Trade-offs of Local SGD at Scale: An Empirical Study

Jose Javier Gonzalez Ortiz*
MIT CSAIL
Cambridge, MA, USA
josejg@mit.edu

Jonathan Frankle
MIT CSAIL
Cambridge, MA, USA
jfrankle@mit.edu

Mike Rabbat
Facebook AI Research
Montreal, QC, Canada
mikerabbat@fb.com

Ari Morcos
Facebook AI Research
Menlo Park, CA, USA
arimorcos@fb.com

Nicolas Ballas
Facebook AI Research
Montreal, QC, Canada
ballasn@fb.com

Abstract

As datasets and models become increasingly large, distributed training has become a necessary component to allow deep neural networks to train in reasonable amounts of time. However, distributed training can have substantial communication overhead that hinders its scalability. One strategy for reducing this overhead is to perform multiple unsynchronized SGD steps independently on each worker between synchronization steps, a technique known as *local SGD*. We conduct a comprehensive empirical study of local SGD and related methods on a large scale image classification task. We find that performing local SGD entails navigating a tradeoff: lower communication costs (and thereby faster training) are accompanied by lower accuracy. This finding is a departure from the smaller-scale experiments in prior work, suggesting that local SGD encounters challenges at scale. We further show that incorporating the slow momentum framework of Wang et al. (2020) consistently improves accuracy without requiring additional communication, hinting at future directions for potentially escaping this trade-off.

1 Introduction

As datasets and models continue to grow in size, it has become a common practice to train deep neural networks in a distributed manner across multiple hardware *workers* (Goyal et al., 2017; Shallue et al., 2018) using some variant of minibatch stochastic gradient descent (Bottou, 2010; Dekel et al., 2012). In distributed scenarios, the communication overhead necessary to synchronize gradients between workers can quickly dominate the time necessary to compute the model updates, hindering the scalability of this approach. Moreover, because of the serial nature of neural network training, all the nodes must wait until the synchronization completes, and performance is therefore dependent on the slowest node (Dutta et al., 2018; Ferdinand et al., 2020). These issues have motivated the development of optimization algorithms that reduce the amount of communication between workers. A simple yet practical example is *local SGD* (closely related to *federated averaging* (McMahan et al., 2017)) where, instead of synchronizing the gradients at every iteration, each worker performs multiple SGD steps locally and then averages the model weights across all workers (Zhang et al., 2016). However, while local SGD does speed up training, the resulting models are often less accurate compared to a synchronous minibatch SGD baseline (Lin et al., 2018).

Post-local SGD is a variant of local SGD introduced by Lin et al. (2018) with the goal of remedying these problems. Post-local SGD divides training into two phases. In the first phase, workers perform

*Corresponding Author, work done while interning at Facebook AI Research

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

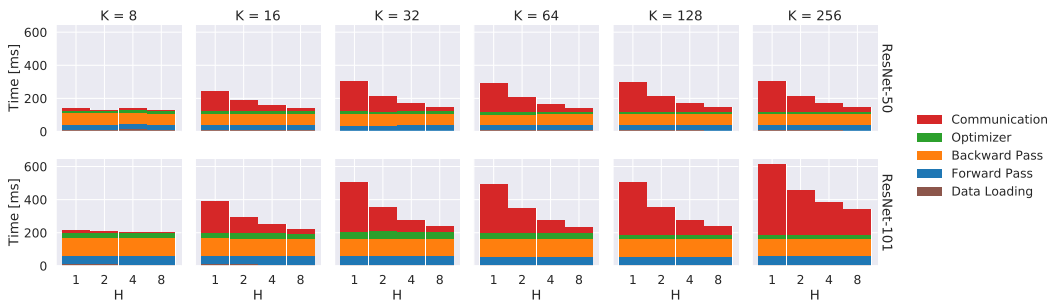

Figure 1: Breakdown of the average wall-clock time per iteration during the training process. Results are reported for various number of workers (K) and number of local steps (H).

synchronous minibatch SGD; in the second, they switch to local SGD. Lin et al. claim that this approach generalizes better on large batch training than both local SGD and minibatch SGD while reducing communication for the second phase of training. The majority of the analysis of local SGD and post-local SGD reported in Lin et al. (2018) is on CIFAR-10 (Krizhevsky et al., 2009). It is not clear whether similar trends hold when training with larger models and/or larger datasets that may be more representative of typical distributed training workloads.

