing the batch size allows us to scale to more machines without reducing the workload on each processor. On modern computational intensive architecture like GPUs, reducing the workload often leads to a lower efficiency. However, when we increase the batch size after a certain point (e.g. 1024), the algorithm's convergence accuracy becomes significantly lower than the baseline (e.g. batch size=128) [7], [9], [12], [16].

Keskar et al [12] concluded that there is a generalization problem for large-batch training. It means that even the large batch can get a low training loss, the test loss will be still significant higher than the training loss. For small batch, the training loss and test loss are close to each other. Hoffer et al [9] and Li et al [16] suggests that training longer will help algorithm to generalize better and keep the accuracy. On the other hand, Goyal et al [7], Krizhevsky et al [13], and Li et al [15] use empirical study to show that controlling the learning rate in a smart way can help the large-batch training to get the same accuracy with the baseline.

2.3 Learning Rate (LR)

When we increase the batch size, we need to increase the base LR at the same time to prevent losing accuracy [7]. The are two rules of increasing the base LR:

Sqrt Scaling Rule [13]. When we increase the batch size by k, we should increase the LR by $\sqrt{k}$ to keep the variance in the gradient expectation constant.

Linear Scaling Rule [13]: When we increase the batch size by k, we should increase the LR by k based on the assumption that $\nabla l(x, y, w_t) \approx \nabla l(x, y, w_{t+j})$, where $j < b$.

Warmup Scheme [7] Usually, under linear scaling rule, $k\eta$ is exetremely large, which may make the algorithm diverge at the beginning. Therefore, people set the initial LR to a small value and increase it gradually to $k\eta$ in a few epochs (e.g. 5 or 10). This method is called as **Gradual Warmup Scheme**. There is another method called as **Constant Warmup Scheme**, which uses a constant small LR during the first a couple of epochs. Constant warmup scheme works efficiently for prototyping object detection and segmentation [6], [17]. Goyal et al [7] showed that gradual warmup performs better than constant warmup for ResNet-50 training.

Bottou et al [2] showed that there should be a upper bound of LR regardless of batch size. Our experimental results are in line with this findings. Chen et al [3] also used linear scaling LR in their experiments when they increase the batch size from 1600 to 6400. However, they did not show the accuracy of the small-batch baseline.

2.4 State-of-the-art Large Batch Traing

Krizhevsky [13] reported 1 percent loss in accuracy when he increased the the batch size from 128 to 1024. He achevied 56.7% accuracy for using batch-1024 Alexnet to train Imagenet dataset. Li [15] used batch-5120 ResNet-152 to train Imagenet dataset on 160 GPUs. The batch size on each GPU is 32, and they use 8 GPUs (batch size=256) as the baseline. The baseline achieves 77.8% test accuracy by 110 epochs. The base LR for batch 256 is 0.1, then they reduce the LR by a factor of 10 at 30th epoch, 60th epoch, and 90th epoch. When they increase the batch size to 5120 (160 GPUs), they set the base LR as 1 (not exactaly the linear scaling rule or the sqrt scaling rule). For batch-5120 training, they reduce the LR by factor of 10 at 50th epoch and 100th epoch. In Li's report, batch-5120 achieves the same final accuracy with batch-256. Goyal et al [7] implemented the large-batch ResNet-50 to speed up ImageNet training. They used data parallelism to process ResNet-50 model on 256 NVIDIA P100 GPUs (equal to 32 NVIDIA DGX-1 stations). The methods they used are: (1) Linear Scaling for Base LR, (2) Gradual Warmup Scheme for LR, and (3) Multi-step LR rule. The baseline batch size is 256, which achieves 76.40% Top-1 accuracy. The large-batch size is 8192, which achieves 76.26% Top-1 accuracy.

