Large-Batch Training for LSTM and Beyond

Yang You
James Demmel
Kurt Keutzer
Cho-Jui Hsieh
Chris Ying
Jonathan Hseu

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-138
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-138.html

November 14, 2018

Acknowledgement

We would like to thank Sameer Kumar and Tao Wang at Google for helping us implement our algorithm on TPU Pod.

Large-Batch Training for LSTM and Beyond

Yang You[*] **James Demmel, Kurt Keutzer** UC Berkeley
Cho-Jui Hsieh UCLA & Google
Chris Ying, Jonathan Hseu Google

Abstract

A large-batch training approach has enabled us to apply large-scale distributed processing. By scaling the batch size from 256 to 64K, researchers have been able to reduce the training time of ResNet50 on the ImageNet dataset from 29 hours to 8.6 minutes. However, there are three problems in current large-batch research: (1) Although RNN techniques like LSTM [12] have been widely used in many real-world applications, the current large-batch research is only focused on CNN applications. (2) Even for CNN applications, there is no automated technique for the extending the batch size beyond 8K. Instead it requires significant parameter turning. (3) To keep the variance in the gradient expectation constant, theory suggests Sqrt Scaling scheme should be used in large-batch training. Unfortunately, there is no successful application using such a Sqrt Scaling scheme. In this paper, we propose a new approach called linear-epoch gradual-warmup (LEGW) for better large-batch training. We call this approach **Leg-Warmup**. We observe that LEGW achieves much better results than previous Linear Scaling learning rate scheme (Figure 1). With LEGW, we are able to conduct large-batch training for both CNNs and LSTMs with the Sqrt Scaling scheme. LEGW enables Sqrt Scaling scheme in practice and we achieve much better results than Linear Scaling learning rate scheme (Figure 1). For LSTM applications, we are able to scale the batch size by 64 times without losing accuracy and without tuning the hyper-parameters. For CNN applications, LEGW is able to achieve the constant accuracy when we scale the batch size to 32K. LEGW works better than previous large-batch auto-tuning techniques (Figure 1). We also provide some theoretical explanations for LEGW.

1 Introduction

Speeding up Deep Neural Networks (DNN) training is important because it can improve the productivity of machine learning researchers and developers. Since the efficiency of model parallelism is limited, the current research focus is on data parallelism. Concretely, the large-batch training approach has enabled us to successfully apply large-scale distributed processing [1, 9, 15, 20, 32]. By scaling the batch size from 256 to 64K, researchers are able to reduce the training time of ResNet50 for ImageNet from 29 hours [10] to 8.6 minutes [15]. However, there are three problems in current large-batch study:

- Although RNN techniques like LSTM [12] have been widely used, the current large-batch study is focused on CNN applications. On the other hand, adaptive solvers like Adam do not beat well-tuned Momentum SGD for ImageNet training. We want to evaluate Adam for large-batch LSTM training.
- Even for CNN applications, significant hyper-parameter tuning is required to increase the batch size beyond 8K with no loss in accuracy. For batch sizes lower than 8K, linear scaling usually works well for most applications. However, for batch sizes beyond 8K, even solvers

[*]Work done as part of internship at Google Brain.

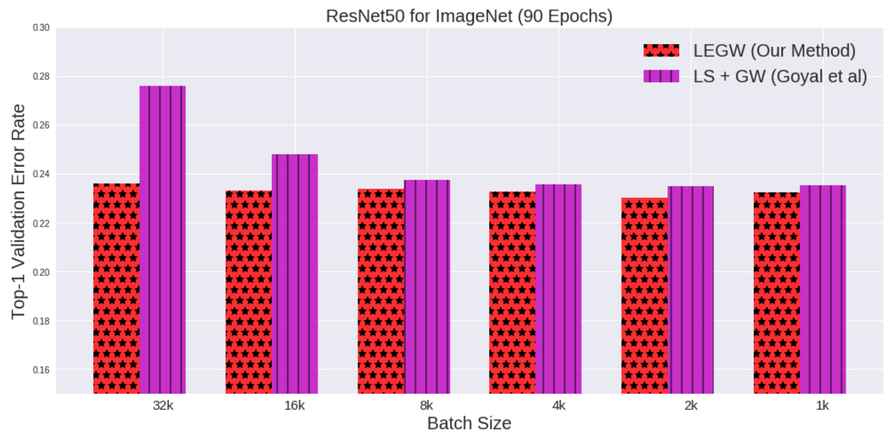


Figure 1: LEGW achieves the constant accuracy when we scale up the batch size without tuning the parameters. LEGW works better than previous large-batch tuning techniques (Goyal et al [9]).

like LARS [31] requires users to manually tune the hype-parameter (including learning rate, warmup, weight decay, and momentum).

- Prior successful large-batch training applications depend on a linear scaling scheme [9]. However, to keep the variance in the gradient expectation constant, theory [18] suggests Sqrt Scaling scheme should be used. Currently there is no successful large-batch training scheme using Sqrt Scaling.

