DATA PARALLELISM IN TRAINING SPARSE NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Network pruning is an effective methodology to compress large neural networks, and sparse neural networks obtained by pruning can benefit from their reduced memory and computational costs at use. Notably, recent advances have found that it is possible to find a trainable sparse neural network even at random initialization prior to training; hence the obtained sparse network only needs to be trained. While this approach of pruning at initialization turned out to be highly effective, little has been studied about the training aspects of these sparse neural networks. In this work, we focus on measuring the effects of data parallelism on training sparse neural networks. As a result, we find that the data parallelism in training sparse neural networks is no worse than that in training densely parameterized neural networks, despite the general difficulty of training sparse neural networks. When training sparse networks using SGD with momentum, the breakdown of the perfect scaling regime occurs even much later than the dense at large batch sizes.

1 INTRODUCTION

Deep learning has been of great interest in recent years, resulting in tremendous success in various applications. Behind its success lie powerful yet often extremely large neural network models which, however, entail a significant cost of processing at use. As such cost can be critical to resource constrained environments, a diverse set of principles and approaches have been developed to compress large neural networks, and *network pruning* – a technique to remove redundant parameters in an overly parameterized neural network – has been widely employed (Reed, 1993; Han et al., 2015).

While there exist various approaches to performing pruning, recent studies discovered that pruning can be done on a randomly initialized neural network prior to training, and the sparse neural network resulted from pruning can simply be trained to achieve good performance (Lee et al., 2019; Wang et al., 2020). Notably, by disentangling the pruning process from training entirely, this process of *pruning at initialization* not only realizes algorithmic efficiency, but also can save a significant amount of effort and time in finding a trainable sparse neural network, which can be of high practical value in environments with limited resources.

However, little has been studied about the training aspects of sparse neural networks. For example, while Evci et al. (2019) examined the difficulty of training sparse neural networks by analyzing the energy landscape in optimization dynamics, Lee et al. (2020) suggested a signal propagation perspective to speed up the training of sparse neural networks. Despite their potential, various aspects of the optimization of sparse neural networks remain rather unknown as of yet.

In this work, we focus on measuring the effects of *data parallelism* on training these sparse neural networks. By data parallelism we refer to utilizing a parallel computing system, where the training data is distributed to multiple processors for gradient computations, so that the training process can be accelerated. As with the development of cost-effective parallel hardware, understanding the effects of data parallelism (or equivalently increasing the batch size) on neural network training is crucial, and in fact, has been an active research topic in recent years (Goyal et al., 2017; Hoffer et al., 2017; Smith et al., 2018; Chen et al., 2018; Golmant et al., 2018; Shallue et al., 2018). Nonetheless, we identify the lack of investigation of its benefits and limits on sparse neural network training, so here we provide empirical evaluation results, while carefully tuning metaparameters independently for each and every study case.

We found out that the relationship between batch size and number of steps to reach a goal error has a similar scaling pattern of data parallelism observed in densely parameterized network training, across varying sparsity levels and different workloads of data set, model and optimization algorithm.

2 EXPERIMENTS

2.1 SETUP

Experiment protocol. We closely follow the experiment protocol presented in Shallue et al. (2018). For each study (*i.e.*, batch size, sparsity level) and workload (*i.e.*, data set, model, optimization algorithm), we measure the number of training steps to achieve a goal validation error, or *steps-to-result*, as the primary quantity of interest. For each study on a workload, we record the lowest steps-to-result while searching the best metaparameters within a budget of trials of 100 runs.

Workload. We consider the following workloads of data set, model and optimization algorithm: (W1) Simple-CNN on MNIST trained using SGD optimizer with a constant learning rate; (W2) Simple-CNN on MNIST trained using SGD with momentum (Momentum) or Nesterov momentum (Nesterov) optimizer with a constant learning rate; (W3) ResNet-8 on CIFAR-10 trained using the Nesterov optimizer with a linear learning rate decay schedule. All metaparameters involved in these workloads (*i.e.*, learning rate, momentum, decay steps and decay factor) are tuned independently for each study by a quasi-random search. The goal validation errors are set to be 0.02 and 0.4 for W1/W2 and W3, respectively. We remove dropout in Simple-CNN to avoid any effect on sparsification. We refer to Shallue et al. (2018) for further architecture details.