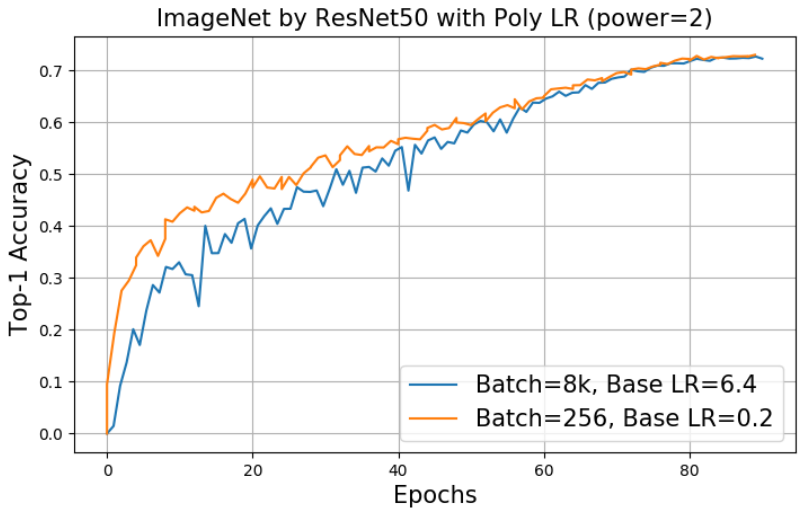


Figure 1: From this figure, we clearly observe that we can use 8k batch size to achieve the same accuracy with 256 batch size by using the same 90 epochs. This experiment was conducted on one NVIDIA DGX-1 station.

Table 1: ImageNet Dataset by ResNet50 Model with poly learning rate (LR) rule.

Batch Size	Base LR	power	momentum	weight decay	Epochs	Peak Test Accuracy
256	0.2	2.0	0.9	0.0001	90	0.730
8192	6.4	2.0	0.9	0.0001	90	0.727

3 ImageNet-1k Training

3.1 Reproduce and Extend Facebook's result

Our first step is to reproduce and extend Facebook's result [7]. Facebook used multistep learning rate rule, warm-up strategy, and linear-scaling learning rate. Specifically, the batch-8192's base LR is 3.2, which is 32 times (8192/256) of batch-256 base LR. During the first five epochs, they gradually increase the learning rate from 0.1 to 3.2, which is called as the warm up range. At 30th epoch, 60th epoch, and 80th epoch, they use $\eta = 0.1 \times \eta$ to update the learning rate.

Like Facebook's paper, we use the warm-up and linear-scaling stargegies for the learning rate. There are two differences between us and Facebook: (1) We push the learning rate higher. The base learning rate for batch-256 case is 0.2 and the base learning rate for batch-8192 is 6.4. (2) We use the poly rule rather than multistep rule to update the learning rate. The poly power is 2 in our setting. From Table 1, we observe that our 8k-batch test accuracy is close to the 256-batch peak test accuracy (0.727 vs 0.730). Moreover, our batch-8192 test accuracy curve is identical to the batch-256 test accuracy curve during the final 30 epochs (Figure 1).

3.2 Train ImageNet by AlexNet

3.2.1 Linear Scaling and Warmup schemes for LR

We use the Batch-512 BVLC-AlexNet[1] as our baseline, which achieves 0.58 accuracy by 100 epochs in our experiments. We use the poly LR rule. The Base LR is 0.01 and the poly power is 2. Our target is to use Batch-4096 and Batch-8192 to achieve 0.58 accuracy in 100 epochs. Firstly, we use linear scaling for Batch-4096 AlexNet and we should set the Base LR as 0.08. However, Batch-4096 is not converged even at LR = 0.01.

[1] https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

Table 2: ImageNet dataset by AlexNet model with ploy LR. We use the linear scaling and warmup scheme for LR, which are the major techniques used in Facebook's paper [7].