To solve these problems, we propose linear-epoch gradual-warmup (LEGW) approach in this paper. We call this approach **Leg-Warmup**. LEGW enables Sqrt Scaling scheme in practice and as a result we achieve much better performance than the previous Linear Scaling learning rate scheme. For the MNIST dataset with LSTM, we are able to scale the batch size by a factor of 64 without losing accuracy and without tuning the hyper-parameters mentioned above. For the PTB dataset with LSTM, we are able to scale the batch size by a factor of 32 without losing accuracy and without tuning the hyper-parameters. Beyond RNN applications, we also successfully applied LEGW in ImageNet training with ResNet50. Together with LARS solver, LEGW is able to achieve the constant accuracy when we scale the batch size to 32K. LEGW works better than previous large-batch tuning techniques (Figure 1). We also give some theoretical insights to explain why LEGW works well in large-batch training.

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 as the number of samples in X, and Y as the labels of X. Let us also 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 ($i \in \{1, 2, ..., n\}$). A typical loss function is cross-entropy loss. The goal of DNN training is to minimize the loss function defined 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).

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

This method is called as Mini-Batch SGD. To simplify the notation, we define the gradient estimator as $\nabla w_t := \frac{1}{b}\sum\nolimits_{x \in B_t} \nabla l(x, y, w)$ and the updating rule in Equation (4) can be denoted as

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

2.2 Large-Batch Training Difficulty

Increasing the batch size allows us to scale to more machines without reducing the workload on each machine. On modern architecture like TPUs, reducing the workload often leads to a lower efficiency. However, when we increase the batch size after a certain point (e.g. 1024) without carefully tuning the hyper parameters, the algorithm usually suffers from slow convergence. The test accuracy of the converged solution becomes significantly lower than the baseline [9, 13, 16, 21]. Keskar et al [16] suggested that there is a generalization problem for large-batch training. Hoffer et al [13] and Li et al [21] suggests that training longer will help algorithm to generalize better and keep the accuracy. On the other hand, Goyal et al [9] can scale the batch size to 8K without losing accuracy.

2.3 Large Batch Traing Technique

When we increase the batch size (B), we need to increase the initial LR to prevent losing accuracy [9]. There are two rules of increasing the initial LR:

Sqrt Scaling Rule [18]. When we increase the batch size by k times, we should increase the LR by $\sqrt{k}$ times to keep the variance in the gradient expectation constant.

Linear Scaling Rule [18]: When we increase the batch size by k times, we should increase the LR by k times based on the assumption that $\nabla l(x, y, w_t) \approx \nabla l(x, y, w_{t+j})$, where $j < B$.

Warmup Scheme [9] 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 few epochs. Constant warmup scheme works efficiently for prototyping object detection and segmentation [8], [22]. Goyal et al [9] showed that gradual warmup performs better than constant warmup for ResNet-50 training. Cyclical Learning Rate [28] is a similar idea to warmup scheme.

Bottou et al [2] showed that there should be an 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 scheme 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.

Krizhevsky [18] reported 1 percent loss in accuracy when he increased the the batch size from 128 to 1024. Iandola et al [14] also scaled the batch size to 1K for AlexNet and GoogLeNet. Li [20] used a batch of 5120 for ResNet-101 to train Imagenet dataset on 160 GPUs. Goyal et al [9] scaled the batch size to 8K for ImageNet training with ResNet-50. You et al [31] proposed the LARS algorithm to scale the batch size to 32K for ImageNet training. The LARS algorithm was implemented on 2048 Intel KNL chips and finishes the ImageNet training with ResNet50 in 15 minutes [32]. Codreanu et al [4] scaled DNN training on 1024 SkyLake CPUs and finished ImageNet training with ResNet50 in 44 minutes. Akiba et al [1] scaled the batch size to 32K and finish the ImageNet training with ResNet50 in 15 minutes. However, their baseline's accuracy was missing. Jia et al [15] combined LARS algorithm with mixed-precision training [25] and finished the ImageNet training with ResNet50 in 8.6 minutes. The other related directions include K-FAC [24] and dynamic batch size [6, 29].

3 Linear-Epoch Gradual Warmup (LEGW or Leg-Warmup)

The warmup technique has been successfully applied in the CNN applications [9, 31]. However, most of the RNN implementations did not use warmup techniques. On the other hand, warmup has become an additional parameter that require developers to tune, which further increases the efforts of DNN system implementation. To make things worse, large-batch training is a sharp minimal problem, a tiny change in the hyper parameters may have a significant influence on the test accuracy. We propose the Linear-Epoch Gradual Warmup (LEGW or Leg-Warmup) scheme. When we increase the batch size by k times, we also increase the warmup epochs by k times. The intuition is that larger batch size usually needs a large learning rate (LR). However, larger LR may make the training algorithm easier to diverge because the gradient changes dramatically in the beginning of neural network training. We use longer warmup to avoid the divergence of larger LR.

