Increasing Mini-Batch Size While Preserving Accuracy for Distributed Learning

Marie McCord
Middle Tennessee State University
marie.a.mccord@gmail.com

Sajal Dash
Oak Ridge National Laboratory
dashs@ornl.gov

August 2021

Abstract

Increasing complexity of deep-learning models and scale of training datasets has yielded impressive and rewarding results, but these advances have also increased training times. An effective way to reduce training time is to use data-parallel distributed training across multiple workers. Each worker in distributed training requires enough work to justify the increased communication overhead and establish a balanced computation-to-communication ratio. This issue can be addressed by using large mini-batch sizes, which refer to the total chunk of training data that is processed through the network during a single iteration. Large mini-batch sizes provide each worker with a significant amount of work and reduce communication overhead by reducing the number of training iterations. However, beyond a certain size limit, the benefit of large mini-batch sizes comes at the cost of accuracy. To understand the effects of large mini-batch sizes on accuracy and the underlying training dynamics, we identified prior techniques that successfully scaled mini-batch size to some extent. These techniques include *gradual warmup* and *linear scaling* of the learning rate. We analyzed both techniques by experimenting with the ResNet-50 model, which was trained for the standard 90 epochs on the ImageNet dataset. We found that gradual warmup is necessary to stabilize early training phases that are sensitive to learning rates after random weight initialization. There is indication that the optimal number of warmup epochs is linearly associated with mini-batch size. Beyond the optimal number of warmup epochs, accuracy decreases due to prolonging low learning rates that update weights too slowly. There is evidence that linear rate scaling does not have a significant impact on mini-batch sizes as large as 30K and is not a useful technique for scaling mini-batch to larger sizes. Future work will continue to analyze techniques that successfully scale mini-batch size in an endeavor to create a generalizable strategy.

Introduction

As the field of artificial intelligence and machine learning progresses, scientists have garnered impressive and rewarding results by increasing both the complexity of deep learning models and the scale of training datasets. There has been a noticeable convergence in recent years of big compute and big data, from which deep-learning benefits greatly. However, along with increased model complexities and dataset sizes also comes increased training times.

Data-parallel distributed learning

An effective way to decrease training time that has now become standard is data-parallel distributed training. In data-parallel distributed training, identical copies of the deep-learning model are distributed onto multiple workers, and chunks of the data are fed through each network in parallel (figure 1). The networks are all initialized with the same values, and during any single training iteration each network processes its own batch of data and calculates its own local gradient. These local gradients are aggregated and averaged together, and the average gradient is then broadcast back to all workers so that each network updates its weights with the same values.

The total chunk of data that is processed by all workers during a single training iteration is referred to as a mini-batch. Mini-batch size is thus defined as $B = s * k$, where s is the chunk of data processed per worker and k is the number of workers. The number of iterations during a single training epoch then becomes $I = T/(s * k)$, where T is the total size of the training dataset. As the number of workers k increases, the number of training iterations decrease and overall training time is reduced.

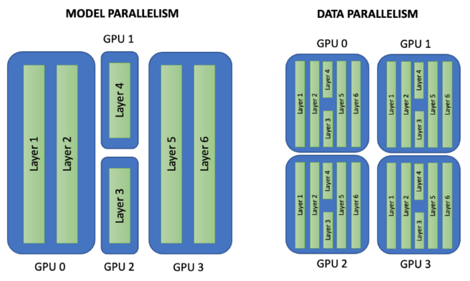


Figure 1: In model parallelism, the network itself is divided into pieces and distributed among multiple workers. In data parallelism, identical copies of a complete model are distributed among multiple workers and chunks of data are processed in parallel [6].