Motivated by these results and the potential of post-local SGD to reduce the communication overhead and, thereby, the training times of large-scale, distributed training jobs, we perform an thorough analysis of local SGD and post-local SGD on the ImageNet-1k classification task (Russakovsky et al., 2015), a de facto benchmark for large-scale vision classification problems. We go beyond the analysis of Lin et al. (2018), studying how the choice of learning rate schedule and the point at which to switch phases affect the generalization accuracy of models trained with post-local SGD. We find that post-local SGD at ImageNet-scale is a double-edged sword: decreases in communication costs (by increasing the number of local steps) are accompanied by decreases in accuracy. As a result, practitioners interested in post-local SGD must weigh the trade-offs between the training speedup and reduction in the accuracy of the final model. This finding is a departure from the CIFAR-10 scale experiments of Lin et al. (2018), who argued that post-local SGD benefited both communication and accuracy. Looking ahead, our analysis of the interaction between post-local SGD, learning rates, and momentum point toward potential opportunities to escape these trade-offs.

2 Trade-Offs of Local SGD and Post-local SGD

While Local SGD undoubtedly reduces communication and thus overall runtime, the specific speedup varies from task to task. We deem it important to quantify the potential speedup that we can achieve by reducing the communication. Consequently, we illustrate the scalability challenges of minibatch SGD in distributed settings and show that Local SGD and post-local SGD achieve substantially better performance by reducing communication. We evaluate our experiments on the computer vision task of image classification on ImageNet-1k (Russakovsky et al., 2015), details about the experimental setup and implementation details are listed under Appendix A. We consider Local SGD where the model is averaged every H steps from the beginning of training and post-local SGD with a switching point at the first learning rate decay (epoch 30) as suggested in Lin et al. (2018).

Local SGD reduces communication overhead. The main performance cost of scaling minibatch SGD to a distributed setting is the communication overhead to synchronize gradients before updating model weights at every iteration. If this overhead is comparable to the other parts of training, the training process will be significantly slowed, leading to reduced performance. Figure 1 presents a breakdown of the running times of different parts of the training loop. For a single node, i.e., 8 GPUs, the communication time is quite low, indicating that intra-node communication is negligible with respect to other parts of training. However, for multi-node minibatch SGD ($H = 1$ and $K > 8$ in the figure), the communication time dominates the other steps as the number of workers K increases. On the other hand, for Local SGD ($H > 1$ in the figure) as we increase the number of local steps H the communication overhead is amortized over several iterations.

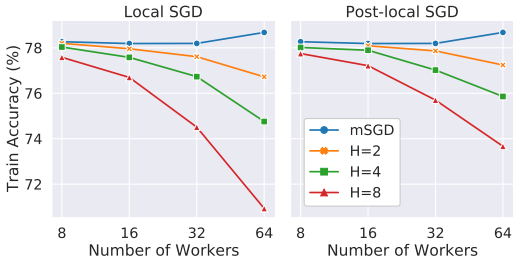

Figure 2: Training Accuracy as a function of the number of workers for local SGD (left) and post-local SGD (right). Separate curves refer to experiments with the same number of local steps H between model averages.

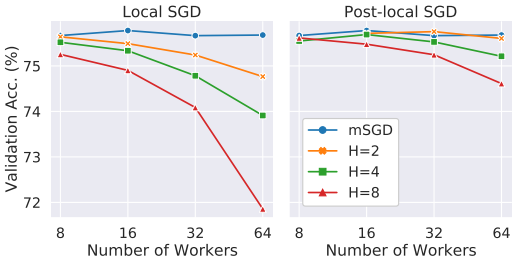

Figure 3: Validation Accuracy as a function of the number of workers for local SGD (left) and post-local SGD (right). Separate curves refer to experiments with the same number of local steps H between model averages.