3.1 Sqrt Learning Rate Scaling

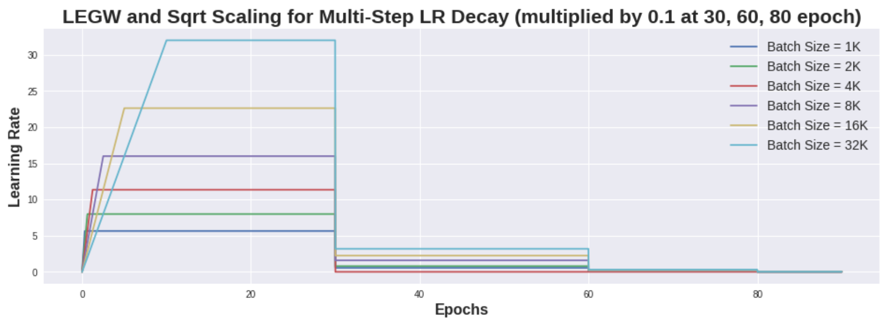
To keep the variance in the gradient expectation constant, theory [18] suggests that Sqrt Scaling scheme should be used in large-batch training in order to make variance constant. In practice, however, researchers observe that Linear Scaling scheme works better than Sqrt Scaling scheme [9, 18, 20, 32]. The constant-epoch warmup scheme was used together with Linear Scaling in previous applications. For example, Goyal et al [9] manually set the warmup length at five epochs. The efficiency of Linear Scaling only works up to 8K batch size, although researchers are able to scale the batch size to 32K with signifiant hyper-parameter tuning (tuning learning rate, warmup, weight decay and momentum for different batch sizes). With the LEGW scheme, Sqrt Scaling scheme can work well in practice, which is able to match the expectation of theory analysis. The results are shown in Section 5.

3.2 Illustration of LEGW

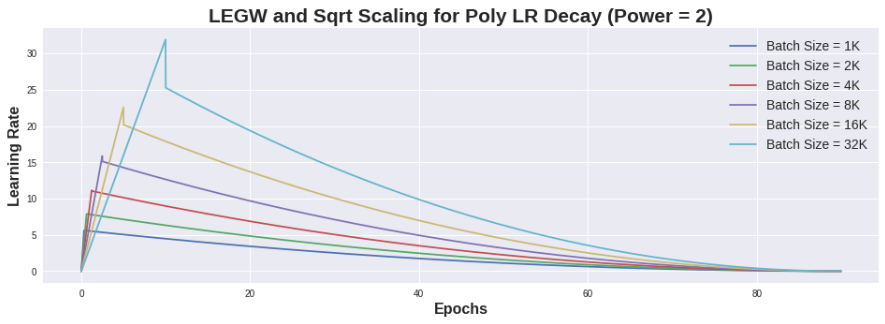
To illustrate how LEGW works with commonly-used LR decay schemes, we use ImageNet training with ResNet50. First, we use multi-step LR decay (essentially the same with exponential decay) scheme (Figure 2.1). The baseline uses a batch size of 1K and an initial learning rate of $2^{2.5}$. In the initial 0.3125 epochs (35,190 iterations), the LEGW gradually increases LR from 0 to $2^{2.5}$. From 0.3125 epoch to 30th epoch, LEGW uses the constant learning rate of $2^{2.5}$. From 30th epoch to 60th epoch, LEGW uses the constant learning rate of $0.1 \times 2^{2.5}$. From 60th epoch to 80th epoch, LEGW uses the constant learning rate of $0.01 \times 2^{2.5}$. From 80th epoch to 90th epoch, LEGW uses the constant learning rate of $0.001 \times 2^{2.5}$. When we scale the batch size from 1K to 2K, LEGW increases the learning rate from $2^{2.5}$ to $2^{3.0}$ based on the Sqrt Scaling scheme. For batch size of 2K, LEGW warms up the learning rate gradually in the initial 0.625 epochs. In the same way, LEGW reduces the learning rate by multiplying it by 0.1 at 30th, 60th, and 80th epoch. The same idea applies when we further scale up the batch size. The details are shown in Figure 2.1. Some users may feel there are too many parameters in multi-step LR decay scheme. The users may need to decide which epoch to decay the learning rate and how much learning rate should be reduced each time. Another commonly-used scheme is polynomial decay or poly decay. Let us use p to denote the power of poly decay, η the initial LR, i the current iteration, and I the total number of iterations. The learning rate of iteration i is $\eta \times (1 - i/I)^p$. The baseline uses a batch size of 1K and an initial learning rate of $2^{2.5}$. In the initial 0.3125 epoch (35,190 iterations), the LEGW gradually increases LR from 0 to $2^{2.5}$. From 0.3125 epoch to 90th epoch, LEGW uses the learning rate of $2^{2.5} \times (1 - i/I)^2$. The final LR of poly decay is 0.0. The same idea applies when we further scale up the batch size. The details are shown in Figure 2.2.

3.3 Minimal Tuning Effort

By using LEGW, the users do not need to manually tune the LR for different batch sizes. For example, the users only need to tune the hyper parameters of a baseline (e.g. batch size = 256). Then, if the user scales up the batch size by k times, they only need to increase the learning rate by $\sqrt{k}$ times and warmup epochs by k times. On the other hand, the users may also choose and tune the hyper parameters of a large-batch case (e.g 32K) and then use LEGW to automatically get the LR schedule for smaller-batch cases. Running a large-batch case is much faster than running a baseline (if the users have enough computational resources). So tuning large batch maybe faster than tuning the small batch. In the way, when user scales down the batch size by k times, they only need to decrease the learning rate by $\sqrt{k}$ times and warmup epochs by k times.