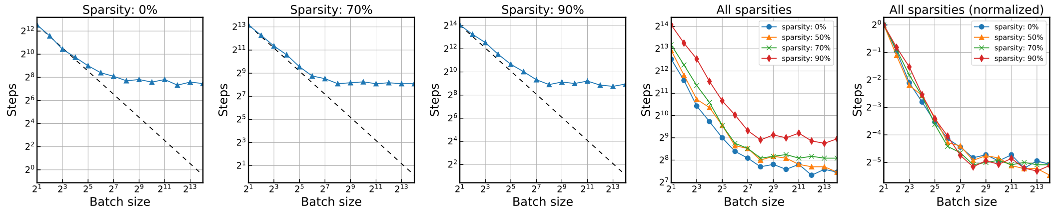
Metaparameter search. We perform a quasi-random search to tune metaparameters efficiently. Specifically, we first generate Sobol low-discrepancy sequences in a unit hypercube and convert them into metaparameters of interest, while taking into account a predefined search space for each metaparameter. The generated values for each metaparameter is length of the budget of trials (*i.e.*, 100) for all studies, and the search space is designed based on preliminary experimental results.

Pruning. Sparse neural networks can be obtained by many different ways, and yet, for the purpose of this work, they must not undergo any training so we can measure the effects of data parallelism on training sparse neural networks in isolation. Adopting the connection sensitivity saliency criterion suggested in Lee et al. (2019), we prune a given network at initialization and train the resulting sparse neural network from scratch.

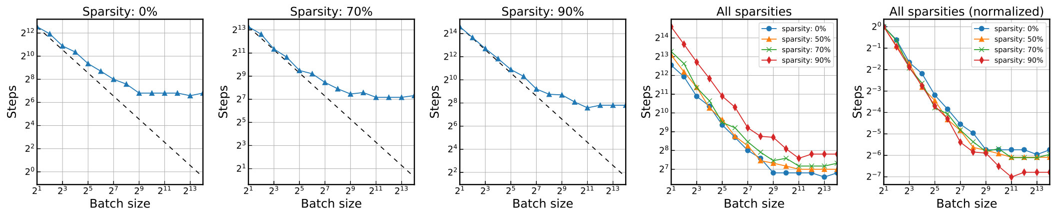
2.2 RESULTS: DATA PARALLELISM IN TRAINING SPARSE NEURAL NETWORKS

First of all, we observe from all cases of sparsity and optimization algorithm the same relationship between steps-to-result and batch size (1st, 2nd, 3rd columns in Figure 1): Initially, we observe a period of *perfect scaling* where doubling the batch size reduces the steps to achieve the goal error by half (*i.e.*, it aligns closely with the dashed line), followed by a region of *diminishing return* where the reduction in the required number of steps by increasing the batch size is less than the inverse proportional amount (*i.e.*, it starts to digress from the perfect scaling region), which eventually arrives at a *maximal data parallelism* (*i.e.*, it hits a plateau) where increasing the batch size no longer decrease the steps to reach a goal error. This is observed across different sparsity levels and optimization algorithms as well as data sets and learning rate decay schedule (Figure 3). Our observation is consistent with the case of dense network training presented in Shallue et al. (2018).

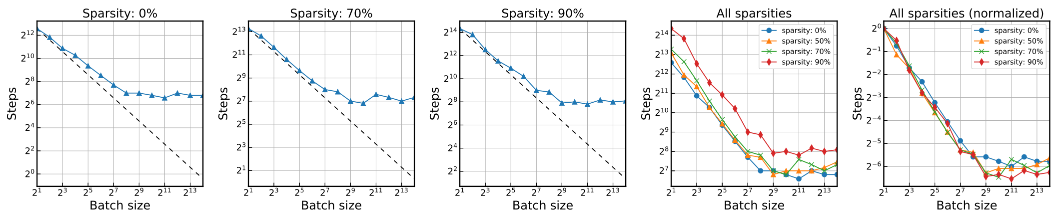
When we put all sparsity results together, the phase transitions of data parallelism are clearly visible and compared (4th column in Figure 1). It appears that the higher the sparsity, the longer it takes for a sparse network to hit the goal error in general; *i.e.*, a curve of higher sparsity usually lies above that of lower sparsity; *e.g.*, compared to the case of sparsity 0% (*i.e.*, the dense unpruned network), 90% sparse network takes about $2^{1.5}$ times longer training time (or the number of training steps), consistently across different batch sizes (vertical shift with a similar slope), until each sparsity case hits its own region of diminishing returns. Recall that we tune the metaparameters (*i.e.*, learning rate in this case) independently for all cases without relying on a single predefined training rule. Therefore, this result on the one hand corroborates the general difficulty of training sparse neural networks, against the ease of training overly parameterized neural networks.