Batch Size	Base LR	power	warmup LR	warmup range	epochs	test accuracy
512	0.02	2	N/A	N/A	100	**0.588**
4096	0.01	2	N/A	N/A	100	0.001
4096	0.01	2	0.001	2.5 epochs	100	0.510
4096	0.02	2	0.001	2.5 epochs	100	0.527
4096	0.03	2	0.001	2.5 epochs	100	0.520
4096	0.04	2	0.001	2.5 epochs	100	0.530
4096	0.05	2	0.001	2.5 epochs	100	**0.531**
4096	0.06	2	0.001	2.5 epochs	100	0.516
4096	0.07	2	0.001	2.5 epochs	100	0.001
8192	0.02	2	0.001	6 epochs	100	0.298
8192	0.03	2	0.001	6 epochs	100	0.448
8192	0.04	2	0.001	6 epochs	100	0.431
8192	0.05	2	0.001	6 epochs	100	0.001

Then we use both the linear scaling and the warmup schemes to Batch-4096 AlexNet. After a seriers of hyper-parameter tunings, we set the warmup LR as 0.001 and warmup range as 2.5 epochs for Batch-4096. Based on the linear scaling scheme, we tune the Base LR from 0.01 to 0.08. Batch-4096 achieves the highest accuracy at LR=0.05. However, the highest accuracy is only 0.531, which is lower than our 0.58 target. Batch-4096 diverged at LR=0.07. We set the warmup LR as 0.001 and warmup range as 6 epochs for Batch-8192. Based on the linear scaling scheme, we tune the Base LR from 0.02 to 0.16. Batch-8192 achieves the highest accuracy at LR=0.03. The highest accuracy is only 0.448, which is much lower than our 0.58 target. Batch-8192's accuracy is even much lower than Batch-4096's accuracy. Batch-8192 diverged at LR=0.05. These two cases are the counter examples of Goyal et al's recipe [7]. We found that only using linear scaling and warmup schemes are not enough for large-batch AlexNet training.

3.2.2 Batch Normalization for Large-Batch Training

To enable large-batch training for AlexNet, we tried many different techniques (e.g. data shuffle, data scaling, multi-step LR, min LR tuning). We observed that only Batch Normalization (BN) improves the accuracy. Let us refer to this version as AlexNet-BN, which is available online[2]. We run 128 epochs for all the cases after the accuracy is stable. We start the base LR from 0.01 and increase the base LR until the algorithm is diverged. For Batch-4096, we stopped at LR=0.30 because the accuracy is significantly lower than the baseline. For Batch-8192, we stopped at LR=0.41 because the accuracy is significantly lower than the baseline. The resuls are shown in Table 3.

Although we achieve a higher accuracy by adding batch normalization, there is still a 1 percent accuracy gap between batch 512 and batch 4096. For batch 8192, the gap is even larger, which is 2 percent. Thus, we still need to improve the accuracy. We tuned the hype-parameters like momentum (0.85-0.95) and weight decay (0.0001-0.0008). However, they did not improve the accuracy. Keskar et al [12] concluded that there is a generalization problem for large-batch training. It means that even the large batch can get a low training loss, the test loss will be still significant higher than the training loss. For small batch, the training loss and test loss are close to each other.

After adding batch normalization to the network, we observe that we can solve the generalization problem better. After this, we think the reason why large batch does not work well is not mainly because of the generalization problem. The reason is because of optimization difficulty. From Figure 2, we clearly observe that the gap between training loss and test loss in large batch is small (especially in the final 30 epochs).

[2]https://github.com/borisgin/nvcaffe-0.16/tree/caffe-0.16/models/alexnet_bn

Table 3: ImageNet dataset by AlexNet-BN model with ploy LR.

Batch Size	Base LR	power	momentum	weight decay	epochs	test accuracy
512	0.02	2	0.9	0.0005	128	**0.602**
4096	0.02	2	0.9	0.0005	128	0.540
4096	0.16	2	0.9	0.0005	128	0.581
4096	0.18	2	0.9	0.0005	128	**0.589**
4096	0.21	2	0.9	0.0005	128	0.585
4096	0.25	2	0.9	0.0005	128	0.583
4096	0.30	2	0.9	0.0005	128	0.571
8192	0.23	2	0.9	0.0005	128	0.576
8192	0.30	2	0.9	0.0005	128	**0.580**
8192	0.32	2	0.9	0.0005	128	0.577
8192	0.41	2	0.9	0.0005	128	0.565