2.1



2.2

Figure 2: Example of LEGW for ImageNet training with ResNet50. The figures show the examples of using multi-step learning rate decay and using polynomial learning rate decay (power=2.0).

4 Explanation of LEGW

In general, it is hard to prove that a specific learning rate schedule works. However, some experimental findings on the change of local Lipschitz constant during iterations partially explained why the linear warm-up scheme works better.

Consider the update along the direction g. Assume the update is $x \leftarrow x - \eta g$, the question is: how to choose step size η? One classical idea is to form a second order approximation around current solution x: f(x+$\Delta) \approx \tilde{f}(x+\Delta) := \{f(x) + \Delta^T \nabla f(x) + \frac{1}{2}\Delta^T \nabla^2 f(x) \Delta\}$, and then find Δ to minimize the approximation function. If we assume $\Delta = -\eta g$, then the optimal η will be $\eta^* = \arg\min_\eta \tilde{f}(x - \eta g) = \frac{1}{\|g^T \nabla^2 f(x) g\| / \|g\|^2} := \frac{1}{L(x,g)}$. Therefore, ideally the step size should be inversely proportional to $L(x, g)$. Moreover, it is known that the update $-\eta g$ will decrease the objective function if $\eta < \min_{x' \in S} \frac{1}{L(x,g)}$ within the region S. This is also called the local Lipchitz constant on one dimension space, and $L(x, g)$ can be viewed as its approximation. In Figure 3, we plot the values of $L(x, g)$ for all the iterations. It is hard to compute $L(x, g)$ exactly since $\nabla^2 f(x)$ involves all the training samples). So we approximate it using a small batch and compute the Hessian-vector product by finite difference. Due to the same reason it's hard to apply a second order method exactly, but the plots in the figures show an interesting phenomenon which explains why linear warmup works. We observe that the value of $L(x, g)$ usually has a peak in the early iterations, implying a smaller step size should be used in the beginning. Furthermore, the peak tends to shift toward right (almost linearly) when as batch size grows. This intuitively explains our linear warm-up strategy—when batch size increases, the warm up should be longer to cover the "peak region".

5 Experimental Results

In all the comparisons of this paper, different methods will use the same hardware and run the same number of epochs.

5.1 The LSTM applications

5.1.1 Handwritten Digits Recognition for MNIST Dataset

MNIST dataset [19] has 60k training examples and 10k test examples. Each sample is a 28-by-28 handwritten digit image. We use this dataset to train a pure-LSTM model. We partition each image as 28-step input vectors. The dimension of each input vector is 28-by-1. Then we have a 128-by-28 transform layer before the LSTM layer, which means the actual LSTM input vector is 128-by-1. The hidden dimension of LSTM layer is 128. Thus the cell kernel of LSTM layer is a 256-by-512 matrix. After training 25 epochs, our model is able to achieve 98.8% accuracy. We use momentum solver (momentum=0.9) and constant learning rate. The baseline's batch size is 128. Our goal is to scale the batch size to 8K without losing accuracy.

5.1.2 Language Modeling for PTB Dataset

The Penn Treebank (PTB) [23] dataset selected 2,499 stories from a three year Wall Street Journal (WSJ) collection of 98,732 stories for syntactic annotation. The vocabulary has 10,000 words. After word embedding, the input vector length is 200. The sequence length is 20. Our LSTM model has two layers. The hidden dimensions of both these two layers are 200. For both layers, the LSTM Cell Kernel is an 400-by-800 dense matrix. We use perplexity to evaluate the correctness of our LSTM model. The perplexity is essentially the cost of the function (lower is better). After training 13 epochs, our model is able to achieve 115.91 perplexity. We use momentum solver (momentum=0.9) and exponential learning rate decay. The network uses constant learning rate in the first seven epochs. Then the learning rate will be decayed by a half after each epoch. The baseline's batch size is 20. Our goal is to scale the batch size to 640 without losing accuracy.

5.2 Compared to Adaptive Solvers

Our goal is to minimize the tuning effort for large-batch training. To evaluate this we need to pick an adaptive solver as a baseline for comparison. We fully evaluate a total of seven solvers: SGD [27], Momentum [26], Nesterov [30], Adagrad [7], RMSprop [11], Adam [17], Adadelta [33]. We pick Adam and Adadelta as the baseline for adaptive solvers because they do not require the users to input hyper-parameters. For MNIST and PTB datasets, we observe Adam performs much better than Adadelta. Moreover, Adam is able to beat the existing tuning techniques (Figure 4). We tune the learning rate for batch size = 128 and refer to it as η_0. Let us also refer to batch size as B. In Figure 4.1, all the tuning versions use η_0. In Figure 4.2, all the tuning versions use the linear scaling scheme (i.e. $\eta_0 \times B/128$). In Figure 4.3, all the tuning versions use the linear scaling scheme (i.e. $\eta_0 \times B/128$) and poly decay with power = 2. In Figure 4.4, all the tuning versions use the linear scaling scheme (i.e. $\eta_0 \times B/128$), poly decay with power = 2, and 5-epoch warmup. For all of the four tuning schemes in Figure 4, Adam is able to beat them. Thus, we use Adam as the adaptive solver baseline for comparison. The comparison between Adam and LEGW is shown in Figure 5. LEGW performs much better than Adam solver for PTB datasets in the same number of epochs. For the MNIST dataset, Adam performs better than LEGW for small-batch cases. However, LEGW is more constant and achieves higher accuracy than Adam for large-batch cases. Therefore, our experiments show that LEGW is a better auto-tuning scheme compared to state-of-the-art approaches.

