Understanding the Effects of Data Parallelism and Sparsity on Neural Network Training

Namhoon Lee[*] University of Oxford
Thalaiyasingam Ajanthan Australian National University
Philip H. S. Torr University of Oxford
Martin Jaggi EPFL

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 studies have found that it is possible to find a trainable sparse neural network even at random initialization prior to training. While this approach of pruning at initialization turned out to be highly effective, there has been little study concerning the subsequent training of these sparse neural networks. In this work, we focus on studying the effects of data parallelism and sparsity on neural network training. For data parallelism, this usually means processing training data in parallel using distributed systems, or equivalently increasing batch size, so that the training process can be accelerated. To this end, we first measure the effects for different study cases of batch size and sparsity level while tuning all metaparameters involved in the optimization. As a result, we find across various workloads of data set, network model, and optimization algorithms that there exists a general scaling trend in the relationship between batch size and number of training steps to convergence for the effect of data parallelism, irrespective of sparsity levels. Also, the effect of data parallelism in training sparse networks turns out to be no worse, or can be even better when the training is done by a momentum based optimizer, than that in training densely parameterized networks, despite the general difficulty of training sparse networks. We further provide theoretical insights based on the convergence properties of stochastic gradient methods and a smoothness analysis, so as to precisely illustrate our empirical findings and hence to develop a better account of the effects of data parallelism and sparsity on neural network training.

1 Introduction

Deep learning has been of great interest in recent years, bringing about tremendous success in science and technology [11]. Behind its success lie powerful, yet often extremely large neural network models however, and they entail a significant amount of processing cost 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 network models, and network pruning – a technique to remove redundant parameters in an overly parameterized neural network – has been widely employed [15, 9].

While there exist various approaches to pruning, recent studies discovered that pruning can be done on a randomly initialized neural network prior to training, and the sparse neural network obtained can be easily trained to achieve good performance [12, 19]. By separating 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 time and effort in finding a trainable sparse neural network.

[*]Correspondence to namhoon@robots.ox.ac.uk

Preprint. Under review.

However, there has been little study of the training aspects of sparse neural networks. For example, while Evci et al. [5] examined the difficulty of training sparse neural networks by analyzing the energy landscape in optimization dynamics, Lee et al. [13] 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 studying 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 batch size) on neural network training is crucial, and in fact, it has been an active research topic in recent years [8, 10, 18, 3, 7, 17]. While we realize that sparse networks can benefit from data parallelism in distributed or large-batch training due to their reduced number of parameters and cost, there has been a lack of investigation of the effects of data parallelism on sparse neural network training. In this regard, we first conduct extensive experiments to measure the effects of data parallelism on sparse neural network training while carefully tuning metaparameters independently for each and every study case. As a result, we find a general trend of effect of data parallelism in training sparse neural networks, across varying sparsity levels and different workloads of data set, model and optimization algorithms, that turns out to be no worse or can be better than that in training densely parameterized networks, despite the difficulty of training sparse networks. Based on the convergence properties of stochastic gradient methods as well as an analysis on Lipschitz smoothness of sparse networks, we further present theoretical results that describe precisely the effects of data parallelism and sparsity on neural network training to this end.

1.1 Contributions

We summarize our main contributions as follows:

- We conduct a substantial amount of experiments based on extensive metaparameter search, and measure the effects of data parallelism for training neural networks while taking sparsity into account. Our results confirm that there exists a universal trend in the relationship between batch size and steps-to-result for the effect of data parallelism irrespective of sparsity levels.
- We also find that sparsity in general reduces the training speed approximately by the same factor across different batch sizes. Interestingly, however, the critical batch size for training sparse neural networks can be significantly increased by using a momentum based SGD.
- We provide theoretical insights into the effects of data parallelism and sparsity in training neural networks, that are established based on the convergence properties of stochastic gradient methods. Our theory precisely describes our observations, and hence, allows us to develop a better account of the effect of data parallelism. Combined with empirical analysis of Lipschitz smoothness, it further provides guidance on understanding the general difficulty of training sparse neural networks.