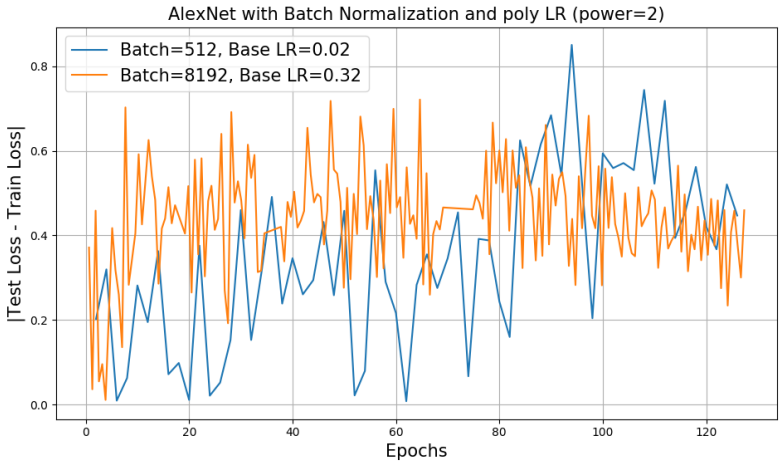


Figure 2: From this figure, we clearly observe that the generalization situation becomes better after we add batch normalization.

3.2.3 Layer-wise Adaptive Rate Scaling (LARS) for Large-Batch Training

To improve the accuracy for large-batch AlexNet, we designed a new rule of updating LR. As mentioned before, we use $w = w - \eta \nabla w$ to update the weights. Each layer has its own weight w and gradient ∇w. Standard SGD algorithm uses the same LR (η) for all the layers. However, from our experiments, we observe that different layers may need different LRs. The reason is that the ratio between $||w||_2$ and $||\nabla w||_2$ varies significantly for different layers. From example, we observe that $||w||_2/||\nabla w||_2$ is only 5.76 for conv1.0 layer. However, $||w||_2/||\nabla w||_2$ is 1345 for fc6.0 layer. To speedup the DNN training, the users need to use a large LR for fc6.0 layer. However, this large LR may lead to divergence of conv1.0 layer. We think this is an important reason of the optimization difficulty in large-batch training. Goyal et al [7] proposed the warmup scheme to solve this problem. The warmup scheme works well for ResNet50 training (for batch size less than 8192). However, only using this recipe does not work for AlexNet training.

In this paper, we design LARS learning rate scheme to improve large-batch training's accuracy. The base LR rule is defined in Equation (5). l is the scaling factor, we set it as 0.001 for AlexNet and ResNet training. γ is the input LR, which is a tuning parameter for users. We usually tune γ from 1 to 50. In this way, different layers will have different LR and SGD will become more stable. In practice, we add momentum and weight decay to SGD. Let us use μ to denote momentum and β to denote weight decay. We use the following sequence for LARS:

(1) we get the local learning rate for each learnable parameter by $\alpha = l \times ||w||_2/(||\nabla w||_2 + \beta||\nabla w||_2)$;

(2) we get the true learning rate for each layer by $\eta = \gamma \times \alpha$;

Table 4: The L2 norm of layer weights and gradients for using AlexNet to train ImageNet dataset. The data is collected at 1st iteration. The batch size is 4096. conv means convolutional layer, and means fully-connected layer. x.0 means the layer weight, x.1 means the layer bias.

Layer	conv1.1	conv1.0	conv2.1	conv2.0	conv3.1	conv3.0	conv4.0	conv4.1
$\|\|w\|\|_2$	1.86	0.098	5.546	0.16	9.40	0.196	8.15	0.196
$\|\|\nabla w\|\|_2$	0.22	0.017	0.165	0.002	0.135	0.0015	0.109	0.0013
$\frac{\|\|w\|\|_2}{\|\|\nabla w\|\|_2}$	8.48	**5.76**	33.6	83.5	69.9	127	74.6	148
Layer	conv5.1	conv5.0	fc6.1	fc6.0	fc7.1	fc7.0	fc8.1	fc8.0
$\|\|w\|\|_2$	6.65	0.16	30.7	6.4	20.5	6.4	20.2	0.316
$\|\|\nabla w\|\|_2$	0.09	0.0002	0.26	0.005	0.30	0.013	0.22	0.016
$\frac{\|\|w\|\|_2}{\|\|\nabla w\|\|_2}$	73.6	69	117	**1345**	68	489	93	19

Table 5: ImageNet Dataset by AlexNet_BN Model with LARS LR and ploy.

