LARGE BATCH TRAINING OF CONVOLUTIONAL NETWORKS

Yang You *
Computer Science Division
University of California at Berkeley
youyang@cs.berkeley.edu

Igor Gitman
Computer Science Department
Carnegie Mellon University
igitman@andrew.cmu.edu

Boris Ginsburg
NVIDIA
bginsburg@nvidia.com

ABSTRACT

A common way to speed up training of large convolutional networks is to add computational units. Training is then performed using data-parallel synchronous Stochastic Gradient Descent (SGD) with mini-batch divided between computational units. With an increase in the number of nodes, the batch size grows. But training with large batch size often results in the lower model accuracy. We argue that the current recipe for large batch training (linear learning rate scaling with warm-up) is not general enough and training may diverge. To overcome this optimization difficulties we propose a new training algorithm based on Layer-wise Adaptive Rate Scaling (LARS). Using LARS, we scaled Alexnet up to a batch size of 8K, and Resnet-50 to a batch size of 32K without loss in accuracy.

Large batch sizes converge to sharp minima

1 INTRODUCTION

Training of large Convolutional Neural Networks (CNN) takes a lot of time. The brute-force way to speed up CNN training is to add more computational power (e.g. more GPU nodes) and train network using data-parallel Stochastic Gradient Descent, where each worker receives some chunk of global mini-batch (see e.g. Krizhevsky (2014) or Goyal et al. (2017)). The size of a chunk should be large enough to utilize the computational resources of the worker. So scaling up the number of workers results in the increase of batch size. But using large batch may negatively impact the model accuracy, as was observed in Krizhevsky (2014), Li et al. (2014), Keskar et al. (2016), Hoffer et al. (2017),..

Increasing the global batch while keeping the same number of epochs means that you have fewer iterations to update weights. The straight-forward way to compensate for a smaller number of iterations is to do larger steps by increasing the learning rate (LR). For example, Krizhevsky (2014) suggests to linearly scale up LR with batch size. However using a larger LR makes optimization more difficult, and networks may diverge especially during the initial phase. To overcome this difficulty, Goyal et al. (2017) suggested doing a "learning rate warm-up": training starts with a small "safe" LR, which is slowly increased to the target "base" LR. With a LR warm-up and a linear scaling rule, Goyal et al. (2017) successfully trained Resnet-50 with batch B=8K (see also Cho et al. (2017)). Linear scaling of LR with a warm-up is the "state-of-the art" recipe for large batch training.

We tried to apply this linear scaling and warm-up scheme to train Alexnet on Imagenet (Deng et al. (2009)), but scaling stopped after B=2K since training diverged for large LR-s. For B=4K the accuracy dropped from the baseline 57.6% (for B=256) to 53.1%, and for B=8K the accuracy decreased to 44.8%. To enable training with a large LR, we replaced Local Response Normalization layers in Alexnet with Batch Normalization (BN). We will refer to this modification of AlexNet as AlexNet-BN throughout the rest of the paper. BN improved both model convergence for large LR as well as accuracy: for B=8K the accuracy gap was decreased from 14% to 2.2%.

*Work was performed when Y.You and I.Gitman were NVIDIA interns

To analyze the training stability with large LRs we measured the ratio between the norm of the layer weights and norm of gradients update. We observed that if this ratio is too high, the training may become unstable. On other hand, if the ratio is too small, then weights don't change fast enough. This ratio varies a lot between different layers, which makes it necessary to use a separate LR for each layer. Thus we propose a novel Layer-wise Adaptive Rate Scaling (LARS) algorithm. There are two notable differences between LARS and other adaptive algorithms such as ADAM (Kingma & Ba (2014)) or RMSProp (Tieleman & Hinton (2012)): first, LARS uses a separate learning rate for each layer and not for each weight, which leads to better stability. And second, the magnitude of the update is controlled with respect to the weight norm for better control of training speed. With LARS we trained Alexnet-BN and Resnet-50 with B=32K without accuracy loss.

2 BACKGROUND

The training of CNN is done using Stochastic Gradient (SG) based methods. At each step t a mini-batch of B samples x_i is selected from the training set. The gradients of loss function $\nabla L(x_i, w)$ are computed for this subset, and networks weights w are updated based on this stochastic gradient:

$$w_{t+1} = w_t - \lambda \frac{1}{B} \sum_{i=1}^{B} \nabla L(x_i, w_t) \tag{1}$$

The computation of SG can be done in parallel by N units, where each unit processes a chunk of the mini-batch with $\frac{B}{N}$ samples. Increasing the mini-batch permits scaling to more nodes without reducing the workload on each unit. However, it was observed that training with a large batch is difficult. To maintain the network accuracy, it is necessary to carefully adjust training hyper-parameters (learning rate, momentum etc).