5.3 Compared to Comprehensive Tuning

To prove the effectiveness of LEGW, we make a comparison between LEGW and the comprehensive tuning baseline for 8K batch size. For the MNIST dataset, since the model uses constant learning rate for momentum solver, we only tune the learning rate. We comprehensively tune the learning rate and find only the range of [0.01, 0.16] is effective. After tuning the learning rate from 0.01 to 0.16, we observe that LEGW's accuracy is higher than the best tuned version (Figure 6.1). For PTB dataset, both the baseline and LEGW use the same exponential learning rate decay scheme. We comprehensively tune the initial learning rate for baseline and we find only the range from 0.1 to 1.6 is effective. Then we tune the learning rate within the effective range, the baseline's highest accuracy is still lower than LEGW's accuracy (Figure 6.2). We also run the training algorithms long enough to make sure all of them are converged. For MNIST dataset, we increase the number of epochs from 25 to 100. For PTB dataset, we increase the number epochs from 13 to 50. Even when comprehensive turning versions are allowed to run longer, LEGW is still able to beat them (Figure 7).

Table 1: LEGW scales the batch size for ImageNet training by ResNet-50 without tuning hype-parameters. According to Stanford DAWN benchmark [5], 93% top-5 accuracy for ImageNet dataset is the metric of correct ResNet50 implementation.

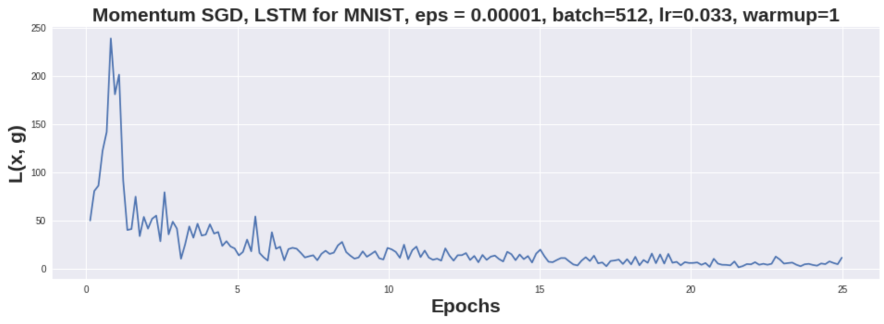
Batch Size	Init LR	LR scheme	Warmup	Epochs	Top-5 Test Accuracy
32768	$2^{5.0}$	poly power = 2	10 epochs	90	0.9318
16384	$2^{4.5}$	poly power = 2	5 epochs	90	0.9343
8192	$2^{4.0}$	poly power = 2	2.5 epochs	90	0.9355
4096	$2^{3.5}$	poly power = 2	1.25 epochs	90	0.9334
2048	$2^{3.0}$	poly power = 2	0.625 epochs	90	0.9325
1024	$2^{2.5}$	poly power = 2	0.3125 epochs	90	0.9336

6 ImageNet Training with ResNet-50

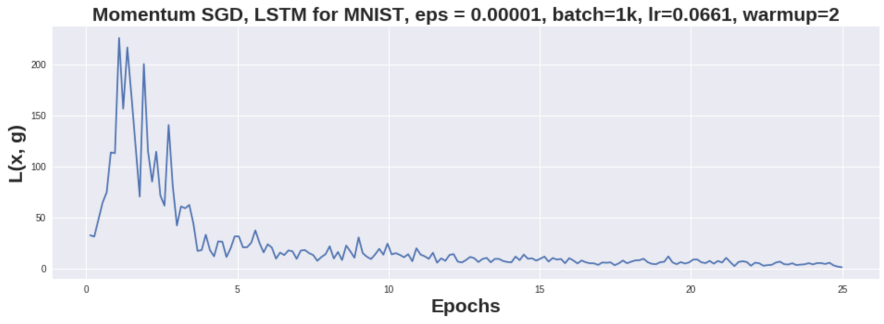
We also apply LEGW to CNN applications. We use LEGW together with LARS optimizer [31] for ImageNet training with ResNet50. According to Stanford DAWN benchmark [5], 93% top-5 accuracy is the metric of correct ResNet50 implementation. We are able to scale the batch size to 32K and achieve the target accuracy without tuning hype-parameters (Table 1). We achieved more constant performance and higher accuracy compared to existing tuning schemes (Figure 1).