Batch Size	poly power	γ	warmup	weight decay	momentum	Epochs	test accuracy
512	2	5	2 epochs	0.0005	0.9	100	0.602
4096	2	10	2 epochs	0.0005	0.9	100	0.604
8192	2	14	2 epochs	0.0005	0.9	100	0.601

(3) we update the gradients by $\nabla w = \nabla w + \beta w$;

(4) we update acceleration term a by $a = \mu a + \eta \nabla w$;

(4) we update the weights by $w = w - a$.

After using LARS, we clearly observe that large-batch can achieve the same accuracy with the baseline (Table 5). Now, we remove the batch normalization and use the original AlexNet model. We use LARS LR for batch-4096 and batch-8192 on AlexNet-ImageNet training. The warmup range for batch-4096 is 13 epochs. The warmup range for batch-8192 is 8 epochs. The large batch can achieve the same accuracy with the baseline (Table 6). From Figure 3, we can clearly observe the effects of LARS. LARS can help us to preserve the high test accuracy. This figure also shows that only using Batch Normalizaiton is not enough. If we use Batch Normalizaiton and LARS together, the result becomes much better.

$$\eta = l \times \gamma \times \frac{||w||_2}{||\nabla w||_2} \tag{5}$$

4 Experimental Results

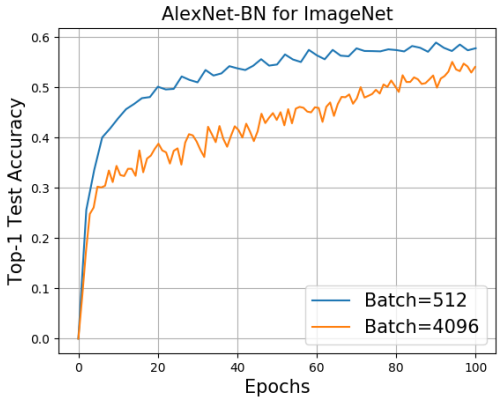
4.1 Systems and Libraries

We use a version of NVIDIA Caffe with our own modification, which is available online[3]. Our experimental system is a NVIDIA DGX-1 station, which includes one Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz and 8 NVIDIA P100 GPUs that are interconnected by NVIDIA NVLink. Our system has 3.2 TB of NVNs SSDs.

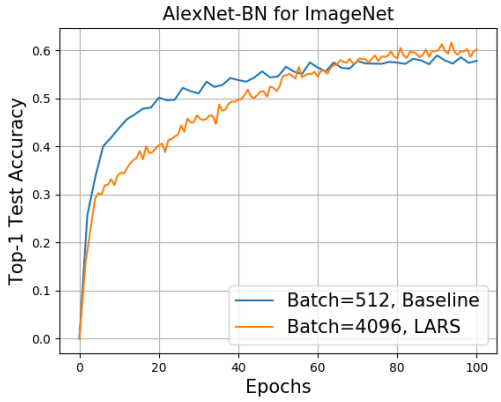
[3]https://github.com/borisgin/nvcaffe-0.16

Table 6: ImageNet Dataset by AlexNet Model.

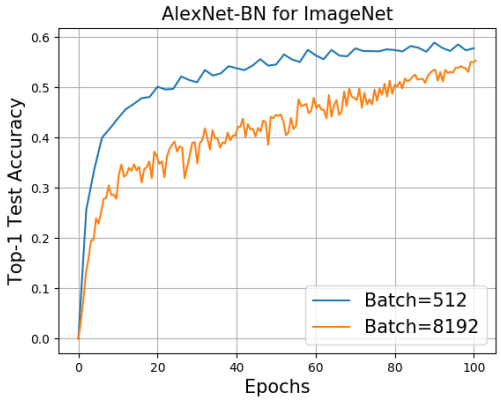
Batch Size	LR rule	power	warmup	weight decay	momentum	Epochs	test accuracy
512	regular + poly	2	N/A	0.0005	0.9	100	0.588
4096	LARS + poly	2	13 epochs	0.0005	0.9	100	0.584
8192	LARS +poly	2	8 epochs	0.0005	0.9	100	0.583