Krizhevsky (2014) suggested the following rules for training with large batches: when you increase the batch B by k, you should also increase LR by k while keeping other hyper-parameters (momentum, weight decay, etc) unchanged. The logic behind **linear LR scaling** is straight-forward: if you increase B by k while keeping the number of epochs unchanged, you will do k fewer steps. So it seems natural to increase the step size by k. For example, let's take $k = 2$. The weight updates for batch size B after 2 iterations would be:

$$w_{t+2} = w_t - \lambda * \frac{1}{B} \left(\sum_{i=1}^{B} \nabla L(x_i, w_t) + \sum_{j=1}^{B} \nabla L(x_j, w_{t+1}) \right) \tag{2}$$

The weight update for the batch $B_2 = 2 * B$ with learning rate λ_2:

$$w_{t+1} = w_t - \lambda_2 * \frac{1}{2 * B} \sum_{i=1}^{2B} \nabla L(x_i, w_t) \tag{3}$$

will be similar if you take $\lambda_2 = 2 * \lambda$, assuming that $\nabla L(x_j, w_{t+1}) \approx L(x_j, w_t)$.

Using the "linear LR scaling" Krizhevsky (2014) trained AlexNet with batch B=1K with minor ($\approx 1\%$) accuracy loss. The scaling of Alexnet above 2K is difficult, since the training diverges for larger LRs. It was observed that linear scaling works much better for networks with Batch Normalization (e.g. Codreanu et al. (2017)). For example Chen et al. (2016) trained the Inception model with batch B=6400, and Li (2017) trained Resnet-152 for B=5K.

The main obstacle for scaling up batch is the instability of training with high LR. Hoffer et al. (2017) tried to use less aggressive "square root scaling" of LR with special form of Batch Normalization ("Ghost Batch Normalization") to train Alexnet with B=8K, but still the accuracy (53.93%) was much worse than baseline 58%. To overcome the instability during initial phase, Goyal et al. (2017) proposed to use **LR warm-up**: training starts with small LR, and then LR is gradually increased to the target. After the warm-up period (usually a few epochs), you switch to the regular LR policy ("multi-steps", polynomial decay etc). Using LR warm-up and linear scaling Goyal et al. (2017) trained Resnet-50 with batch B=8K without loss in accuracy. These recipes constitute the current state-of-the-art for large batch training, and we used them as the starting point of our experiments

Another problem related to large batch training is so called "generalization gap", observed by Keskar et al. (2016). They came to conclusion that "the lack of generalization ability is due to the fact that large-batch methods tend to converge to sharp minimizers of the training function." They tried a few methods to improve the generalization with data augmentation and warm-starting with small batch, but they did not find a working solution.

3 ANALYSIS OF ALEXNET TRAINING WITH LARGE BATCH

We used BVLC[1] Alexnet with batch B=512 as baseline. Model was trained using SGD with momentum 0.9 with initial LR=0.01 and the polynomial (power=2) decay LR policy for 100 epochs. The baseline accuracy is 58% (averaged over last 5 epochs). Next we tried to train Alexnet with B=4K by using larger LR. In our experiments we changed the base LR from 0.01 to 0.08, but training diverged with LR > 0.06 even with warm-up [2]. The best accuracy for B=4K is 53.1%, achieved for LR=0.05. For B=8K we couldn't scale-up LR either, and the best accuracy is 44.8% , achieved for LR=0.03 (see Table 1(a)).

To stabilize the initial training phase we replaced Local Response Normalization layers with Batch Normalization (BN). We will refer to this model as Alexnet-BN [3]. The baseline accuracy for Alexnet-BN with B=512 is 60.2%. [4] With BN we could use large LR-s even without warm-up. For B=4K the best accuracy 58.9% was achieved for LR=0.18, and for B=8K the best accuracy 58% was achieved for LR=0.3. We also observed that BN significantly widen the range of LRs with good accuracy.

Table 1: Alexnet and Alexnet-BN: B=4K and 8K. BN makes it possible to use larger learning rates.