In light of our contributions, we make a further note that it has been considered difficult to obtain such results or estimate a priori. Therefore, we will release all our data points measured along with our code used in this work to the public in the hope of facilitating future research.

2 Measuring the effects of data parallelism and sparsity

2.1 Setup

Experiment protocol. We closely follow the experiment settings in [17]. For a given workload (*i.e.*, data set, model, optimization algorithm) and study (*i.e.*, batch size, sparsity level), we measure the number of training steps to reach a pre-defined goal error (*e.g.*, error rate for image classification). In this process, we search for the best metaparameters with a budget of runs to record the lowest number of steps, namely *steps-to-result*, as our primary quantity of interest. For each run of training, we evaluate the intermediate model checkpoints on the entire validation set of a given data set at every fixed iterations to check if they reached the goal error. We describe more details and the scale of our experiments in Appendix B.

Workloads. We consider the workloads constructed as the combinations of the following data sets, models, and optimization algorithms: (data sets) MNIST, Fashion-MNIST, CIFAR-10; (models) Simple-CNN, ResNet-8; (optimization algorithms) SGD, Momentum, Nesterov with either a constant learning rate or a linear learning rate decay schedule. In the main paper, we present results for the following four workloads: (W1) MNIST, Simple-CNN, SGD with a constant learning

rate, (W2) MNIST, Simple-CNN, Momentum with a constant learning rate, (W3) Fashion-MNIST, Simple-CNN, Momentum with a constant learning rate, and (W4) CIFAR-10, ResNet-8, Nesterov with a linear learning rate decay; results for other workloads of combinations of above are provided in Appendix D. We use 10% of the training set as a validation set, and the goal validation errors are set to be 0.02, 0.12, 0.4 for MNIST, Fashion-MNIST, CIFAR-10, respectively. The network models are the same as [17], except that we remove dropout to prevent any effect on sparsification.

Metaparameter search. We perform a quasi-random search to tune metaparameters efficiently. More precisely, 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 in length of the budget of trials, and the search space is designed based on preliminary experimental results.

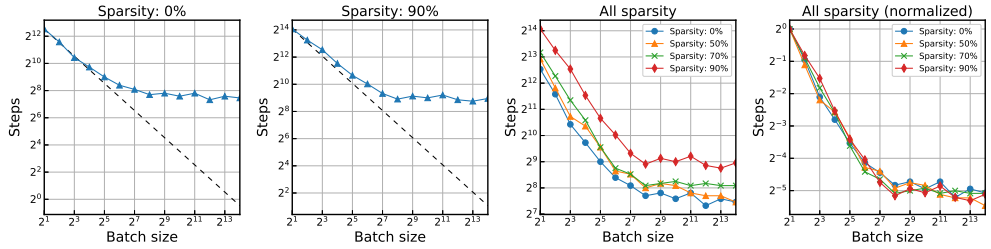
Pruning. We use the connection sensitivity saliency criterion in [12] to prune a given network model at initialization prior to training. Sparse neural networks can be obtained by many different ways, and yet, for the purpose of this work, they must not undergo any training beforehand so we can measure the effects of data parallelism in isolation on training neural networks from scratch.

2.2 Results

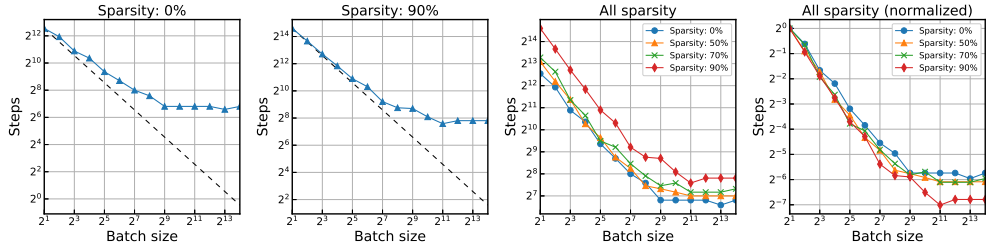
First of all, we observe in each and every sparsity level across different workloads a general trend in the relationship between batch size and steps-to-result for the effects of data parallelism (see the 1st and 2nd columns in Figure 1): Initially, we observe a period of *linear scaling* where doubling 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 returns* 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 linear scaling region), which eventually arrives at a *maximal data parallelism* (*i.e.*, it hits a plateau) where increasing the batch size no longer decreases the steps to reach a goal error. This is observed across other workloads of different combinations of data set, network model, and optimization algorithm as well as different goal errors (see Appendix D). We note that our observation is consistent with the results of regular network training presented in [17]. We develop a theory of the effect of data parallelism that precisely accounts for this universal phenomenon in Section 3.1.