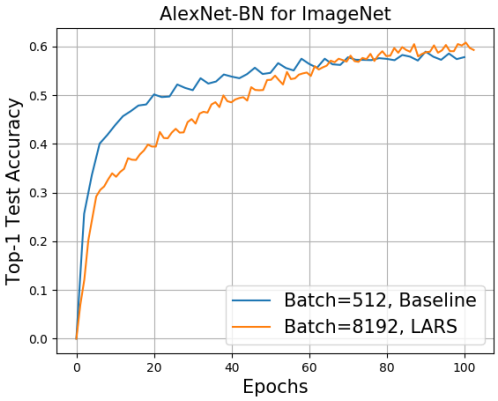
(a) Without LARS, Batch 4096



(b) With LARS, Batch 4096



(c) Without LARS, Batch 8192



(d) With LARS, Batch 8192

Figure 3: All cases only run 100 epochs. We can clearly observe the effects of LARS LR.

4.2 Implementation Details

We do not have enough memory to hold batch size $\geq$ 8192 for ImageNet training by ResNet-50 on our system. Thus, we partition the 8192-batch into 32 256-batches, and then computed these 32 pieces of gradients sequentially. We store these 32 pieces of gradients into memory and conduct an average operation after we finish computing all the gradients. We use the same method for processing batch 16k and batch 32k. In many Caffe versions, users can use $iter\_size$ to implement this function. For the small dataset like CIFAR-10, in order to use the computational resources of each GPU, we use multiple solvers on each GPU. For batch 1024 CIFAR-10 training, 8 solvers on 4 GPUs can achieve 1.4$\times$ speedup over 4 solvers on 4 GPUs. The multi-solver implementation is available online[4].

4.3 State-of-the-art Results for ResNet50 training

By using LARS, we can scale the batch size up to 32K for ImageNet-1K training by ResNet50 model (Figures 4 and 5, Table 7). We use Batch 256 as the baseline. For peak test accuracy, Batch 8K lost 0.3% accuracy and Batch 32K lost 0.4% accuracy. We believe these accuracy gaps can be eliminated by hyper-parameters tuning. Our baseline is lower than state-of-the-art results [7] [8] because we did not use data augmentation. For the versions without data augmentation, our baseline achieves the state-of-the-art accuracy. In our experiments, we observe that adding data augmentation can improve 3% - 4% accuracy. However, because data augmentation is not easy to reproduce, we only report the version without data augmentation in this paper. Goyal et al. [7] scaled the batch size to 8192