Scaling ratio

Data-parallel distributed training reduces training time, but it also introduces communication overhead as the workers must convey their gradients and receive the average. The computation workload per worker therefore needs to be substantial enough to fully utilize the worker's abilities and justify this communication overhead. You et al. (2017) introduce the term **scaling ratio** in their paper "ImageNet Training in Minutes", which refers to the computation-to-communication ratio of data-parallel distributed training [9]. This ratio should remain balanced, or else a bottleneck that impedes training will occur. If the workload is too low compared to the number of workers, then the communication overhead will be the bottleneck; alternatively, if the number of workers is too low compared to the workload, then computation will be the bottleneck. When using distributed training for complex models on large-scale datasets, the aim is to reduce training time by scaling up the number of workers. This creates the need to increase the computation workload per worker so that the scaling ratio remains balanced, preventing a communication bottleneck.

Large mini-batch sizes

Increasing s in $B = s * k$ achieves the goal of increasing the computation workload of each worker, ensuring that the scaling ratio remains balanced. Additionally, since $I = T/(s * k)$, increasing s also reduces the number of training iterations, and thus overall training time. This means that large mini-batch sizes B are generally ideal for distributed learning. Unfortunately, ever-growing mini-batch sizes are not a current possibility. Beyond a certain size limit, increasing mini-batch sizes start to cause decreasing accuracy rates, and eventually network divergence.

Background Work

Facebook scales mini-batch size to 8K

Facebook pushed the boundary when they successfully scaled mini-batch size without losing accuracy in their 2017 paper, "Accurate, Large Minibatch SGD: Training Imagenet in 1 Hour". Goyal et al. trained ResNet-50 for 90 epochs on the ImageNet dataset, which consists of 1.2 million training images, 50,000 validation images, and 1000 categories. They scaled mini-batch sizes up to 8K while preserving top-1 tier accuracy by employing the two methods of **gradual warmup** and **linear rate scaling**. Goyal et al. suggested that a main challenge for large mini-batch sizes is optimization issues during early training [3].

Methods

Gradual warmup

Due to random weight initialization, the gradient changes considerably during early training phases. This means that a large learning rate is more likely to cause training instability and divergence. A prudent technique is gradual warmup, which initializes training with a very low and safe learning rate and steadily increases it to the desired rate over the first few epochs. In our experiments, we test this method by training ResNet-50 for 90 epochs on the ImageNet data with 0, 5, 10, and 20 warmup epochs.

Linear rate scaling

Linear rate scaling is a technique that linearly scales the learning rate by k when the mini-batch size is scaled by k. As observed above, scaling the mini-batch size by k reduces the number of training iterations, which by definition reduces the number of weight updates. Taking bigger steps by scaling the learning rate by k compensates for the fewer number of weight updates.

For example, as You et al. explain in their 2017 paper "Large Batch Training of Convolutional Networks", the weight update after two iterations for mini-batch size B using Stochastic Gradient Descent is:

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

If $k = 2$, then the weight update after one iteration for a scaled mini-batch size of $2 * B$ and a scaled learning rate of $2 * \lambda$ is:

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

The two equations are similar assuming $\nabla L(x_j, w_{t+1}) \approx \nabla L(x_i, w_t)$ [7].

We test linear rate scaling against no scaling of the learning rate, initializing all experiments with a base learning rate of 0.0125.

Horovod All-Reduce

All-Reduce Method

During distributed training, the all-reduce method aggregates the local gradients of each worker, calculates the average, and broadcasts the result back to all workers for uniform weight updates. There have been different implementations of this operation over the years, but one of the most popular and simplest is ring all-reduce, originally developed by Baidu.

Ring All-Reduce Implementation

In the ring all-reduce implementation, each worker is connected to two other workers: one from which it receives values and the other to which it sends values. In this way, workers can be imagined as forming a logical circle or ring, hence the name (figure 2). The operation completes in $2(N-1)$ iterations, or 2 full rotations around the circle where N is the total number of workers. During the first rotation of (N–1) steps, each worker's local gradient is passed to its connecting neighbor and added to their own local gradient. At the end of this first rotation, the last worker in the circle possesses the sum of the aggregated gradients. In the second rotation of (N–1) steps, this sum is passed through the circle to all other workers, replacing their existing values. When ring all-reduce is complete, every worker possesses the aggregated gradients and can easily compute the average for uniform weight updates. A benefit of ring all-reduce is that it optimally utilizes bandwidth if the gradient buffers are large enough, leading to improved network performance [5].