When we put the results for all sparsity levels together, the effect of data parallelism can be clearly visible and compared (see the 3rd column in Figure 1). In general, we find that the higher the sparsity level, the longer it takes for a sparse network to reach the given goal error. More precisely, a data parallelism curve for higher sparsity usually lies above than that for lower sparsity. For example, compared to the case of sparsity 0% (*i.e.*, the dense, over-parameterized network), 90% sparse network takes about $2-4$ times longer training time (or the number of training steps), consistently across different batch sizes (we provide more precise comparisons in Figure 12, Appendix D.1). Recall that we tune all metaparameters independently for each and every study case of batch size and sparsity level without relying on a single predefined training rule, so as to find the best steps-to-result. 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. We further find a potential cause of this difficulty based on our theory and a Lipschitz smoothness analysis in Section 3.2.

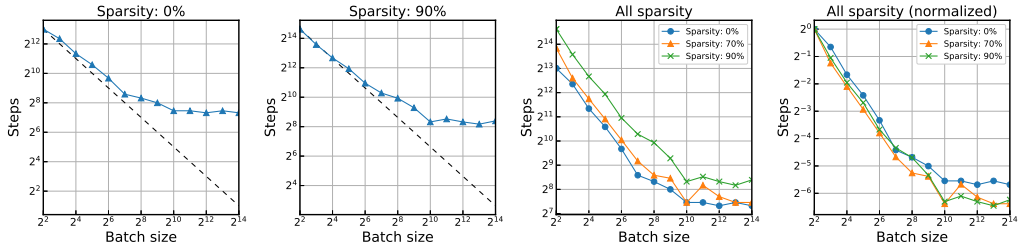
On the other hand, when we normalize the y-axis of each plot by dividing by the number of steps for the first batch size, we can see the phase transitions more clearly. As a result, we find that the regions of diminishing returns and maximal data parallelism appear no earlier when training sparse networks than the dense network (see the 4th 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 many cases the breakdown of linear scaling regime occurs even much later at larger batch sizes for a higher sparsity case; this is especially evident for Momentum and Nesterov optimizers (*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 system 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.