[4]https://github.com/drnikolaev/nvcaffe-dev/tree/caffe-0.16-multisolver-2

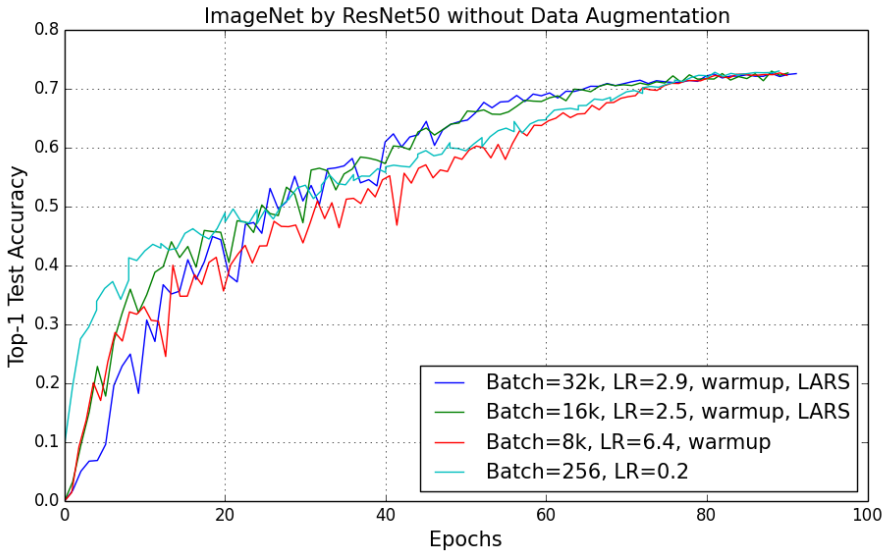
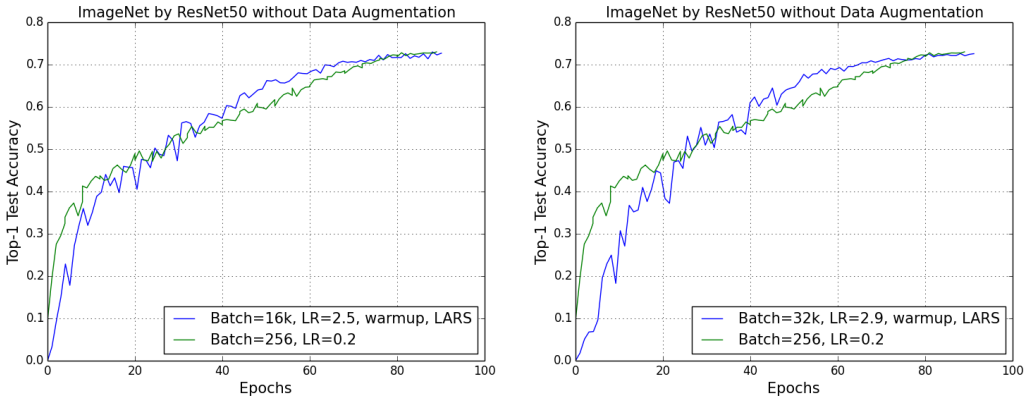


Figure 4: By using LARS, we can scale the batch size of ResNet50 to 32K.



(a) Batch 16K vs Batch 256

(b) Batch 32K vs Batch 256

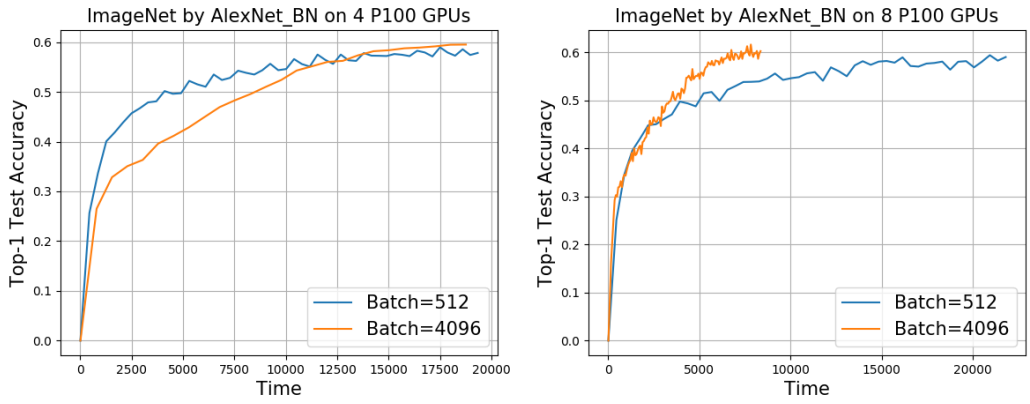
Figure 5: By using LARS, we can scale the batch size of ResNet50 to 32K.

while we successfully scaled the batch size to 32768. To encourage others to reproduce our results, we share our Caffe log files online[5].

[5]https://people.eecs.berkeley.edu/∼youyang/publications/batch

Table 7: ImageNet Dataset by ResNet50 Model.

Batch Size	LR rule	Base LR	power	warmup	Epochs	peak test accuracy
256	regular + poly	0.2	2	N/A	90	0.730
8192	LARS + poly	6.4	2	5 epochs	90	0.727
16384	LARS +poly	2.5	2	5 epochs	90	0.730
32768	LARS +poly	2.9	2	5 epochs	90	0.726



(a) The Same Speed

(b) Large Batch is faster

Figure 6: On a half of DGX-1 station (4 GPUs), Batch 512 has the same speed with Batch 4096. On a DGX-1 station (8 GPUs), Batch 4096 is about 3× faster than Batch 512. Thus, large batch size can benefit from computational power enhancement.