(a) Alexnet (warm-up 2.5 epochs)

Batch	Base LR	accuracy,%
512	**0.02**	**58.8**
4096	0.04	53.0
4096	**0.05**	**53.1**
4096	0.06	51.6
4096	0.07	0.1
8192	0.02	29.8
8192	**0.03**	**44.8**
8192	0.04	43.1
8192	0.05	0.1

(b) Alexnet-BN (no warm-up)

Batch	Base LR	accuracy,%
512	**0.02**	**60.2**
4096	0.16	58.1
4096	**0.18**	**58.9**
4096	0.21	58.5
4096	0.30	57.1
8192	0.23	57.6
8192	**0.30**	**58.0**
8192	0.32	57.7
8192	0.41	56.5

Still there is a 2.2% accuracy loss for B=8K. To check if it is related to the "generalization gap" (Keskar et al. (2016)), we looked at the loss gap between training and testing (see Fig. 1). We did not find the significant difference in the loss gap between B=256 and B=8K. We conclude that in this case the accuracy loss is not related to a generalization gap, and it is caused by the low training.

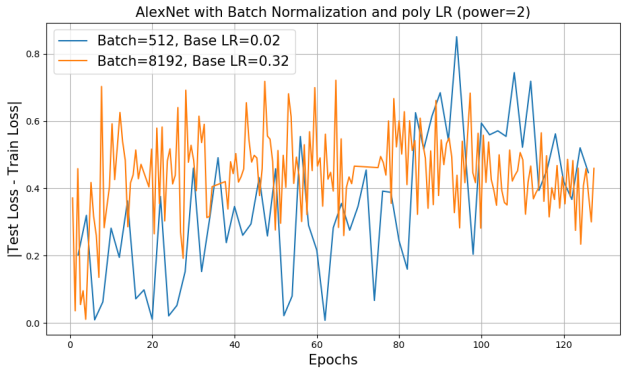


Figure 1: Alexnet-BN: Gap between training and testing loss

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

[2] LR starts from 0.001 and is linearly increased it to the target LR during 2.5 epochs

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

[4] Alexnet-BN baseline was trained using SGD with momentum=0.9, weight decay=0.0005 for 128 epochs. We used polynomial (power 2) decay LR policy with base LR=0.02.

4 LAYER-WISE ADAPTIVE RATE SCALING (LARS)

The standard SGD uses the same LR λ for all layers: $w_{t+1} = w_t - \lambda \nabla L(w_t)$. When λ is large, the update $||\lambda * \nabla L(w_t)||$ can become larger than $||w||$, and this can cause the divergence. This makes the initial phase of training highly sensitive to the weight initialization and to initial LR. We found that the ratio the L2-norm of weights and gradients $||w||/||\nabla L(w_t)||$ varies significantly between weights and biases, and between different layers. For example, let's take AlexNet-BN after one iteration (Table 2, "*.w" means layer weights, and "*.b" - biases). The ratio $||w||/||\nabla L(w)||$ for the 1st convolutional layer ("conv1.w") is 5.76, and for the last fully connected layer ("fc6.w") - 1345. The ratio is high during the initial phase, and it is rapidly decrease after few epochs (see Figure 2).

Table 2: AlexNet-BN: The norm of weights and gradients at 1st iteration.

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

If LR is large comparing to the ratio for some layer, then training may becomes unstable. The LR "warm-up" attempts to overcome this difficulty by starting from small LR, which can be safely used for all layers, and then slowly increasing it until weights will grow up enough to use larger LRs.

We would like to use different approach. We use local LR λ^l for each layer l:

$$\triangle w_t^l = \gamma * \lambda^l * \nabla L(w_t^l) \tag{4}$$

where γ is a global LR. Local LR λ^l is defined for each layer through "trust" coefficient $\eta < 1$:

$$\lambda^l = \eta \times \frac{||w^l||}{||\nabla L(w^l)||} \tag{5}$$

The η defines how much we trust the layer to change its weights during one update [5]. Note that now the magnitude of the update for each layer doesn't depend on the magnitude of the gradient anymore, so it helps to partially eliminate vanishing and exploding gradient problems. This definition can be easily extended for SGD to balance the local learning rate and the weight decay term β:

$$\lambda^l = \eta \times \frac{||w^l||}{||\nabla L(w^l)|| + \beta * ||w^l||} \tag{6}$$

