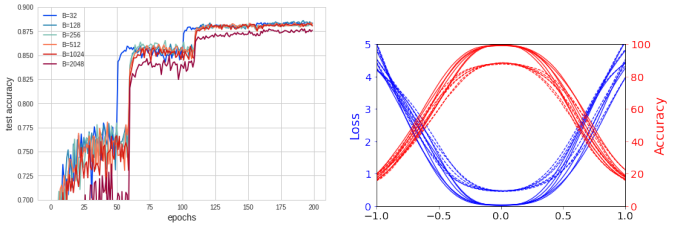
On the large-batch training of CIFAR-10

Yifei Li
321947

Haoyu Sheng
321449

***Abstract*—With the drastic increase of the size of models and datasets in recent layers, it is necessary to distribute the workloads to multiple machines. However, the linear scalability is no free lunch. While increasing the batch size, other hyper-parameters such as learning rate need to adapt accordingly. And it is also widely discovered by prior work that large-batch training usually suffers from the optimization difficulty and generalization gap. In this report, we study the linear scalability of Stochastic Gradient Descent(SGD) on the CIFAR-10 dataset, where we try to explore and answer the following two questions: (1) *how could we linearly scale without loss of generalization ability?* (2) *how much could we scale and why couldn't we scale further?* With a step-by-step identification and validation of key aspects for large-batch training, we finally achieve the linear scalability up to a factor of 32 without loss of accuracy and a factor of 64 with little loss of accuracy($< 1\%$).**



(a) Test accuracy (b) Sharpness visualization of minima

Fig. 1: We show the results of the linear scalability for the batch size from **32** to **2048**. Note this result is obtained by the methods described in Sec.III-B. In (a), we show the test accuracy with regards to training epochs. In (b), we show the sharpness of minima found by each batch size, using the method proposed in [12].

I. Introduction

The size of model and dataset is rapidly increasing these years[1][2][3]. As a result, it is nearly infeasible to utilize only one machine and we need to distribute the workloads to multiple machines. Ideally, the linear scalability is expected to be achieved, such that K times resources comes with K times reduced training time. However, trivially increasing the batch size by adding more nodes will typically lead to a longer convergence time and even do harm to the model's generalization ability, which has been extensively discussed by previous work [4][5][6][7]. P. Goyal et al. propose the linear scale rule, which scales the learning rate proportionally to the batch size. Despite its simplicity, the linear scale rule is especially effective for large-batch training [4][5]. However, even with the linear scale rule, the linear scalability will only hold below a certain threshold of batch size [4][7][8]. For one thing, the problem of the generalization gap pops out again. For another, the training procedure will even diverge for huge batch sizes[9]. Apart from aforementioned empirical results, S. Stich et al. [8] give a theoretical foundation on this phenomenon recently.

In this report, we focus on the linear scalability of SGD[10] on the CIFAR-10[11] dataset. With sufficient experiments, we try to explore and answer the following questions:

- How could we linearly scale without loss of generalization ability?
- How much could we scale and why couldn't we scale further?

II. Methods

A. SGD and linear scale rule

The update rule of mini-batch SGD is shown as below:

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathrm{B}} \bigtriangledown l(x, w_t) \tag{1}$$

where t is the index of the iteration, w are weights of the network, η is the learning rate, B is a minibatch sample from a labeled training set, and $n = |\mathrm{B}|$ is the minibatch size. l represents the sum of the training loss and a regularization term. After k iterations of mini-batch SGD, we have the following:

$$w_{k+t} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathrm{B}_j} \bigtriangledown l(x, w_{t+j}) \tag{2}$$

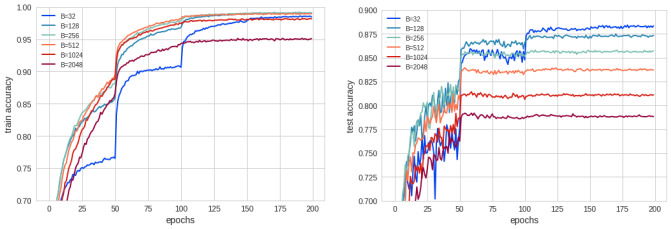
For the large minibatch $\cup_j \mathrm{B}_j$ with size kn, taking a single iteration of SGD yields the following:

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathrm{B}_j} \bigtriangledown l(x, w_t) \tag{3}$$