The measured benefits of reducing communication during the training process suggest that local SGD presents an opportunity to substantially reduce training times. As such, we now analyze the effect on generalization of these techniques to understand whether this performance comes at a cost. We explore the scalability of both local SGD and post-local SGD (switching at epoch 30 unless otherwise noted) as we increase the total number of workers (i.e., GPUs) and as we increase the number of local steps between model averaging. We show that for ResNet-50 and ResNet-101 on ImageNet-1k, both local SGD and post-local SGD struggle to scale in any of these axes without causing a nontrivial drop in validation performance.

Increasing the number of workers. Our first analysis focuses on the trade-off between increasing the number of workers and the effects on generalization as measured by decrease in validation accuracy. We find that increasing workers has a negative effect on accuracy for both local and post-local SGD, consistently leading to worse training and validation results. Figures 2 and 3 show the accuracy for ResNet-50 on the ImageNet-1k training and validation sets as the number of workers increases. In the figures, mSGD corresponds to the minibatch SGD baseline. Neither local SGD and post-local SGD maintain accuracy as the number of workers increases, specially in terms on validation accuracy. Although post-local SGD experiences smaller absolute drops in accuracy compared to local SGD, this comes at the expense of more communication and consequently more runtime.

Reducing synchronization frequency. As we saw from the earlier timing results, reducing the frequency of synchronizations is the main mechanism that local SGD has for reducing the communication overhead. Here, we investigate the effect of increasing the number of local steps between model averaging synchronizations, identifying a very similar trade-off to increasing the number of workers. Figures 2 and 3 show training and validation accuracy for varying numbers of local steps H for both local and post-local SGD. As the number of local steps grows larger, training and validation accuracy monotonically decrease. We observe this phenomena for both local and post-local SGD.

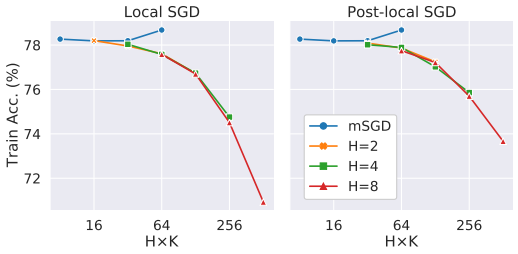

Figure 4: Training Accuracy as a function of the number of local model updates between averages for local SGD (left) and post-local SGD (right).

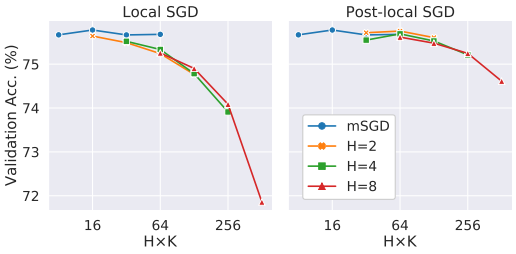

Figure 5: Validation Accuracy as a function of the number of local model updates between averages for local SGD (left) and post-local SGD (right).

Towards a unified trade-off. We showed that increasing the number of workers K or local steps H led to lower accuracy. Interestingly, neither of these factors was clearly dominant and the absolute drop in training and validation accuracy is similar as we increase either of them. We show that this

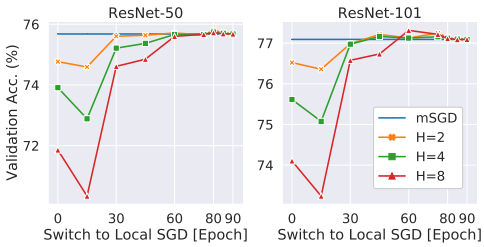

Figure 6: Validation accuracy for post-local SGD as a function of the epoch when the switch to local SGD is done. Results are for $K = 64$ workers and various number of local steps (H).

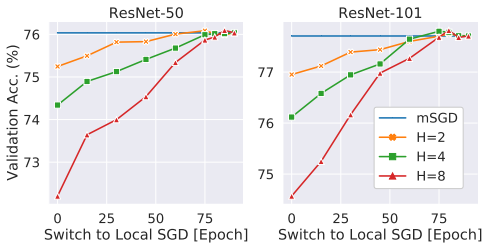

Figure 7: Validation accuracy for post-local SGD as a function of the epoch when the switch to local SGD is done. A 90 epoch half cosine learning rate schedule is used for training.

phenomenon can be better viewed in terms of a different statistic: total number of local model updates between synchronizations, i.e., the product $K \times H$. Figures 4 and 5 show accuracy as a function of the product $K \times H$. When viewed this way, we can observe that the trade-off curves for different numbers of workers and frequencies line up for both training and validation. This suggests that the main factor influencing final model quality is the total number of local model updates between synchronizations. Consequently, in order to maintain model accuracy, an increase in the number of local steps should be matched with a proportional decrease in the number of workers and *vice versa*.