Table 8: The speed and time for AlexNet-BN.

Batch Size	Stable Accuracy	4-GPU speed	4-GPU time	8-GPU speed	8-GPU time
512	0.602	6627 img/sec	5h 22m 30s	5771 img/sec	6h 10m 30s
4096	0.604	6585 img/sec	5h 24m 44s	15379 img/sec	2h 19m 24s

4.4 Benefits of Using Large Batch

After getting the same accuracy for baseline and large batch, we focus on the speed comparison, which is our initial goal of large-batch training. We use the results of ImageNet training by AlexNet-BN and AlexNet to illustrate the benefits of large batch. Let us have a look at Figure 6, Table 8, and Table 9. For AlexNet-BN training, we observe that Batch-512 and Batch-4096 roughly have the same speed on 4 GPUs. However, when we move the code to 8 GPUs, we observe that Batch-4096 achieves 3× speedup over Batch-512. In fact, Batch-512 on 8 GPUs is even slower than Batch-512 on 4 GPUs. The reason is that Batch-512 can not make use all the computational resources of 4 GPUs (Batch 128 per GPU). The GPU efficiency becomes even lower for Batch-512 on 8 GPUs (Batch 64 per GPU). On the other hand, the computational resources are not enough for Batch-4096 on 4 GPUs (1024 per GPU). The situation becomes better when we provide 8 GPUs for Batch-4096 training (512 per GPU). ImageNet training by AlexNet achieves similar results (Tables 10 and 11). Batch-4096 achieves 3× speedup over Batch-512 for AlexNet-ImageNet training on the same 8 GPUs.

5 Conclusion

The optimization difficulty leads to the accuracy loss for large-batch training. Only using existing methods such as linear scaling and warmup scheme are not enough to complicated application like using AlexNet to train ImageNet. Changing the network structure such adding batch normalization can help improve the accuracy. However, only adding batch normalization is not enough. We proposed Layer-wise Adaptive Rate Scaling (LARS) , which uses different LRs for different layers based on the norm of their weights and the norm of their gradient. LARS is shown highly efficient in our experiments. By using LARS, we can achieve the same accuracy when we increase the batch size

Table 9: The speedup of AlexNet-BN training. B4096/4-GPU: Batch 4096 on 1 CPU and 4 GPUs

B4096/4-GPU over B512/4-GPU	B4096/8-GPU over B512/8-GPU	B512/8-GPU over B512/4-GPU	B4096/8-GPU over B4096/4-GPU
1	2.7	0.87	2.3

Table 10: The speed and time for AlexNet.

Batch Size	Stable Accuracy	4-GPU speed	4-GPU time	8-GPU speed	8-GPU time
512	0.588	6780 img/sec	5h 15m 15s	5797 img/sec	6h 9m 0s
4096	0.584	5172 img/sec	6h 52m 55s	16373 img/sec	2h 10m 52s

Table 11: The speedup of AlexNet training. B4096/4-GPU: Batch 4096 on 1 CPU and 4 GPUs

B4096/4-GPU over B512/4-GPU	B4096/8-GPU over B512/8-GPU	B512/8-GPU over B512/4-GPU	B4096/8-GPU over B4096/4-GPU
0.76	2.8	0.86	3.2

from 128 to 8192 for AlexNet model in ImageNet training. We can also scale the batch size to 32768 for ResNet50 model in ImageNet training. Large batch can make full use the system's computational power. For example, batch-4096 can achieve $3\times$ speedup over batch-512 for ImageNet training by AlexNet model on a DGX-1 station (8 P100 GPUs).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

[3] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[4] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

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

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

[9] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv preprint arXiv:1705.08741*, 2017.

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

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

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

[13] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

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

[15] M. Li. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, Intel, 2017.

[16] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[17] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144*, 2016.