(a) SGD



(b) Momentum



(c) Nesterov

Figure 1: Results of measuring the data parallelism in sparse neural network training using Simple-CNN on MNIST. We first prune the network for {0, 50, 70, 90}% sparsity levels, train using either SGD, SGD with momentum (Momentum), or SGD with Nesterov momentum (Nesterov) optimizer, and then record steps-to-result to achieve the goal error of 0.02. Across all sparsity levels and optimizers, the same scaling pattern is observed for the relationship between batch size and steps-to-result: increasing the batch size initially decreases the steps required to achieve the goal error proportionally (*perfect scaling*), and then there begins a region of *diminishing returns*, followed by the phase of *maximal data parallelism* where increasing the batch size no longer speeds up the training. Notably, the data parallelism in sparse neural network training is no worse than regular neural network training, despite the general difficulty of training sparse networks. More interestingly, when training with momentum, the breakdown of the perfect scaling regime occurs much later at larger batch sizes for a higher sparsity network; *i.e.*, a *critical batch size* can be even larger when training sparse neural networks (*e.g.*, 2^{11} for the sparsity 90% network compared to 2^9 for the sparsity 0% network when using Momentum). This potentially indicates that one can exploit large batch sizes more effectively when training sparse neural networks than regular neural networks.

On the other hand, when we normalize the y-axis of each plot by dividing by the number of steps for the first batch size, we can see that the regions of diminishing returns and maximal data parallelism appear no earlier when training sparse networks than the dense network (5th column in Figure 1). This is quite surprising in that one could have easily guessed that the general optimization difficulty incurred by sparsification may influence the data parallelism too, at least to some degree; however, it turns out that the effects of data parallelism on sparse network training remain no worse than the dense case. Moreover, notice that in some cases the breakdown of perfect scaling regime occurs even much later at larger batch sizes for a higher sparsity case; this is especially evident for Momentum and Nesterov optimzicers (*e.g.*, compare training 90% sparse network using Momentum against 0% dense network). In other words, for sparse networks, a *critical batch size* can be larger, and hence, when it comes to training sparse neural networks, one can increase the batch size (or design a parallel computing for distributed optimization) more effectively, while better exploiting given resources. We find this result particularly promising since SGD with momentum is often the method of choice in practice. We further show that momentum optimizers being capable of exploiting large batch sizes hold the same across different sparsity levels by displaying all plots togeter in Figure 2. Overall, we believe that it is positively important to confirm the robustness of the data parallelism in sparse neural network training, which has been unknown thus far and difficult to estimate a priori.

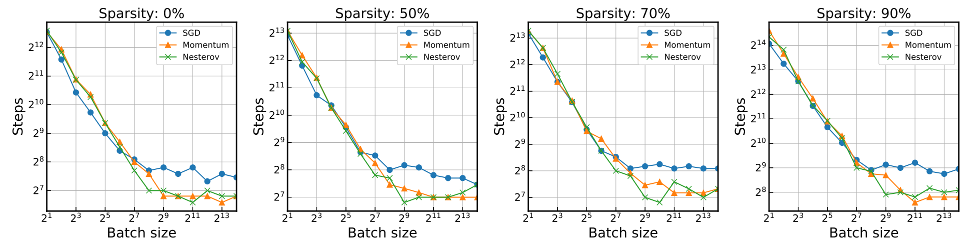


Figure 2: Comparing the relationship between batch size and steps-to-result by different optimizers. For all sparsity levels, Momentum and Nesterov optimizers are able to exploit much larger batch sizes than the plain SGD optimizer. Identifying such pattern is crucial especially when training in resource constrained enviroments, as practitioners can benefit from reducing the training time by deciding a critical batch size properly, while utilizing resources more effectively.