Under the assumption of $\bigtriangledown l(x, w_t) \approx \bigtriangledown l(x, w_{t+j})$, $\hat{w}_{t+1} \approx w_{k+t}$ holds if we set $\hat{\eta} = k\eta$, as is pointed out in [4]. Further, S.L.Smith et al. state that the linear scale rule is effective because it keeps the scale of random fluctuations in the SGD dynamics approximately unchanged [5]:

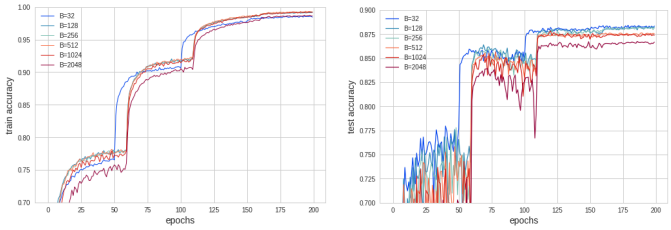
$$g = \eta(\frac{N}{B} - 1) \tag{4}$$

Aforementioned possible explanations also lead to similar assumptions: $B << N$, where N is the dataset size and B is



(a) Train accuracy (b) Test accuracy

Fig. 2: Trivially increasing the batch size(**single worker**): training and testing accuracy with regards to the training epochs. The test accuracy drops drastically while the training accuracy remains similar except for $B = 2048$.



(a) Train accuracy (b) Test accuracy

Fig. 3: Linear scale rule + warmup(**single worker**): training and testing accuracy with regards to the training epochs. The test accuracy drops notably when $B \leq 512$, while the training accuracy remains similar. The generalization gap between different batch sizes shows up.

the total batch size. S. Stich et al. formulate a unified theory to show that the linear scale rule is in essence sub-linear [8]. That is, beyond a certain batch size, the linear scale rule will gradually break and even fail.

III. Experiments

In this section, we systematically evaluate the scalability of SGD on CIFAR-10 dataset. Our main result is that we could successfully scale the batch size to 1024 without loss of generalization and 2048 with a little loss of generalization, under the constraints of linear scalability.

A. Environment Setup

Environment We use an RTX3070 graphics card and Google Colab to conduct our experiments. Detailed information is listed in Appendix A.

Dataset and Model Throughout the entire experiment section, we use CIFAR-10[11] as our experimental benchmark. We use the official train-test split-up, which contains 50000 and 10000 images respectively. As for the model, we use ResNet-18[13] as the model. The initialization follows Pytorch [14] default initialization. The parameters of convolutional layers are initialized by He-initialization [15]. As for the batch normalization layer, the γ and running variance are initialized to 1, while the β and running mean are initialized to 0.

Optimizer For each optimizer(SGD and RMSProp) used in this report, the fine-tuned hyperparameter(including the batch size) is served as the baseline model of each optimizer. The detailed specification could be found in the following sections.

Training procedure

We follow previous work [13] to conduct basic data augmentation on the dataset. Commonly, we train 200 epochs for each experiment and decay the learning rate to 0.1 every 50 epochs. For the experiments equipped with warmup scheduler[4], we gradually increase the learning rate to the expected initial learning rate in 10 epochs and then fall back to the default step scheduler

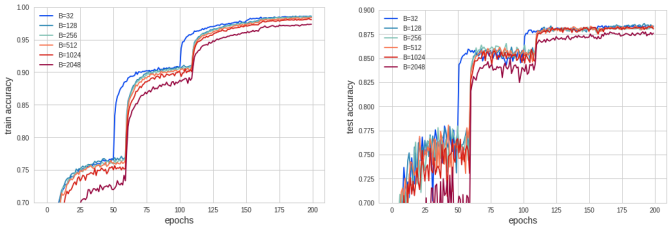
B. Validation of key aspects of large-batch training

Strong baseline model with SGD For the baseline SGD model, we finetune the hyper-parameters to construct a strong baseline model. We set momentum as 0.9, weight decay as 5×10^{-4}, and batch size as 32. The initial learning rate is 0.025 and is multiplied by 0.1 every 50 epochs during the 200-epoch training procedure. The baseline SGD model could achieve 98.45% and 88.33% accuracy on the train and test dataset respectively.