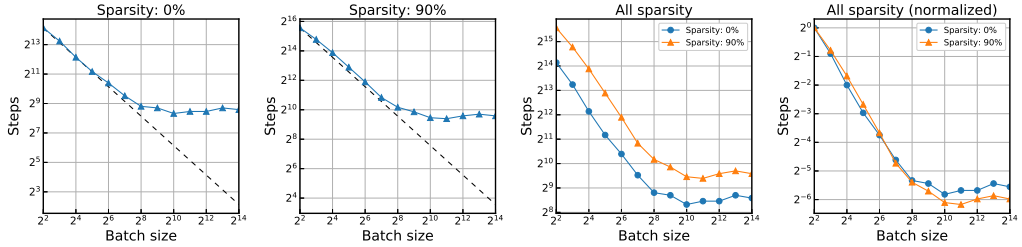
(a) MNIST, Simple-CNN, SGD



(b) MNIST, Simple-CNN, Momentum



(c) Fashion-MNIST, Simple-CNN, Momentum



(d) CIFAR-10, ResNet-8, Nesterov

Figure 1: The effects of data parallelism and sparsity on neural network training for various workloads. Across all workloads and sparsity levels, the same scaling pattern is observed for the relationship between batch size and steps-to-result: it starts with the initial phase of *linear scaling* where increasing the batch size decreases the steps required to achieve the goal error inverse proportionally, and then there begins a region of *diminishing returns*, which is followed by the saturating phase of *maximal data parallelism* where increasing the batch size no longer speeds up the training process. Also, the effect of data parallelism in training sparse neural networks is no worse than that the dense, over-parameterized counterpart, despite the general difficulty of training sparse neural networks. When training using a *momentum* based SGD (*e.g.*, Momentum, Nesterov), the breakdown of the linear scaling regime often occurs much later at larger batch sizes for a network with higher sparsity; *i.e.*, a *critical batch size* can be even larger when training sparse neural networks with momentum. For example, in the case of workload {MNIST, Simple-CNN, Momentum}, the critical batch size for the sparsity 90% network is around 2^{11} whereas it is 2^{9} for the sparsity 0% network (see the 4th column in row (b)). This potentially indicates that one can exploit large batch sizes more effectively when training sparse neural networks than densely parameterized neural networks. We supply more results in Appendix D.

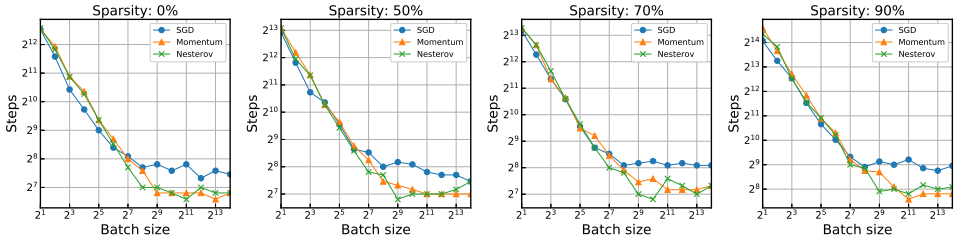


Figure 2: Comparing different optimization algorithms (SGD, Momentum, Nesterov) for the effects of data parallelism and sparsity. Across all sparsity levels, momentum based SGD optimizers (*i.e.*, Momentum, Nesterov) record lower steps-to-result in a large batch regime and have much bigger critical batch sizes than SGD without momentum. Identifying such patterns is crucial especially when training in resource constrained environments, as practitioners can potentially benefit from reducing the training time by deciding a critical batch size properly, while utilizing resources more effectively.

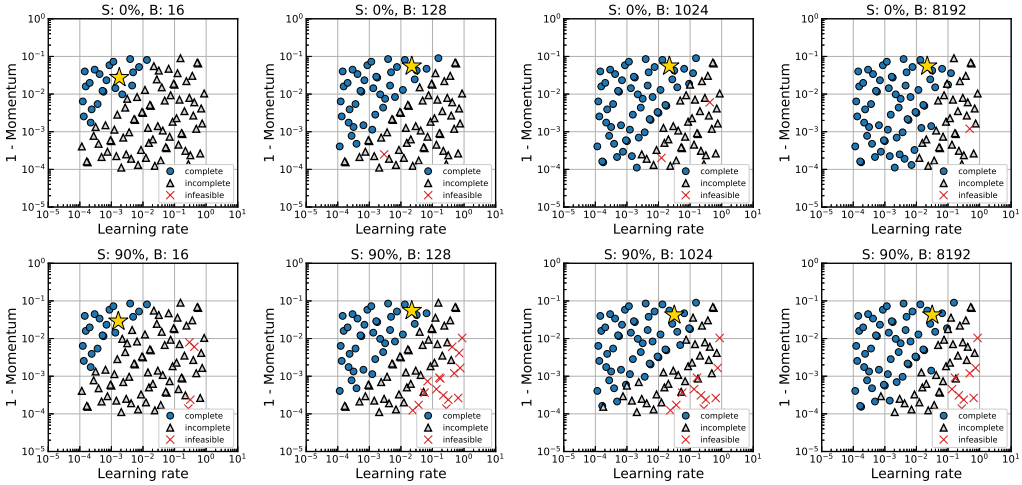


Figure 3: Visualizing all trials (100) for Simple-CNN on MNIST trained using Momentum optimizer with a constant learning rate. Sparsity level (S) and batch size (B) are denoted at the top of each plot. The best trial that records the lowest steps-to-result is marked by gold star (⋆). Complete/incomplete 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.