The post-local switching point poses another trade-off. Post-local SGD divides training into two phases. In the first phase, workers perform synchronous minibatch SGD; in the second, they switch to local SGD. However the iteration at which to switch is another hyperparameter that affects both the amount of communication; here, we study whether it also affects final accuracy. Up to this point, all the reported experiments switched to local SGD at epoch 30 out of 90. This is when the first learning rate decay happens with a step-wise learning rate schedule with decays at epochs 30, 60 and 80. This is the recommended switching point from the analysis of Lin et al. (2018). However, switching earlier can reduce the amount of communication and conversely, we might be able to recover some of the lost model accuracy by switching at a later point. To study this phenomena, we selected switching points at epochs where the learning rate decreases (30, 60 and 80) and intermediate epochs (15, 45, 75, and 85). Figure 6 depicts these results for both ResNet-50 and ResNet-101 for various numbers of local steps. Notably, the learning rate decay points seem to have a significant effect into the final performance for this task. Threrefore, we find that the switching point presents a trade-off between training time and final accuracy, with later switching points improving accuracy at the cost of additional communication.

Post-local SGD behavior depends on the task scale. Lin et al. (2018) advertise post-local SGD as a way to remedy the generalization gap present in large batch training (Chen & Huo, 2016; Shallue et al., 2018). However, our observations do not align with the generalization improvements identified by Lin et al. (2018) for ResNet-20 on CIFAR-10 for increasing values of K and H even when exploring different switching points. We have thus identified an example where distributed optimization behaves differently for tasks of different scales. We highlight the fact that the lost accuracy seems to be a direct result of the optimization mechanics of local and post-local SGD since the minibatch baseline does not experience this trade-off as the number of workers increases. In summary, our empirical findings suggest that, in its current form, post-local SGD might not be able to close the generalization gap of large batch training for tasks similar to the one explored in this work.

3 Expanding the Design Space: Learning Rate and Momentum

In the preceding section, we explored the effect of the number of workers, the number of steps between synchronization, and the post-local switching point on the accuracy of models trained with post-local SGD. Regardless of which parameter we varied, we found that post-local SGD involved a tradeoff: measures that improved training time also reduced final accuracy. However, these are only a small number of the numerous hyperparameter choices that must be made when using post-local SGD. If we are to find ways to escape the trade-offs we encounter in Section 2, we will need to consider the role of these additional hyperparameters.

In this section, we focus on the effects of the learning rate schedule and momentum. Learning rates are known to interact with large-batch training (Goyal et al., 2017), and a technique called *slow momentum* (SlowMo; Wang et al., 2020) has improved performance in distributed settings. We find that switching from a step-wise to a half-cosine learning rate schedule indeed affects the performance of post-local SGD, and SlowMo improves performance. These results demonstrate that there may be opportunities to escape the trade-offs from Section 2 by better understanding the effect of learning rate and momentum on post-local SGD.

Post-local SGD depends on learning rate schedule. One unexplored aspect of post-local SGD is how the switching time heuristic interacts with the choice of learning rate schedule. Here, we investigate how the choice of learning rate schedule affects the generalization results of post-local SGD. Recently, learning rate schedules other than step-wise, such as cosine annealing, have gained popularity due to the competitive results they provide without having to carefully specify the learning rate decay points (He et al., 2019; Radosavovic et al., 2019) As an alternative to the step-wise schedule, we consider a half-period cosine schedule which sets the learning rate to $\eta_t = \eta_0(1 + \cos(\pi \cdot t/(T_{\max}))/2$, where η_0 is the initial learning rate, t is the current epoch, and $T_{\max}$ is the total number of epochs. Figure 7 presents a trade-off analysis of the how the switching point affects the final training accuracy of the model but now with a half cosine learning rate schedule. Unlike before, we see a more monotonic trade-off between when the switch to local SGD is performed and the final validation accuracy.