Trivially increasing the batch size In this section, we fix the worker size to 1 and trivially increase the batch size on this single worker. Other hyper-parameters are not changed compared to the baseline model. As one can see in Figure 2, the main issue between $B = 32$ and $B = 1024$ is the generalization gap, such that the test accuracy drops notably faster than the training accuracy. This phenomenon is especially significant for $B \in [32, 512]$, where the training accuracy is even higher than the baseline method. For $B = 2048$, the issue turns to the difficulty of optimization, since the training accuracy is notably lower than any other variants.

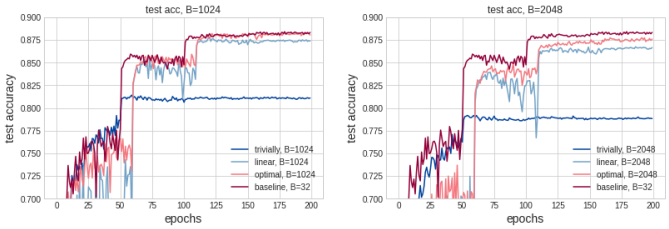
Linear scale rule with warmup scheduler Further, we follow [6] to linearly scale the learning rate with respect to the batch size. The initial learning rate is gradually approached during the first 10 epochs. Still, the worker size is 1, since we aim to demonstrate the effect of linear scale rule and warmup policy under the context of vanilla large-batch training. As is shown by Figure 3, this policy largely eases the difficulty of the optimization, such that even $B = 2048$ obtains a comparable train accuracy. What's more, it also helps maintain the generalization ability while increasing the batch size, especially for $B \leq 256$. However, beyond $B = 256$, the test accuracy drops nearly 1% for $B \in [512, 1024]$ and nearly 2% for $B > 2000$. In sum, this policy's scalability is still limited to the factor of 8.

Simulation of multi-worker training In this section, we *approximately* simulate the large-batch training on a single graphics card. Details of this simulation are presented in Appendix B. The main difference lies in the update of batch normalization running statistics. Under the scheme of vanilla large-batch training, the running mean and running variance are updated with respect to the whole batch. Yet for multi-worker large-batch training, the community usually partition



(a) Train accuracy (b) Test accuracy

Fig. 4: Linear scale rule + warmup(**multiple-worker (simulation)**): training and testing accuracy with regards to the training epochs. The test accuracy drops notably when $B \leq 512$, while the training accuracy remains similar. The generalization gap between different batch sizes shows up.



(a) Batch size = 1024 (b) Batch size = 2048

Fig. 5: Comparison of each variant in Sec. III-B with $B = 1024$ and $B = 2048$. For $B = 1024$, the multi-worker scheme("optimal" legend in the graph) could fully recover the generalization ability of the baseline method, where the single-worker scheme("trivially" and "linear" scale rule) suffers from over-fitting. For $B = 2048$, although all variants fail to fill the generalization gap, the multi-worker scheme only suffers from a 0.7% drop in the test accuracy.

the batch into small batches, and thus each worker could process assigned small batch in parallel, such that the running statistics of the batch normalization layer is summarized locally. As is pointed out by [6], this communication-efficient approach is the key to the scalability of large-batch training. We fix each worker's batch size to 32, which is the same as the baseline model, then for total batch size B, the number of workers is $\frac{B}{32}$. As Figure 4 indicates, we could successfully scale the batch size from 32 to 1024 without loss of accuracy in the same training epochs. Further, the curve of training and testing metrics are also similar to the baseline method, thus we suspect that the training trajectory is also similar compared to the baseline method. We also visualize the loss landscape by the method proposed in [12] in Figure 1, which further confirms that the generalization ability found by models of different batch sizes is also similar.

C. Why couldn't we scale further?

As is demonstrated in previous sections, we could linearly scale the batch size of the baseline model from 32 to 1024 without loss of accuracy. In this section, we take a step further to investigate the limit of the linear scale rule and discuss the reasons why the linear scale rule breaks as the batch size is increasing. In Figure 4 and 5, we can see that the test and train accuracy have slightly dropped compared to other variants. As for $B = 4096$ (η=3.2 accordingly), the training becomes unstable and even diverges in the middle of the epochs. The linear scalability tends to slow down when batch size is larger than 2048 and even breaks when it is beyond 4k. Similar phenomenon has also been discussed by [4], where they state that the linear scale rule breaks when batch size exceeds 8k on the Imagenet dataset[16], since the assumption that $w_t \approx w_{t+j}$ in Eq.1 and 2 does not apply when the learning rate and batch size are too large. On the other hand, [5] stated that the linear scale rule comes from the maintenance of the scale of random fluctuations in the SGD dynamics(Eq. 4), and will only hold if $N >> B$. [8] further gives a unified theoretical foundation for this phenomenon, such that the predicted scalability of linear scale rule is in theory sub-linear.