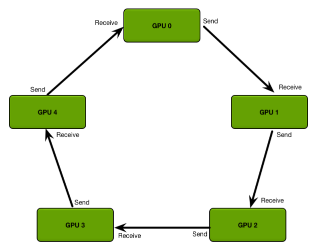


Figure 2: An illustration of ring all-reduce. Workers can be imagined as forming a logical circle or ring, hence the name [2].

Horovod Operation

Horovod is an open-source library created by Uber employees Alexander Sergeev and Mike Del Balso that implements the all-reduce method for efficient multi-GPU communication. Horovod is compatible with any version of TensorFlow and is simple to use, requiring minimal code modification. Before the creation of Horovod in 2017, distributed training with multiple workers using the standard TensorFlow was difficult, requiring explicit and heavy code modification that lead to subtle bugs and errors. Standard TensorFlow also did not employ efficient multi-GPU communication, which meant that it didn't scale well for larger numbers of workers since much of the resources were lost to communication overhead. This inspired Sergeev and Del Balso to create the Horovod, originally based on Baidu's ring all-reduce implementation. Sergeev and Del Balso first wrote the ring all-reduce implementation as a standalone Python package that was compatible with any version of TensorFlow. They secondly replaced Baidu's implementation with NVIDIA's collective communication library, NCCL, which provided an optimized version that could run on multiple machines. They thirdly added support and made improvements based on user feedback [5]. Currently, Horovod is also capable of other all-reduce implementations, such as hierarchical all-reduce and hybrid all-reduce; however, for our experiments we only used Horovod's ring all-reduce.

Periodic decay

As model accuracy converges during training, it is beneficial to gradually decrease the learning rate and take smaller steps during weight updates. This prevents the network from overstepping the minimum of the gradient as it gets closer. One technique commonly used is periodic decay, where the learning rate is linearly decreased by some scalar every specified number of epochs. We employ the use of periodic decay in all of our experiments, decreasing the learning rate by $\frac{1}{10}$ at epochs 30, 60, and 90.

Model and Equipment

All experiments used the ResNet-50 model trained for 90 epochs on the ImageNet dataset. This is the standard for contemporary research and facilitates easier comparisons of our results against others. Training was employed on the Summit supercomputer, courtesy of the Oak Ridge Leadership Computing Facility (OLCF). Each node in Summit consists of 2 IBM POWER9 CPUs and 6 NVIDIA Volta V100 GPUs. Both CPUs are located in the center of the node and are connected to 3 GPUs each, as well as to each other (figure 3). The communication network is Mellanox EDR 100G InfiniBand and Non-blocking Fat Tree.

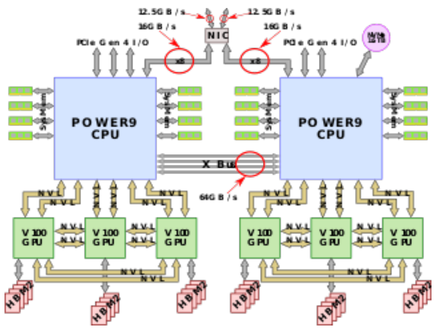


Figure 3: A single node in Summit which consists of 2 IBM POWER9 CPUs and 6 NVIDIA Volta V100 GPUs [1].

Experiments and Results

Baseline data

Our baseline data was collected using 40 nodes and run with mini-batch sizes of 3,840, 7,680, 15,360, 23,040, and 30,720 (figures 4 and 5). We additionally used 5 warmup epochs, the linear scaling rule, Horovod all-reduce, and periodic decay.

The research problem of decreasing accuracy with increasing mini-batch size is clearly visible.