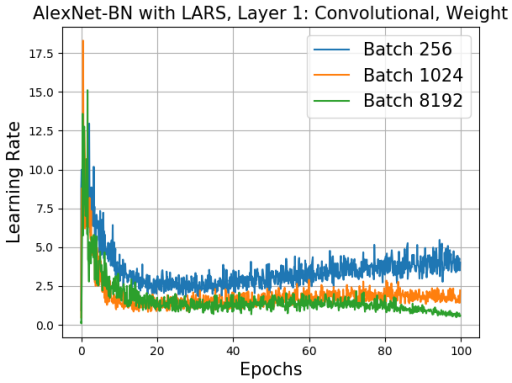
Algorithm 1 SGD with LARS. Example with weight decay, momentum and polynomial LR decay.

Parameters: base LR γ_0, momentum m, weight decay β, LARS coefficient η, number of steps T
Init: $t = 0, v = 0$. Init weight w_0^l for each layer l
while $t < T$ for each layer l **do**
  $g_t^l \leftarrow \nabla L(w_t^l)$ (obtain a stochastic gradient for the current mini-batch)
  $\gamma_t \leftarrow \gamma_0 * \left(1 - \frac{t}{T}\right)^2$ (compute the global learning rate)
  $\lambda^l \leftarrow \frac{||w_t^l||}{||g_t^l|| + \beta ||w_t^l||}$ (compute the local LR λ^l)
  $v_{t+1}^l \leftarrow m v_t^l + \gamma_{t+1} * \lambda^l * (g_t^l + \beta w_t^l)$ (update the momentum)
  $w_{t+1}^l \leftarrow w_t^l - v_{t+1}^l$ (update the weights)
end while

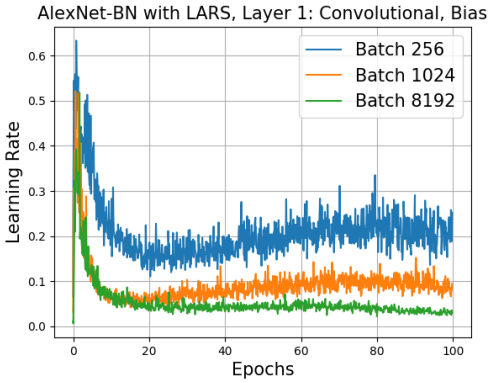
The network training for SGD with LARS are summarized in the Algorithm 1. One can find more implementation details at https://github.com/borisgin/nvcaffe-0.16

The local LR strongly depends on the layer and batch size (see Figure. 2)

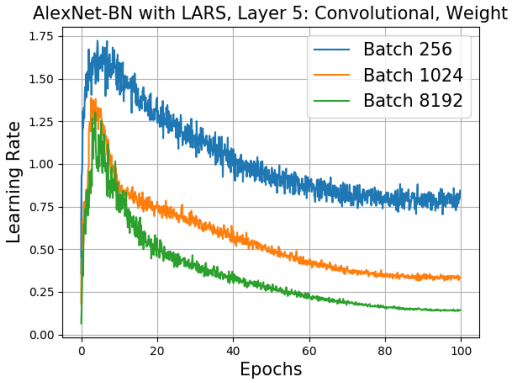
[5] One can consider LARS as a private case of block-diagonal re-scaling from Lafond et al. (2017).



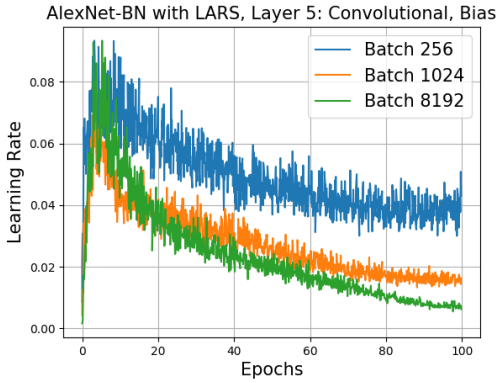
(a) Local LR, conv1-weights



(b) Local LR, conv1-bias



(c) Local LR , conv5-weights



(d) Local LR, conv5-bias

Figure 2: LARS: local LR for different layers and batch sizes

5 TRAINING WITH LARS

We re-trained Alexnet and Alexnet-BN with LARS for batches up to 32K [6]. For B=8K the accuracy of both networks matched the baseline B=512 (see Figure 3). Alexnet-BN trained with B=16K lost 0.9% in accuracy, and trained with B=32K lost 2.6%.