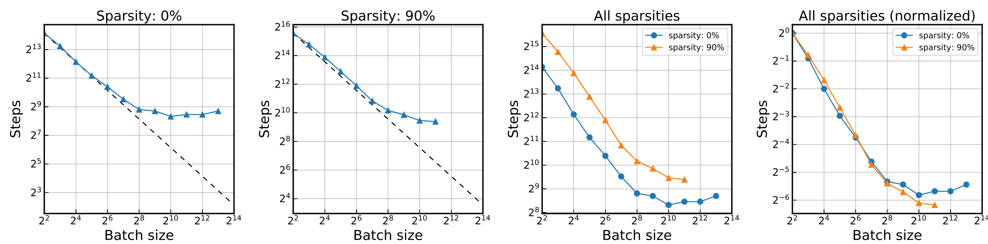


Figure 3: Results of measuring the data parallelism in sparse neural network training using ResNet-8 on CIFAR-10. We prune the network for $\{0, 90\}$% sparsity levels, train using Nesterov optimizer with linear learning rate decay, and then record steps-to-result to achieve the goal error of 0.4. A scaling pattern of data parallelism observed here is similar to that of the previous workloads; *i.e.*, a critical batch size for the sparse network appears to be larger than that for the dense network.

2.3 METAPARAMETER SEARCH

In this section, we present the metaparameter search results used for investigating the relationship between batch size and steps-to-result. We perform a quasi-random search to find the lowest steps-to-result, while tuning the best metaparameters independently for each study and workload.

We first visualize all trials for the workload of Simple-CNN on MNIST trained using the Momentum optimizer with a constant learning rate (Figure 4). We display eight studies for combinations of sparsity levels (S: 0, 90%) and batch sizes (B: 16, 128, 1024, 8192). For all studies, the search spaces for learning rate and momentum metaparameters are set reasonably well; the best learning rate is found in the middle of the search space, while the momentum parameter is located around a bit larger than 0.9. We also see that as the batch size increases, the more complete trials (blue circles) and the less incomplete trials (grey triangle) are recorded, for both the dense and sparse neural networks. This potentially indicates that the data parallelism in sparse network training remain the same as or similar to that in regular neural network training. We further present the metaparameter search results for the workload of ResNet-8 on CIFAR-10 trained using the Nesterov optimizer with a linear learning rate decay schedule, for each metaparameter separately in Figure 5.

3 CONCLUSION

In this work, we measure the effects of data parallelism on sparse neural network training. Based on our results of various cases of training sparse neural networks, we found out that the data parallelism in training sparse neural networks remain no worse, or in some cases, even better than that in training densely parameterized neural networks. We believe that our findings render a positive impact on the community, by potentially helping practitioners to utilize resources more effectively.

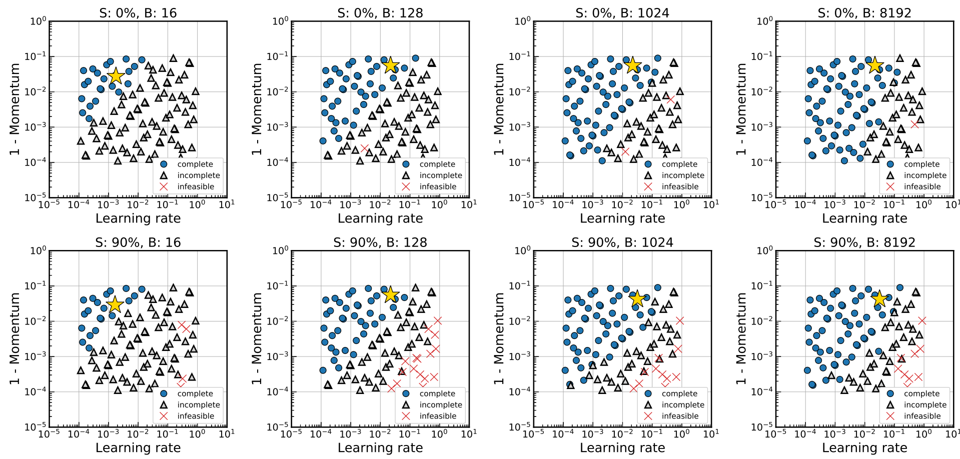


Figure 4: Visualizing all trials (100) for Simple-CNN on MNIST trained using Momentum optimizer with a constant learning rate. Sparsity level (S) and batch size (B) are denoted at the top of each plot. The best trial that records the lowest steps-to-result is marked by gold star (⋆). Complete/incomplete refer to the trials of goal reached/not reached given a maximum training step budget, while infeasible refers to the trial of divergence during training.