Local SGD as a regularizer. Up to this point, our analysis has focused on the accuracy achieved by post-local SGD. However, the best validation accuracy of the model might not be representative of how local SGD and post-local SGD affect the optimization of the model over the course of training. Here, we study how the loss and the errors evolve over time for different switching points revealing that switching to local-SGD has a regularization effect regardless of when it is performed. In Figure 8, we plot the training and validation error for the minibatch baseline, local SGD and post-local SGD for both learning rate schedules. We observe a regularization effect when the switch to local SGD is performed for the post-local SGD method: the training error is higher than the minibatch baseline while the validation error is lower than the baseline. Interestingly, after switching, the post-local SGD training error closely tracks the local SGD curve despite the 30 epoch difference between the two methods. This suggests that the switch to local SGD has a regularization effect on the model and that, for the second phase of post-local SGD, optimization more closely resembles that of local SGD.

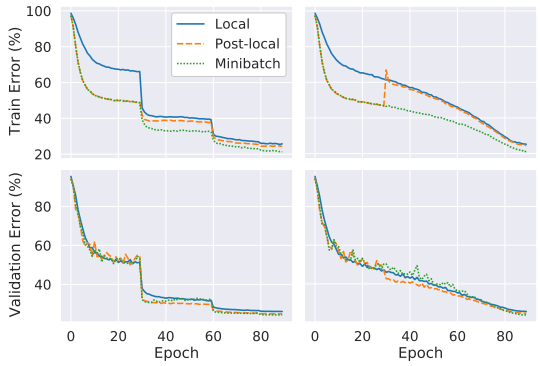

Figure 8: Training and validation error minibatch, local SGD, and post-local SGD for step wise and half cosine learning rate schedules. The left column is for step-wise and the right column is for half cosine. Results are for 64 workers and 4 local steps.

We now extend this analysis for other switching points along the training process in order to determine whether the regularization effect still takes place at earlier or later points in training. This is depicted in Figure 9, which displays the training loss, top-1 training error and top-1 validation error for experiments with 64 workers and 4 local steps between model averages. As discussed earlier, we see the regularization effect that takes places once the switch is performed: the training loss and error increase while the validation error decreases. However, now we observe that see that these patterns hold regardless of when we perform the switch. More interestingly, after the switching point, the curves of the local SGD and post-local SGD models closely follow the same trajectory both for training and validation metrics. This pattern holds for both learning rate schedules although it is more noticeable for the half cosine learning rate schedule because of the higher absolute change. We hypothesize that the change is related to the scale of the learning rate, which is higher for the half cosine schedule than the step-wise schedule for most switching points.

Improving accuracy with slow momentum. The *slow momentum* algorithm (SlowMo Wang et al., 2020) works by adding an additional momentum update after model averaging steps. SlowMo

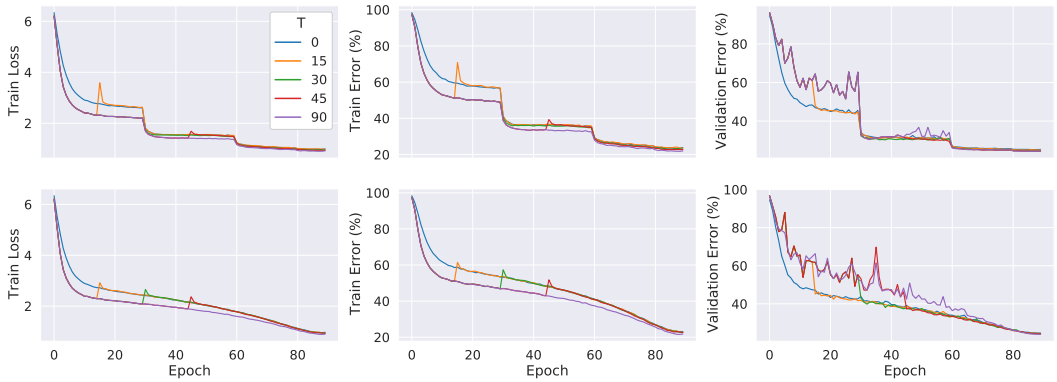

Figure 9: Training Loss (1st column), Training Error (2nd column) and Validation Error (3rd column) for post-local SGD experiments with different switching points T and number of local steps $H = 4$. Top row corresponds to results with step wise learning rate schedule with decays at 30, 60 and 80 epochs and bottom row corresponds to a half cosine schedule.