We further show that momentum optimizers being capable of exploiting large batch sizes hold the same across different sparsity levels by displaying all plots together 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.

2.3 Metaparameter search

We perform metaparameter search to find the best metaparameter values that yield steps-to-result. In this section, we present and analyze the metaparameter search results used to measure the effects of data parallelism studied in the previous section. Specifically, we investigate the case of workload {MNIST, Simple-CNN, Momentum} where there are two metaparameters to tune (*i.e.*, learning rate and momentum) as this allows us to analyze all metaparameters easily in one figure. The results are presented in Figure 3, and other results can be found in Appendix D.2.

First of all, our quasi-random search enables us to sample metaparameters efficiently, so that they are distributed evenly (without being cluttered) in a log-space and flexibly (rather than sitting in a grid with fixed spacing) within the search spaces. The search spaces are determined based on preliminary experiments and considered to be designed reasonably, in that the best metaparameters

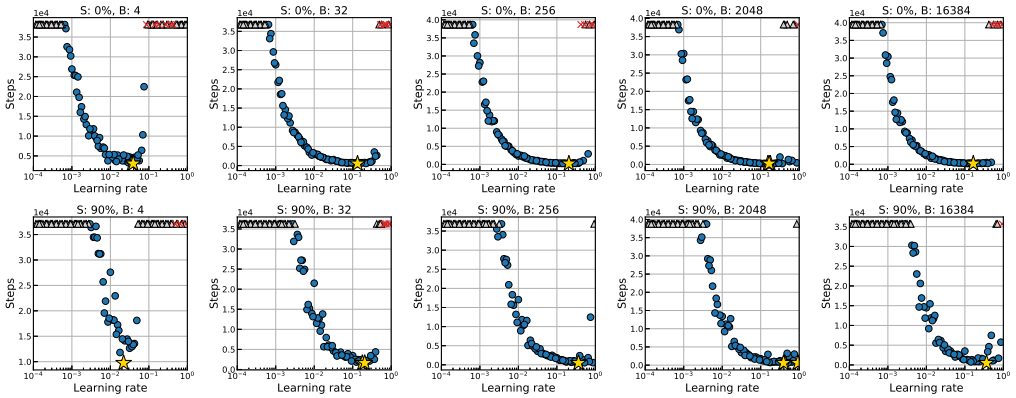


Figure 4: Visualizing metaparameter search results for the workload of {MNIST, Simple-CNN, SGD}. The metaparameter being tuned is the learning rate $\bar{\eta}$. We present results for different batch sizes (2^2, 2^5, 2^8, 2^{11}, 2^{14} – columns) and sparsity levels (0, 90% – rows). The blue circles (•) denote successful runs (*i.e.*, it reached the goal error), and the best trial that records the *lowest* steps to reach the goal (*i.e.*, steps-to-result) is marked by gold star (⋆); also, the grey triangles (▲) and red crosses (×) refer to incomplete and infeasible runs, respectively. We supply more results in Appendix D.2.

to yield lowest steps (marked by gold star ⋆) are located in the middle of the search ranges rather than sitting at the search boundaries, across different batch sizes and sparsity levels; *e.g.*, the best learning rate is located around 10^{-2} while the best momentum is found around a bit larger than 0.9. Also, the best learning rate increases with batch size, which aligns well with the classic result in learning theory that large batch training allows using bigger learning rates.