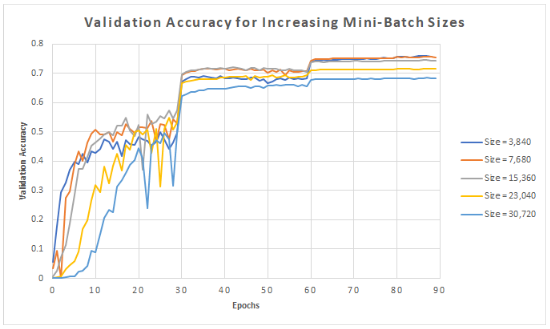


Figure 4: Overall validation accuracy decreases with increasing mini-batch sizes.

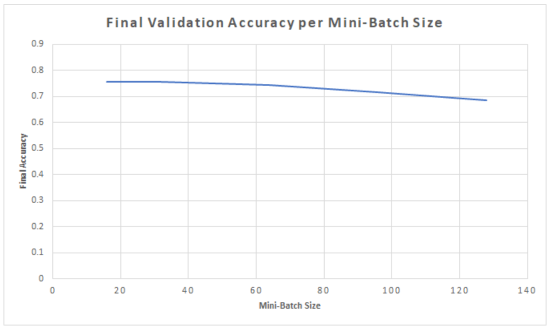


Figure 5: Final validation accuracy decreases with increasing mini-batch size.

Gradual warmup experiments

We ran a series of experiments for mini-batch sizes 7,680 and 30,720 with gradual warmup for 0 epochs, 5 epochs, 10 epochs, and 20 epochs (figures 6a, 6b, 6c, 7a, and 7b). Additionally, we employed the techniques of linear rate scaling, Horovod all-reduce, and periodic decay.

For all experiments, training without a warmup produced the worst accuracy rates. For mini-batch size 7K, the optimal number of warmup epochs is 5. For mini-batch size 30K, the optimal number of warmup epochs is 10. For all experiments, accuracy decreases past the optimal number of warmup epochs.

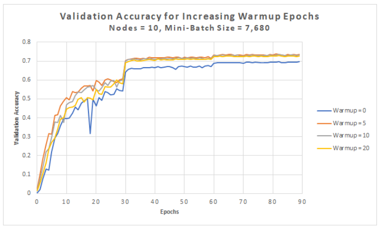
These results make sense as the initial phases of training generally have extreme gradient changes due to random weight initialization, which makes them sensitive to high learning rates. A gradual warmup with an initial low and safe learning rate is necessary to stabilize the network, but only for the first few epochs until the gradient changes become less extreme. After that, low learning rates will begin to impede training by taking too small of steps and updating the weights too slowly.

Since we used linear rate scaling, which scaled the learning rate by k when mini-batch size is scaled by k, it also makes sense that the optimal number of warmup epochs is linearly scaled as well. A larger k means both larger mini-batch sizes and larger learning rates. A bigger learning rate would require more epochs to gradually increase the initialized low learning rate to the desired rate. Thus, a mini-batch size of 7K needs fewer warmup epochs than a mini-batch size of 30K.

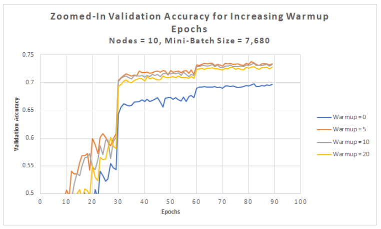
Even though the optimal number of warmup epochs appears to be linearly scaled with the mini-batch size and learning rate, it does not appear that the warmup epochs is scaled by k. The mini-batch size and learning rate from the first set of experiments are scaled by $k = 4$, yet the optimal number of warmup epochs increases by a factor of 2. Since increasing mini-batch size reduces the number of weight updates and requires a larger learning rate to compensate, as was explained above, it would make sense that the incremental scale used during gradual warmup would also need to increase to take bigger steps each iteration. If that were the case, then a proportionally longer warmup period would be necessary to accommodate a larger desired learning rate, but the incremental scale would adjust as well so that it would take slightly less steps to get there. Thus, the number of warmup epochs would be scaled by some amount less than k. More experimentation is needed to determine the exact amount to scale the warmup epochs and the precise logic behind it.