consistently provided improvements when incorporated into algorithms with reduced communication including local SGD and other decentralized algorithms. SlowMo does not require additional communication since the model averages provide enough information to compute the slow gradients and perform the update. SlowMo does introduce two additional hyperparameters: α, the "slow" learning rate and β, the "slow" momentum coefficient. For computational reasons, we do not sweep the α and β hyperparameters of SlowMo and set them to recommended defaults of $\alpha = 1$ and $\beta = 0.5$ and use the step wise learning rate schedule. We evaluate SlowMo on top of local SGD and post-local SGD with switching points at 15 or 30 epochs. Figure 10 shows that adding the slow momentum updates leads to improved validation accuracy in all cases. Incorporating SlowMo at the switching point at epoch 15 leads to noticeably better accuracy, outperforming the local SGD case and inverting the behavior previously seen.

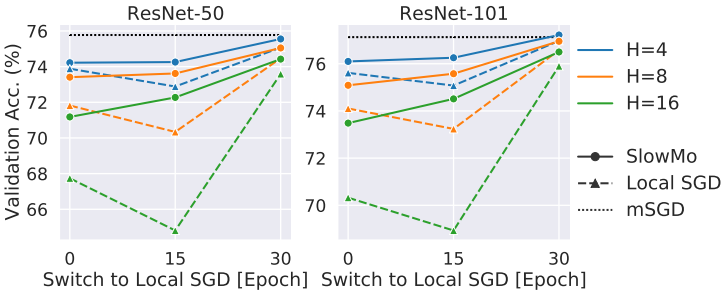

Figure 10: Validation accuracy for various local SGD setting with and without the slow momentum update (SlowMo). SlowMo consistently yields better performance than local or post-local SGD. Results are for 64 workers and step-wise learning rate.

4 Conclusion

In this work, we present a thorough analysis of local and post-local SGD at scale by evaluating them on the ImageNet-1k classification task. We identify several scalability limitations that local and post-local SGD experience and analyze the trade-offs they present in terms of final model accuracy. We find that the number of workers and the number of local steps for local SGD present accuracy trade-offs, and we unify these hyperparameters into a trade-off expressed in terms of number of local model updates between synchronizations. We characterize the behavior of post-local SGD for different switching points along the training process, revealing a similar trade-off between communication and model accuracy. Furthermore, we identify that the choice of learning rate schedule has a large impact for the resulting accuracy of post-local SGD. We find that switching to local SGD has a regularization effect on optimization that is beneficial in the short term but detrimental to final model accuracy. Lastly, we further show that incorporating the slow momentum framework of Wang et al. (2020) consistently improves accuracy without requiring additional communication, hinting at future opportunities for potentially escaping these trade-offs.

References

Assran, M., Loizou, N., Ballas, N., and Rabbat, M. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pp. 344–353. PMLR, 2019.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Chen, K. and Huo, Q. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *2016 ieee international conference on acoustics, speech and signal processing (icassp)*, pp. 5880–5884. IEEE, 2016.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13:165–202, 2012.

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

Dutta, S., Joshi, G., Ghosh, S., Dube, P., and Nagpurkar, P. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. *arXiv preprint arXiv:1803.01113*, 2018.

Ferdinand, N., Al-Lawati, H., Draper, S. C., and Nokleby, M. Anytime minibatch: Exploiting stragglers in online distributed optimization. *arXiv preprint arXiv:2006.05752*, 2020.

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

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

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

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 558–567, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., et al. Pytorch distributed: experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 5330–5340, 2017.

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

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Radosavovic, I., Johnson, J., Xie, S., Lo, W.-Y., and Dollár, P. On network design spaces for visual recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1882–1890, 2019.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

Wang, J., Tantia, V., Ballas, N., and Rabbat, M. Slowmo: Improving communication-efficient distributed sgd with slow momentum. In *International Conference on Learning Representations*, 2020.

Zhang, J., De Sa, C., Mitliagkas, I., and Ré, C. Parallel sgd: When does averaging help? *arXiv preprint arXiv:1606.07365*, 2016.