Table 3: Alexnet and Alexnet-BN: Training with LARS

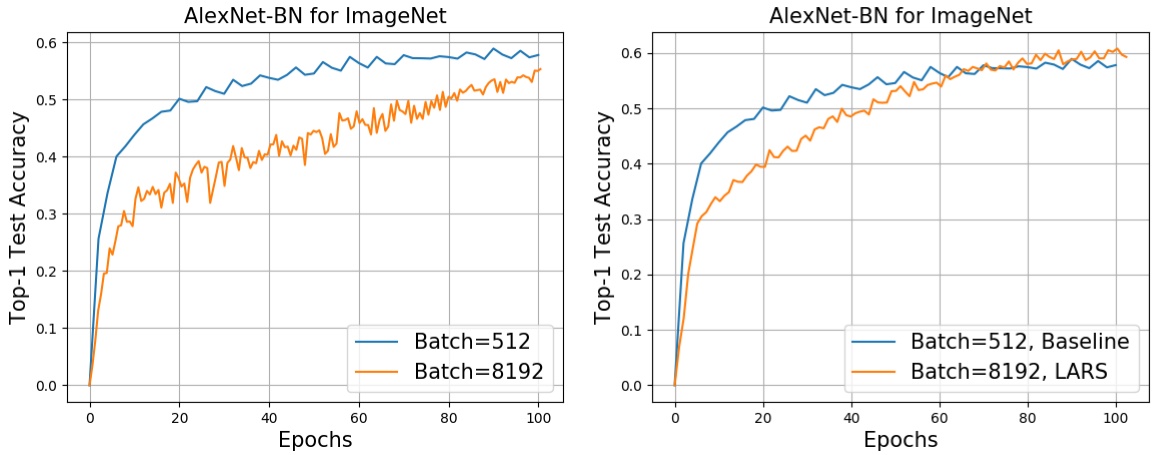
(a) Alexnet (warm-up for 2 epochs)

Batch	LR	accuracy,%
512	2	58.7
4K	10	58.5
8K	10	58.2
16K	14	55.0
32K	TBD	TBD

(b) Alexnet-BN (warm-up for 5 epochs)

Batch	LR	accuracy,%
512	2	60.2
4K	10	60.4
8K	14	60.1
16K	23	59.3
32K	22	57.8

[6] Models have been trained for 100 epochs using SGD with momentum=0.9, weight decay=0.0005, polynomial (p=2) decay LR policy, and LARS coefficient $\eta = 0.001$. Training have been done on NVIDIA DGX1. To emulate large batches (B=16K and 32K) we used $iter\_size$ parameter to partition mini-batch into smaller chunks. The weights update is done after gradients for the last chunk are computed.



(a) Training without LARS

(b) Training with LARS

LARS can support higher batch size

Figure 3: LARS: Alexnet-BN with B=8K

There is a relatively wide interval of base LRs which gives the "best" accuracy. for example, for Alexnet-BN with B=16K LRs from [13;22] give the accuracy ≈ 59.3, for B=32k, LRs from [17,28] give ≈ 57.5

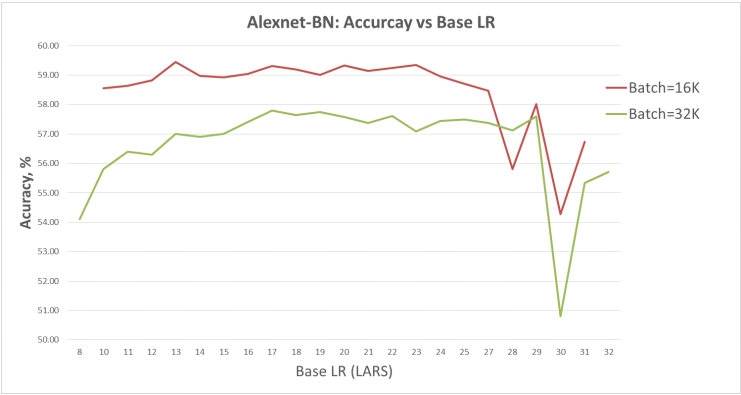


Figure 4: Alexnet-BN, B=16K and 32k: Accuracy as function of LR

Next we retrained Resnet-50, ver.1 from He et al. (2016) with LARS. As a baseline we used B=256 with corresponding top-1 accuracy 73%. [7]

Table 4: ResNet50 with LARS.