7 Conclusion

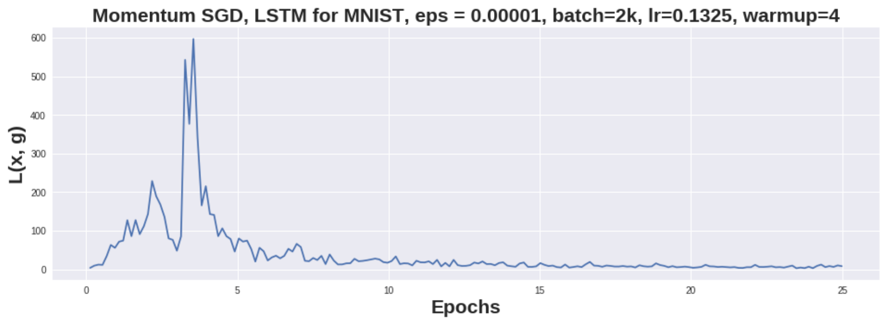
For large-batch training, warming up the learning rate is necessary to avoid divergence. LEGW is an auto-tuning technique based on warmup technique and Sqrt Scaling scheme. In practice, LEGW performs well on both RNN applications and CNN applications. For LSTM applications, we are able to scale the batch size by 64 times without losing accuracy and without tuning the hyper-parameters. For CNN applications, LEGW is able to achieve the constant accuracy when we scale the batch size to 32K. LEGW works better than previous large-batch auto-tuning techniques (Figure 1). We also provide a theoretical explanation for the effectiveness of LEGW. We conclude that LEGW is an efficient approach for large-batch training.



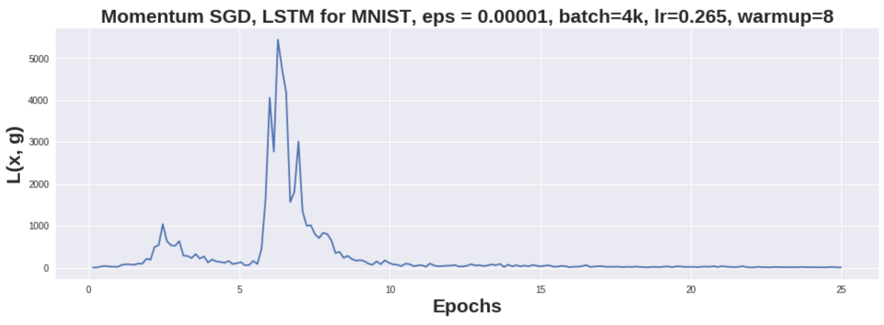
3.1



3.2

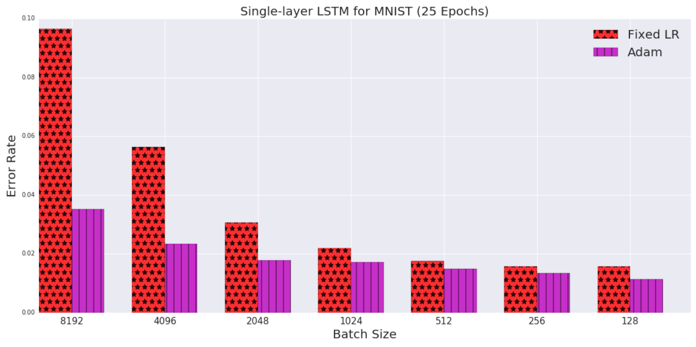


3.3

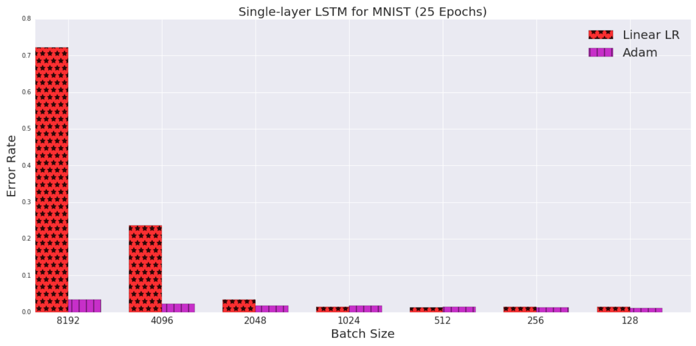


3.4

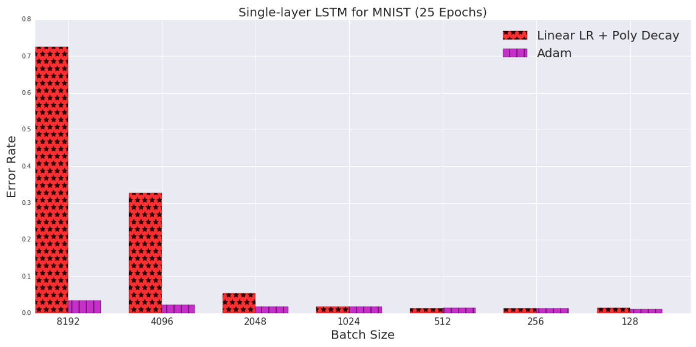
Figure 3: The approximation of Lipchitz constant for different batch sizes.