A Experimental Setup

Our aim in this work is to experimentally investigate the performance of post-local SGD and the related methods mentioned above on a large-scale distributed training workload. Here we describe the experimental setup for the all the results presented in the paper.

Dataset. We focus on the computer vision task of image classification on the ImageNet dataset (Deng et al., 2009; Russakovsky et al., 2015). This is a 1000-way classification task with $\sim$1.28 million training images. Performance is evaluated by top-1 error on 50,000 held out validation images. For the training examples, we follow the standard channel normalization and data augmentation scheme as in He et al. (2016b) and use the default training and validation splits from Russakovsky et al. (2015).

Models. We use the ResNet-50 and ResNet-101 architectures (He et al., 2016a). We initialize the weights as in He et al. (2015) and incorporate the modifications described in Goyal et al. (2017) for initializing the fully connected layer and the last Batch Normalization layer of the residual blocks. We use L2 weight decay with $\lambda = 10^{-4}$.

Training. Following recommendations from Goyal et al. (2017), we scale the learning rate linearly with the global batch size according to $\eta = KB_{\text{loc}}0.1/256$, where K is the number of workers and B_{loc} is the per-worker batch size, which is set to 32. In addition, we apply a learning rate warm-up strategy that starts from 0.1 and linearly increases the learning rate every iteration during the first five epochs of training so that the target learning rate η is reached at the end of the fifth epoch. All experiments are trained with Nesterov momentum (Nesterov, 2013) with $\beta = 0.9$. We also apply momentum correction as described in Goyal et al. (2017) to stabilize training whenever the learning rate value is modified. Unless mentioned otherwise, we use a step wise learning rate schedule that decays the learning rate by a factor of 10 at the end of the 30$^{\text{th}}$, 60$^{\text{th}}$ and 80$^{\text{th}}$ epochs, similar to He et al. (2016a).

Local SGD. For local SGD and post-local SGD, we average the model weights of all the workers before computing the validation performance at the end of every epoch. We consistently found that averaging the models before computing validation was beneficial to generalization performance. This model average is performed outside of the training process. For post-local SGD, before the switch to local SGD is performed, we average the gradients and not the model weights as in Lin et al. (2018). During the post-local SGD regime the per-worker momentum buffers are kept local and not synchronized, this is a common approach (Lian et al., 2017; Assran et al., 2019; Wang et al., 2020).

Platform Details. All methods are implemented using PyTorch 1.6 (Paszke et al., 2019) and we use the ResNet implementations from torchvision 0.7 with CUDA 10.1 and the NCCL communication library. Our experiments run on nodes with eight NVIDIA V100 GPUs each. The nodes communicate over 10 Gb/s Ethernet links. Throughout, a worker refers to a process in a node that makes use of one GPU exclusively. Thus, for the rest of the paper the number of workers is equivalent to the number of GPUs and, in our setup, is equivalent to eight times the number of nodes.

B Background

Minibatch SGD. Neural networks are typically trained with *minibatch SGD*. Let $[N] = \{1, 2, \ldots, N\}$ denote the first N natural numbers. In minibatch SGD, the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i\in[N]}$ is divided into non-overlapping subsets $\{\{(x_{(m-1)B+i}, y_{(m-1)B+i})\}_{i\in[B]}\}_{m\in[\frac{N}{B}]}$ of size B known as *minibatches*. Gradient descent is performed sequentially on these minibatches, passing through the entire dataset over the course of an *epoch*. The dataset is typically randomly shuffled before each epoch, meaning the minibatch composition and order are different on each pass through the dataset. See Algorithm 1 below for full details.

In practice, networks are often trained in a *distributed* data-parallel fashion across K workers (Li et al., 2020). Each worker has a separate copy of the weights $w^{(k)}$ and computes gradients using a disjoint subset of the dataset. The entire dataset is reshuffled and split among workers at the beginning of every epoch. After the backward pass, the models average the gradients across workers before updating the model weights. Distributed training is therefore identical to mini-batch SGD but makes use of multiple workers.

Algorithm 1 Minibatch SGD with learning rate schedule $\gamma(t)$, batch size B, initial weights w_0, and loss ℓ.