D. The scalability of another optimizer

In this section, we explore the scalability of another well-used optimizer RMSProp[17]. The RMSprop model with momentum=0.9, weight decay=5×10^{-2}, batch size=128, and initial learning rate=0.001 is our baseline model. It could achieve 96.06% and 87.10% accuracy on the train and test datasets respectively. Unlike vanilla SGD, this adaptive optimizer is proposed to make the training more insensitive to the learning rate. As a result, the training trajectory would vary a lot. The results of experiments in Appendix D show that the linear scale rule does not work well for RMSProp.

IV. Conclusion

In this report, we study the effective large-batch training scheme on the CIFAR-10[11] dataset. We step by step identify and validate the key aspects to realize linear scalability. Further, we explore the limitation of the linear scale rule and connect the experiment results with potential explanations proposed by the prior art. What's more, we also test the scalability of RMSProp. The adaptive optimizers like RMSProp are less insensitive to the learning rate. However, this good property is suboptimal when we try to scale since there will be no simple guide to help us maintain the performance. In short, scalability usually comes with simplicity, where the SGD, communication-efficient batch normalization, and linear scale rule are preferred. With these three simple pieces, we could parallelize the workloads without loss of accuracy up to a scaling factor of **32**, and with little loss of accuracy up to a scaling factor of **64**.

APPENDIX A
DETAILED ENVIRONMENT SETUP

For the team member using RTX 3070 graphics card, the experiments are conducted on Ubuntu 20.04 with an Amd Ryzen 5 3600 cpu. The version of cuda tool kit is 11.1 and the version of Pytorch is 1.8.1.

For the team member using Google Colab with GPU: 1xTesla K80 , compute 3.7, having 2496 CUDA cores , 12GB GDDR5 VRAM. The version of cuda tool kit is 11.1 and the version of Pytorch is 1.8.1.

APPENDIX B
SIMULATION OF DISTRIBUTED LARGE BATCH TRAINING ON A SINGLE GPU

With only one GPU, it has rendered us impossible to actually replicate a model on several workers, no matter logically or physically. As is mentioned in Sec. III-B, the key difference between multi-worker and single-worker schemes lies in the update of running statistics of batch normalization. For communication efficiency, normally the workers will update the statistics of their own mini-batches and only communicate the gradients with each other. So in our implementation, we partition the large-batch to mini-pseudo-batches(same as baseline method, i.e. 32 in SGD case). For one thing, it is different from the baseline model since we still update the model after the whole batches of the gradient are calculated. For another, it is different from vanilla large-batch training since the running mean and running variance of the batch normalization layer is updated mini-batch-wise. However, It is still an approximation of a true multi-worker scheme since the gradient with regard to the batch normalization layer and the update of running statistics is not fully simulated.

Algorithm 1 Approximate Simulation of multi-worker training

Input: BatchedTrainingData D, Optimizer $Optim$, NeuralNet: Net, LossFunction: $Lossfn$

for *inputs, targets in* D **do**
- loss := 0
- $Optim$.zero_grad()
- pseudoPhases = int(math.ceil(len(inputs) / 32))
- **for** *pseudoBatchIdx in range(pseudoPhases)* **do**
  - psuedoInputs := inputs[pseudoBatchIdx*32:pseudoBatchIdx*32+32]
  - psuedoTargets := targets[pseudoBatchIdx*32:pseudoBatchIdx*32+32]
  - psuedoOutputs := Net(psuedoInputs)
  - loss := loss + $Lossfn$(psuedoOutputs, psuedoTargets)
- loss := loss / pseudoPhases
- loss.backward()
- $Optim$.step()

APPENDIX C
EXPERIMENTS' RESULTS OF SGD

Last two columns of tables represent the test accuracy and training accuracy of the final training epoch.