Learning rate scaling experiments

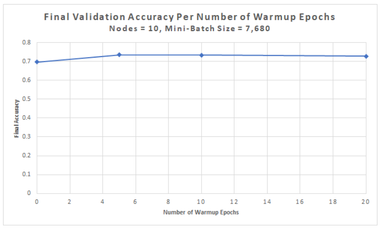
Our last series of experiments tested linear rate scaling against no scaling of the learning rate (figures 8 and 9). Both learning rates were initialized to a base rate of 0.0125. The tests were run on 40 nodes with a mini-batch size of 30,720. Additionally, we employed 5 warmup epochs, Horovod all-reduce, and periodic decay.



(a) The accuracy plots follow slightly different trajectories for 0, 5, 10, and 20 warmup epochs. The optimal number of warmup epochs for mini-batch size 7K is 5.

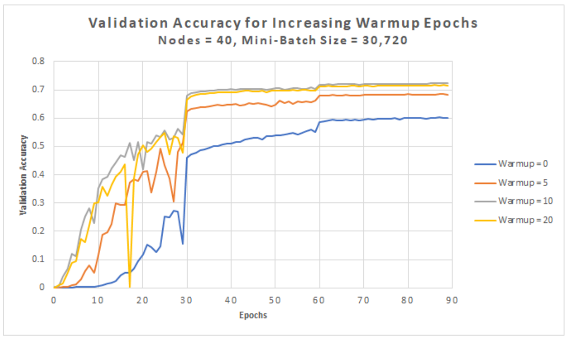


(b) The graph is zoomed-in for better detail. The optimal number of warmup epochs for mini-batch size 7K is 5; 10 and 20 epochs yield close but suboptimal accuracies.

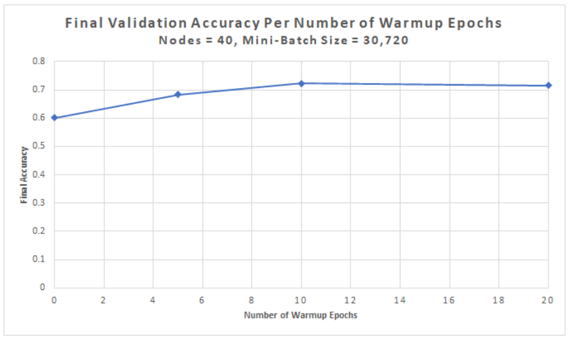


(c) No warmup produces the worst accuracy. 5 warmup epochs is optimal for mini-batch size 7K. Accuracy decreases beyond 5 warmup epochs.

Figure 6: Validation accuracies for different warmup epochs for minibatch size 7000. Since training with no warmup epochs gives us the worst validation accuracy, we can conclude that gradual warmup is beneficial in mitigating the accuracy drop while training with large minibatches.



(a) The optimal number of warmup epochs for mini-batch size 30K is 10. Training with 0 warmup epochs yields the worst performance.



(b) No warmup produces the worst accuracy. 10 warmup epochs is optimal for mini-batch size 30K. Accuracy decreases beyond 10 warmup epochs.

Figure 7: Validation accuracies for different warmup epochs for minibatch size 30000. We observe that the optimal number of warmup epochs increased from 5 to 10 when the minibatch size increased from 7000 to 30000. This change in optimal value strongly suggests that the optimal number of warmup epochs is linearly correlated to the minibatch size.