1: $S \leftarrow \lceil \frac{N}{B} \rceil$ (iterations per epoch)
2: **for** each epoch e **do**
3: Shuffle dataset $\mathcal{D} = \{x_i, y_i\}_{i \in N}$
4: **for** each iteration m in 1 to S **do**
5: $t \leftarrow eS + m$ (the current step of training)
6: $o \leftarrow (m-1)B$
7: $w_t \leftarrow w_{t-1} - \gamma(t) \frac{1}{B} \sum_{i=1}^{B} \nabla \ell(f(x_{o+i}; w_t), y_{o+i})$

Local SGD. In local SGD, described in Algorithm 2, the workers update the local copies of their weights, and every $H > 1$ iterations they *synchronize* the weights across workers by averaging the weights stored on each worker.

Algorithm 2 Local SGD with learning rate schedule $\gamma(t)$, batch size B, initial weights w_0, loss ℓ, and R workers.

1: $S \leftarrow \lceil \frac{N}{BK} \rceil$ (iterations per epoch)
2: **for** each epoch e **do**
3: Shuffle dataset $\mathcal{D} = \{x_i, y_i\}_{i \in N}$
4: **for** each iteration m in 1 to S, at worker k (in parallel) **do**
5: $t \leftarrow eS + m$ (the current step of training)
6: $o \leftarrow (m-1)B + (k-1)SB$
7: $w_t^{(k)} \leftarrow w_{t-1}^{(k)} - \gamma(t) \frac{1}{B} \sum_{i=1}^{B} \nabla \ell(f(x_{o+i}; w_t), y_{o+i})$
8: **if** $t \bmod H = 0$ **then**
9: $w_t^{(k)} \leftarrow \frac{1}{K} \sum_{k=1}^{K} w_t^{(k)}$ (synchronize)

Post-local SGD. Post-local SGD involves performing synchronous SGD for the first T steps of training and local SGD thereafter. The hyperparameters may differ between the synchronous and local phases. For example, the local phase may use a different learning rate or batch size.

Slow momentum. The slow momentum method (Wang et al., 2020), shown in Algorithm 3, builds on top of the BMUF method (Chen & Huo, 2016) by generalizing local SGD in a way similar to how SGD with momentum generalizes SGD. In particular, rather than simply averaging the weights every H steps, as in line 9 of Algorithm 2, the difference $w_{t-H}^{(k)} - w_t^{(k)}$ is treated as a pseudo-gradient (or "slow" gradient) and used in a typical momentum update. Wang et al. (2020) previously found that adding slow momentum improves the optimization performance in comparison to local SGD, at the cost of additional memory overhead for the slow momentum buffer $u_t^{(k)}$ and parameters from H steps back $w_{t-H}^{(r)}$.

Algorithm 3 Slow Momentum optimizer with fast learning rate schedule γ_t, slow learning rate schedule α_t, slow momentum parameter β, batch size B, initial weights w_0, loss ℓ, and K workers.

$S \leftarrow \lceil \frac{N}{BK} \rceil$ (iterations per epoch)
$u_0^{(k)} \leftarrow 0$ for all $k \in \{1, \ldots, K\}$ (initialize slowmo buffer)
for each epoch e **do**
    Shuffle dataset $\mathcal{D} = \{x_i, y_i\}_{i \in N}$
    for each iteration m in 1 to S, at worker k (in parallel) **do**
        $t \leftarrow eS + m$ (the current step of training)
        $o \leftarrow (m-1)B + (k-1)SB$
        $w_t^{(k)} \leftarrow w_{t-1}^{(k)} - \gamma(t) \frac{1}{B} \sum_{i=1}^{B} \nabla \ell(f(x_{o+i}; w_t), y_{o+i})$
        if $t \bmod H = 0$ **then**
            $w_t^{(k)} \leftarrow \frac{1}{R} \sum_{r=1}^{R} w_t^{(k)}$ (synchronize)
            $u_t^{(k)} \leftarrow \beta u_{t-H}^{(k)} + \frac{1}{\gamma(t)}(w_{t-H}^{(k)} - w_t^{(k)})$
            $w_t^{(k)} = w_{t-H}^{(k)} - \alpha_t \gamma_t u_t^{(k)}$