Optimizer	learning rate	decay	Batch Size	Test Accuracy	Training Accuracy
SGD	0.025	0.1	32	88.33%	98.45%
			64	88.21%	98.78%
			128	87.34%	99.09%
			256	85.71%	99.00%
			512	83.70%	98.95%
			1024	81.10%	98.17%
			2048	78.83%	95.11%

TABLE I: Trivially increasing batch size(single-worker)

Optimizer	decay	warm_up	learning rate	Batch Size	Test Accuracy	Training Accuracy
SGD	0.1	10	0.025	32	88.33%	98.45%
			0.05	64	88.10%	98.92%
			0.1	128	88.24%	99.17%
			0.2	256	88.14%	99.19%
			0.4	512	87.53%	99.27%
			0.8	1024	87.38%	99.21%
			1.6	2048	86.66%	98.65%

TABLE II: Linear scale of SGD(single-worker)

Optimizer	decay	warm_up	learning rate	Batch Size	Test Accuracy	Training Accuracy
SGD	0.1	10	0.025	32	88.31%	98.57%
			0.05	64	88.18 %	98.35 %
			0.1	128	88.32%	98.58%
			0.2	256	88.07%	98.49%
			0.4	512	88.17%	98.39%
			0.8	1024	88.04%	98.07%
			1.6	2048	87.59%	97.36%

TABLE III: Simulation of multi-worker training for SGD

APPENDIX D
EXPERIMENTS' RESULTS OF RMSPROP

At first, we fix other parameters and tune the learning rate for RMSProp as shown in Table IV. After finding the best learning rate, we tune the weight decay as shown in Table V and batch size as shown in TableVI for RMSProp.

Optimizer	learning rate	decay	batch size	Test Accuracy	Training Accuracy
RMSProp	0.1	0.1	128	10.00%	9.43%
	0.01	0.1	128	75.72%	75.77%
	0.001*	0.1	128	87.09% *	96.06%
	0.0001	0.1	128	82.02%	97.20%

TABLE IV: Tuning learning rate for RMSProp

Optimizer	learning rate	decay	batch size	Test Accuracy
RMSProp	0.001	0.1	128	87.09%
		0.5	128	86.53%
		0.05 *	128	87.10% *
		0.01	128	82.020%

TABLE V: Tuning weight decay for RMSProp

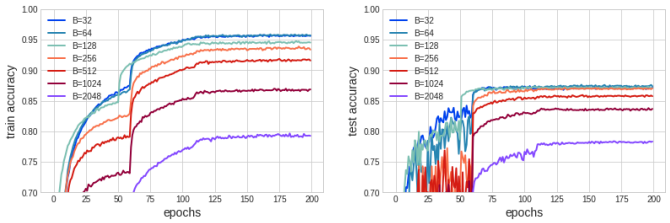
Optimizer	learning rate	decay	batch size	Test Accuracy
RMSProp	0.001	0.05	32	86.22%
			64	86.02%
			128	87.10% *
			256	86.84%
			512	85.67%

TABLE VI: Tuning batch size for RMSProp

Optimizer	decay	warm_up	learning rate	batch size	Testing Accuracy	Training Accuracy
RMSProp	0.05	10	0.00025	32	87.20%	95.61%
			0.0005	64	87.39%	95.72%
			0.001	128	87.10%	96.06%
			0.002	256	87.04%	93.38%
			0.004	512	85.75%	91.59%
			0.008	1024	83.68%	86.86%
			0.016	2048	78.33%	79.26%

TABLE VII: Simulation of multi-worker training for RMSProp

In Figure 6, we could notice that both training and test accuracy drop notably when $B \geq 512$.



(a) Train accuracy (b) Test accuracy

Fig. 6: RMSProp - Linear scale rule + warmup(**multiple worker(simulation)**)

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, D. Keysers, J. Uszkoreit, M. Lucic *et al.*, "Mlp-mixer: An all-mlp architecture for vision," *arXiv preprint arXiv:2105.01601*, 2021.

[4] 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.

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

[6] 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.

[7] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, "Don't use large mini-batches, use local sgd," *arXiv preprint arXiv:1808.07217*, 2018.

[8] S. Stich, A. Mohtashami, and M. Jaggi, "Critical parameters for scalable distributed learning with large batches and asynchronous updates," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 4042–4050.

[9] Y. You, I. Gitman, and B. Ginsburg, "Scaling sgd batch size to 32k for imagenet training," *arXiv preprint arXiv:1708.03888*, vol. 6, p. 12, 2017.

[10] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[11] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[12] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Neural Information Processing Systems*, 2018.

[13] 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*, 2016, pp. 770–778.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[17] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, 2012.