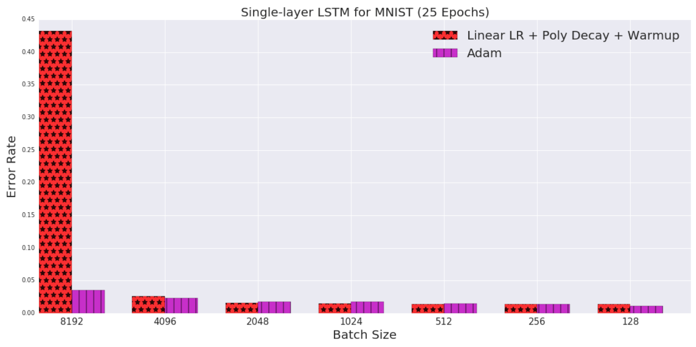
4.1



4.2

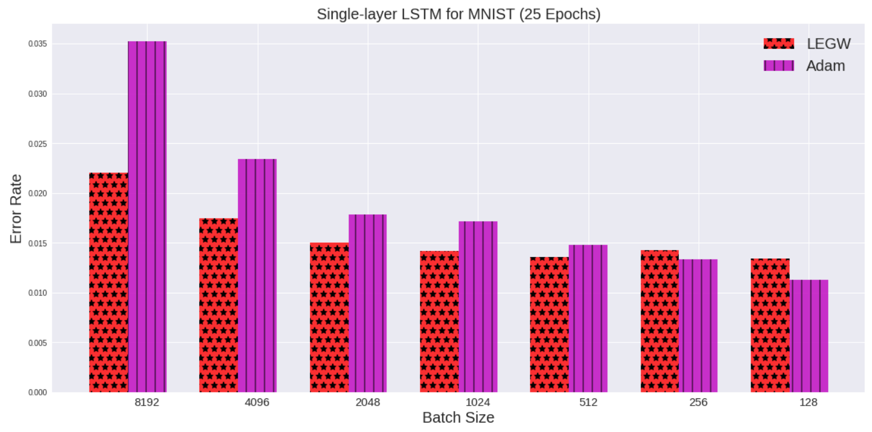


4.3

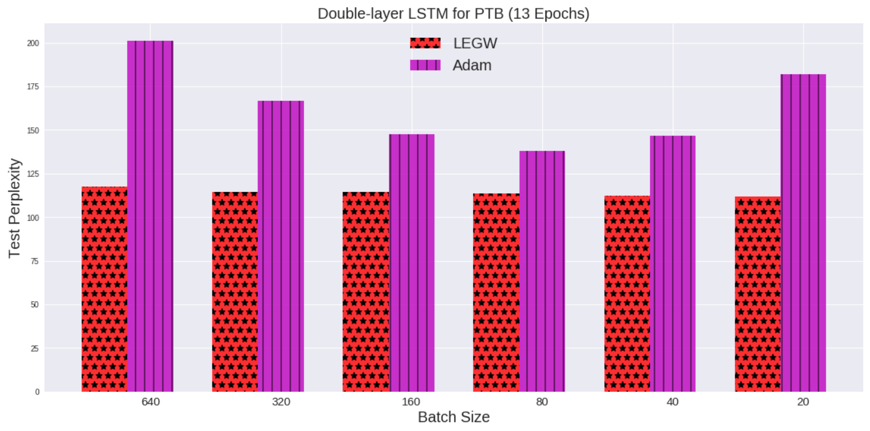


4.4

Figure 4: Adam can beat existing tuning techniques. We tune the learning rate for batch size = 128 and refer to it as η_0. Let us also refer to batch size as B. In Figure 4.1, all the tuning versions use η_0. In Figure 4.2, all the tuning versions use the linear scaling scheme (i.e. $\eta_0 \times B/128$). In Figure 4.3, all the tuning versions use the linear scaling scheme (i.e. $\eta_0 \times B/128$) and poly decay with power = 2. In Figure 4.4, all the tuning versions use the linear scaling scheme (i.e. $\eta_0 \times B/128$), poly decay with power = 2, and 5-epoch warmup.