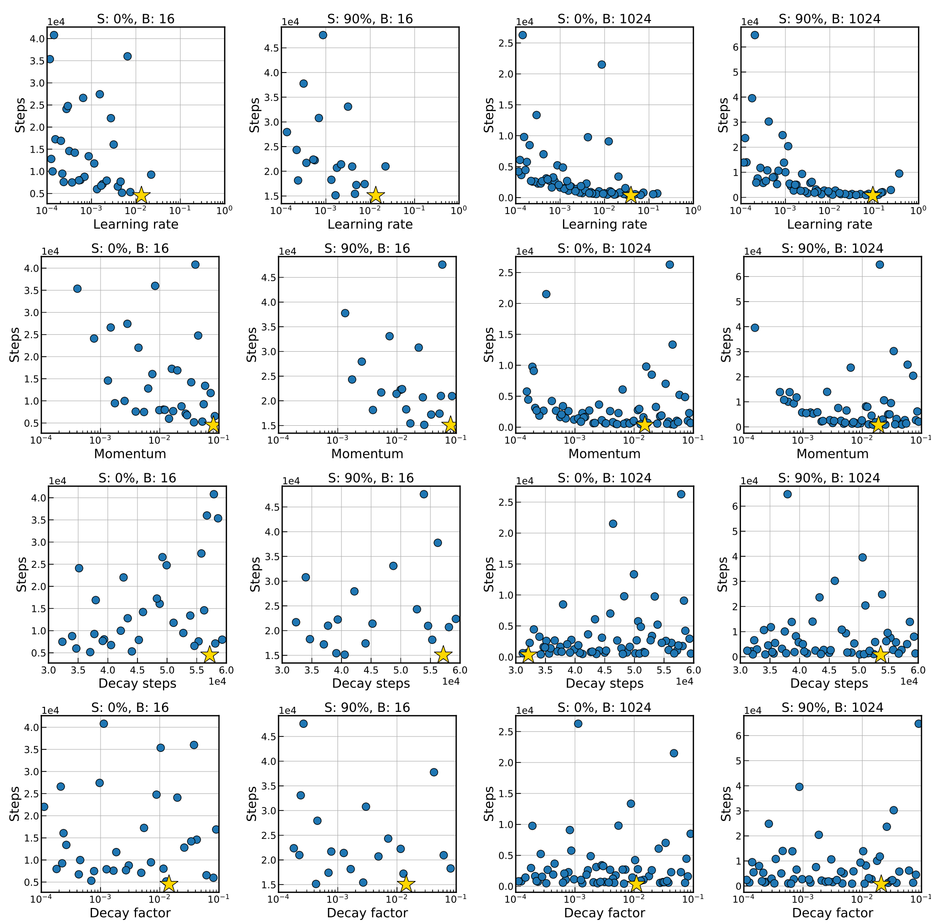


Figure 5: Visualizing the metaparameter search results for ResNet-8 on CIFAR-10 trained using the Nesterov optimizer with a linear learning rate decay schedule. Sparsity level (S) and batch size (B) are denoted at the top of each plot. The best trial that records the lowest steps-to-result is marked by gold star (⋆).

REFERENCES

Lingjiao Chen, Hongyi Wang, Jinman Zhao, Dimitris Papailiopoulos, and Paraschos Koutris. The effect of network width on the performance of large-batch training. In *NeurIPS*, 2018.

Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *arXiv preprint arXiv:1906.10732*, 2019.

Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *NeurIPS*, 2017.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *ICLR*, 2020.

Russell Reed. Pruning algorithms-a survey. *Neural Networks*, 1993.

Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *JMLR*, 2018.

Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. In *ICLR*, 2018.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.