Batch	LR policy	γ	warm-up	accuracy, %
256	poly(2)	0.2	N/A	73.0
8K	LARS+poly(2)	0.6	5	72.7
16K	LARS+poly(2)	2.5	5	73.0
32K	LARS+poly(2)	2.9	5	72.3

[7] Note that our baseline 73% is lower than the published state-of-the-art 75% Goyal et al. (2017) and Cho et al. (2017) for few reasons. We trained with the minimal data augmentation (pre-scale images to 256x256 and use random 224x224 crop with horizontal flip). During testing we used one model and 1 central crop. The state-of-the art accuracy 75% was achieved with more extensive data augmentation during testing, and with multi-model, multi-crop testing. For more details see log files https://people.eecs.berkeley.edu/∼youyang/publications/batch.

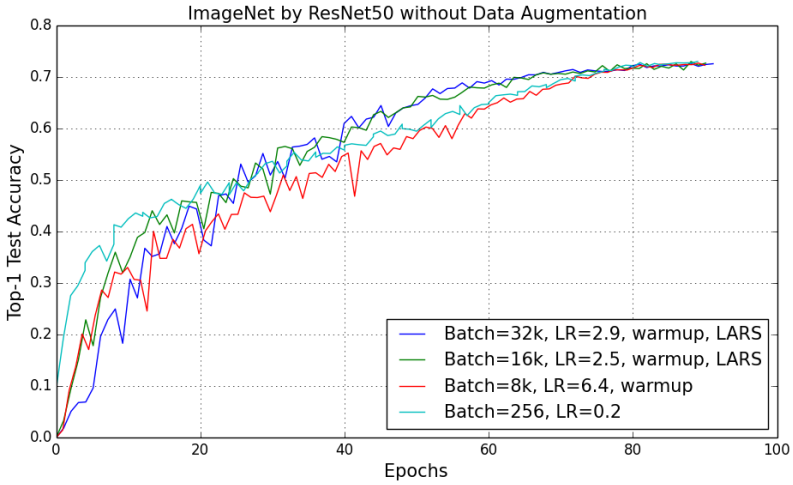


Figure 5: Scaling ResNet-50 up to B=32K with LARS.

All networks have been trained using SGD with momentum 0.9 and weight decay=0.0001 for 90 epochs. We used LARS and warm-up for 5 epochs with polynomial decay (power=2) LR policy.

We found that with LARS we can scale up Resnet-50 up to batch B=32K with almost the same (-0.7%) accuracy as baseline

6 Large Batch vs Number of Steps

As one can see from Alexnet-BN exmaple for B=32K, even training with LARS and using large LR does not reach baseline accuracy. But the accuracy can be recovered completely by just training longer. We argue that when batch very large, the stochastic gradients become very close to true gradients, so increasing the batch does not give much additional gradient information comparing to smaller batches.

Table 5: Alexnet-BN, B=32K: Accuracy vs Training duration

Num of epochs	accuracy, %
100	57.8
125	59.2
150	59.5
175	59.5
200	59.9

7 Conclusion

Large batch is a key for scaling up training of convolutional networks. The existing approach for large-batch training, based on using large learning rates, leads to divergence, especially during the initial phase, even with learning rate warm-up. To solve these optimization difficulties we proposed the new algorithm, which adapts the learning rate for each layer (LARS). Using LARS, we extended scaling of Alexnet and Resnet-50 to B=32K. Training of these networks with batch above 32K without accuracy loss is still open problem.

References

Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. Powerai ddl. *arXiv preprint arXiv:1708.02188*, 2017.

Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. Blog: Achieving deep learning training in less than 40 minutes on imagenet-1k with scale-out intel® xeon™/xeon phi™ architectures. *blog https://blog.surf.nl/en/imagenet-1k-training-on-intel-xeon-phi-in-less-than-40-minutes/*, 2017.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

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. *arXiv preprint arXiv:1705.08741*, 2017.

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

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

Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

Jean Lafond, Nicolas Vasilache, and Léon Bottou. Diagonal rescaling for neural networks. *arXiv preprint arXiv:1705.09319v1*, 2017.

Mu Li. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, CMU, 2017.

Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670. ACM, 2014.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Tech. Rep*, 2012.