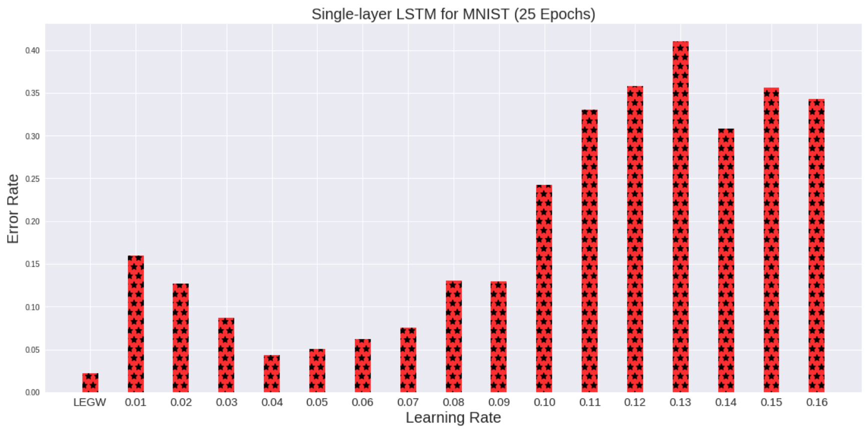


5.1

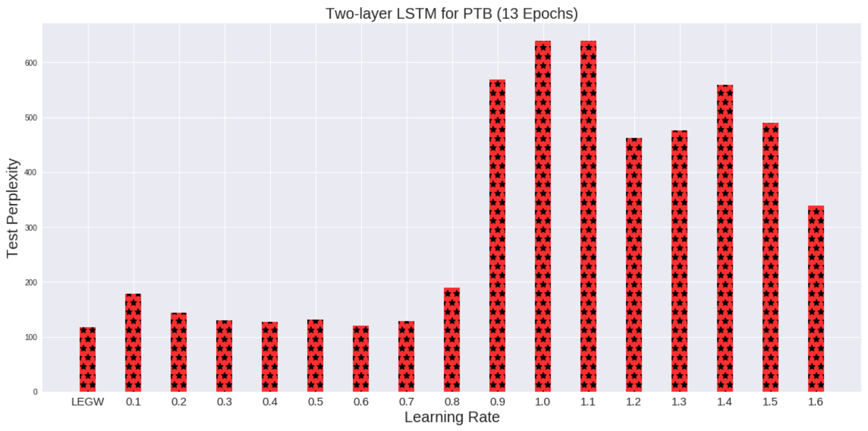


5.2

Figure 5: LEGW performs much better than Adam solver for PTB datasets (Runing the same number of epochs). For MNIST dataset, Adam performs better than LEGW for small-batch case. However, LEGW is more constant and achieves higher accuracy than Adam for large-batch cases.

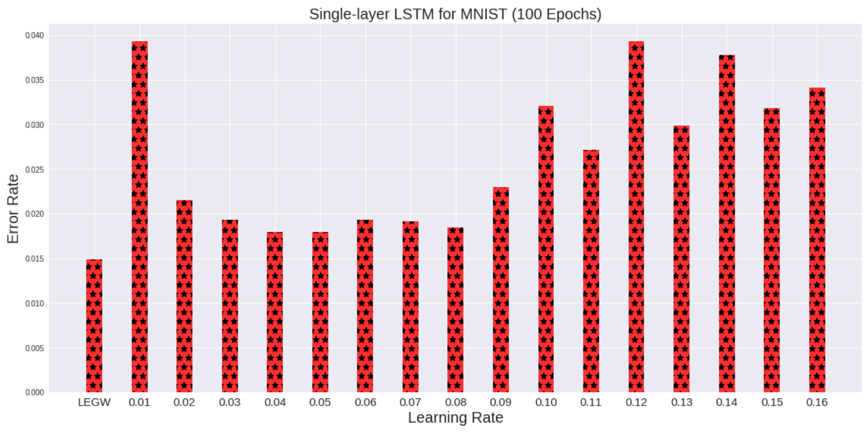


6.1

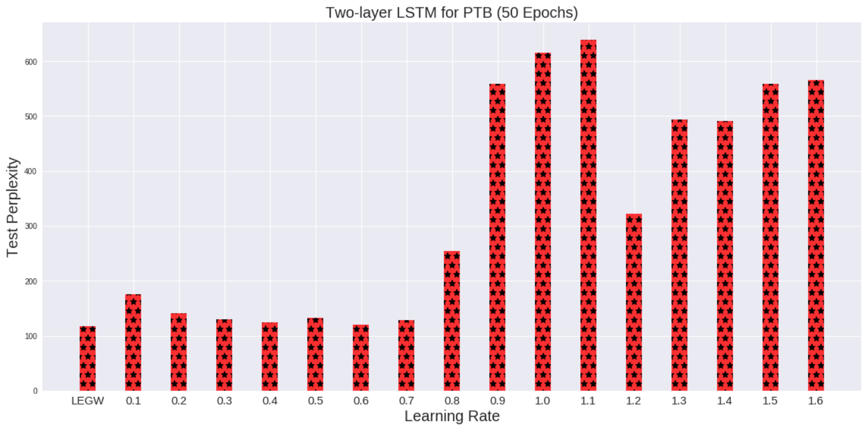


6.2

Figure 6: The data in this figure is collected from 8K batch size. Even when we comprehensively tune the learning rate of the baseline, it still is not able to beat LEGW. For other hyper-parameters and learning rate decay schemes, LEGW uses the same setting with the baseline.



7.1



7.2

Figure 7: The data in this figure is collected from 8K batch size. Even we comprehensively tune the initial learning rate of the baseline, it still is not able to beat LEGW. For other hyper parameters and learning rate decay scheme, LEGW uses the same setting with the baseline. Furthermore, we run the training long enough to make sure all of them converge. LEGW is still better.

References

[1] T. Akiba, S. Suzuki, and K. Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[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] V. Codreanu, D. Podareanu, and V. Saletore. Scale out for large minibatch sgd: Residual network training on imagenet-1k with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017.

[5] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition.

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

[7] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

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

[9] 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.

[10] 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.

[11] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] 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.

[14] 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.

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

[16] 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.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[21] 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.

[22] 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.

[23] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[24] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[25] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaev, G. Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[26] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[27] H. Robbins and S. Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985.

[28] L. N. Smith. Cyclical learning rates for training neural networks. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 464–472. IEEE, 2017.

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

[30] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[31] Y. You, I. Gitman, and B. Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017.

[32] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer. Imagenet training in minutes. *CoRR, abs/1709.05011*, 2017.

[33] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.