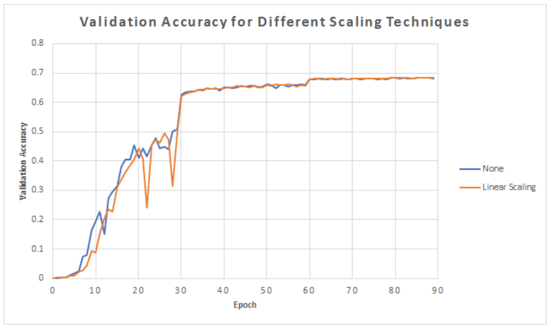


Figure 8: No significant difference in the validation accuracy between linear scaling and no scaling of base learning rate = 0.0125.

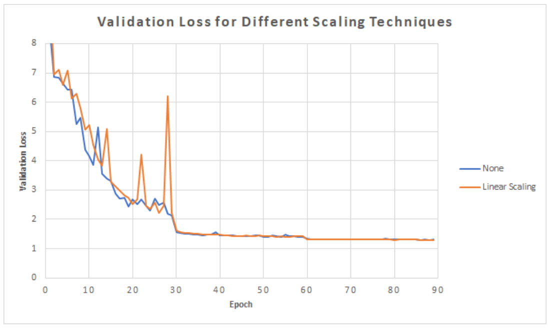


Figure 9: No significant difference in the validation loss bewteen linear scaling and no scaling ofbase learning rate = 0.0125.

Our results show that there is no significant difference between the run with linear rate scaling and the run without any scaling. Both runs achieve similar, subpar accuracies that are below the top-1 tier accuracy of 75.3

This suggests that at large mini-batch sizes of 30k, linear rate scaling has very little to no effect. A better scaling method for the learning rate is needed for mini-batch sizes this large.

Conclusions

Our conclusions are that gradual warmup, which affects the learning rate hyperparameter, is necessary for early training phases to stabilize the network after random weight initialization. Once the initial phases of training and weight updates have occurred, and the changes in gradient are no longer so pronounced, then the continued use of a low learning rate will impede accuracy by updating weights too slowly. When using linear rate scaling, the optimal number of warmup epochs needs to be scaled as well, so larger mini-batch sizes with larger learning rates require longer warmup periods. More experimentation is needed to determine the exact amount by which the warmup period should be scaled, as it does not appear to be precisely k.

Furthermore, while linear rate scaling may increase accuracy for a range of smaller mini-batch sizes, it seems to have no impact with mini-batch sizes as large as 30k. For mini-batch sizes of 30k, a learning rate scaling method is necessary to maintain accuracy, but it must be a more advanced and optimized technique than simply linearly scaling the learning rate with mini-batch size.

Future Work

Future work will further analyze the techniques and combinations of gradual warmup, linear-epoch gradual warmup (LEGW) [8], linear rate scaling, layer-wise adaptive rate scaling (LARS) [9], periodic decay, and polynomial decay. In addition, we will examine the techniques of mixed-precision training [4], elimination of weight decay on bias and batch normalization layers [4], label smoothing, and other optimized all-reduce methods besides Horovod, such as hierarchical and hybrid all-reduce [4]. Through understanding the range of parameters affected by each technique and the underlying training dynamics, we hope to create a generalizable strategy to further increase mini-batch size sometime in the future.

Acknowledgements

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of

Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship program.

References

[1] summit single-node.svg. SVG File, June 2018. Illustration of Summit node.

[2] Andrew Gibiansky. Bringing hpc techniques to deep learning. Blog, February 2017. Illustration of ring all-reduce.

[3] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[4] 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. *CoRR*, abs/1807.11205, 2018.

[5] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. Technical report, Uber, 2017.

[6] Jordi Torres.AI. Deep learning frameworks for parallel and distributed infrastructures. Online Article, November 2020. Illustration of data parallelism.

[7] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for imagenet training. *CoRR*, abs/1708.03888, 2017.

[8] Yang You, Jonathan Hseu, Chris Ying, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large-batch training for LSTM and beyond. *CoRR*, abs/1901.08256, 2019.

[9] Yang You, Zhao Zhang, Cho-Jui Hsieh, and James Demmel. 100-epoch imagenet training with alexnet in 24 minutes. *CoRR*, abs/1709.05011, 2017.