Interestingly, there are two distinguished regions (*i.e.*, complete (•) and incomplete (▲)) being separated by a seemingly linear boundary as per the relationship between learning rate and momentum. This indicates that the optimization is being done by an interplay between these two metaparameters. More precisely, if one metaparameter is not chosen carefully with respect to the other (*e.g.*, increasing the learning rate for a fixed momentum), the optimizer may be stuck in a region spending time oscillating and eventually results in incomplete runs that did not reach a goal error within the maximum training iterations. Also, we can see that the successful region (filled with blue circles •) becomes larger as with increasing batch size, showing that large batch training reaches a given goal error in less number of iterations than small batch training, and hence, yields more complete runs.

We further present results for the workload {MNIST, Simple-CNN, SGD} in Figure 4. By isolating learning rate from momentum, it facilitates the search result analysis, and also allows us to study the effect of sparsity theoretically. More specifically, notice that for each batch size, the size of the range for successful learning rate $\bar{\eta}$ decreases by introducing sparsity. This supports our claim later in Section 3.2 that sparsity reduces the smoothness of gradients. We explain further in Appendix C.

3 Understanding the effects of data parallelism and sparsity

3.1 Convergence analysis as a tool to understand the general effects of data parallelism

We have empirically demonstrated across various workloads and sparsity levels that there seems to exist a general trend in the relationship between batch size and steps-to-result for training neural networks. While our results align well with previous findings [17, 21], we wish to gain theoretical insights into such global phenomenon and to develop a better account of the effects of data parallelism. To this end, we seek an answer from the convergence theory of stochastic gradient methods.

Let us begin with reviewing the convergence properties of stochastic gradient methods as the choice of numerical algorithms for solving optimization problems. Consider a generic optimization problem where the objective is to minimize some risk with the objective function $f : \mathbb{R}^m \to \mathbb{R}$, a prediction function $h : \mathbb{R}^{d_x} \times \mathbb{R}^m \to \mathbb{R}^{d_y}$, and a loss function $l : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$ which yields the loss $l(h(\boldsymbol{x}; \mathbf{w}), \boldsymbol{y})$ given an input-output pair $(\boldsymbol{x}, \boldsymbol{y})$, where $\mathbf{w} \in \mathbb{R}^m$ is the parameters of the

prediction model h, and d_x and d_y denote the dimensions of input $\boldsymbol{x}$ and output $\boldsymbol{y}$, respectively. A generalized stochastic gradient method to solve this problem can be of the following form:

$$\mathbf{w}_{k+1} \coloneqq \mathbf{w}_k - \eta_k g(\mathbf{w}_k, \boldsymbol{\xi}_k)\,, \tag{1}$$

where η_k is a scalar learning rate, $g(\mathbf{w}_k, \boldsymbol{\xi}_k) \in \mathbb{R}^m$ is a stochastic vector (*e.g.*, gradient) with $\boldsymbol{\xi}_k$ denoting a random variable to realize data samples (either a single sample as in the prototypical stochastic gradient method [16] or a set of samples as in the mini-batch version [1]). Given an initial iterate $\mathbf{w}_1$, it finds a solution by performing the above update iteratively until convergence.

Under assumptions[2] on Lipschitz smoothness and variance of $g(\mathbf{w}_k, \boldsymbol{\xi}_k)$, the convergence rate result states that for such generic problem with nonconvex objective and optimization method with a fixed learning rate $\eta_k = \bar{\eta}$ for all k satisfying $0 < \bar{\eta} \leq \frac{\mu}{LM_G}$, the expected average squared norm of gradients of the objective function is guaranteed to satisfy the following inequality for all $K \in \mathbb{N}$ [2]:

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \|\nabla f(\mathbf{w}_k)\|_2^2\right] \leq \frac{\bar{\eta}LM}{\mu} + \frac{2(f(\mathbf{w}_1) - f_\infty)}{K\mu\bar{\eta}}\,. \tag{2}$$

Here, $f(\mathbf{w}_1)$, f_∞, $\nabla f(\mathbf{w}_k)$ refer to the objective function's value at $\mathbf{w}_1$, lower bound, gradient at $\mathbf{w}_k$, respectively. Also, L is the Lipschitz constant of ∇f, and μ, M, M_G denote scalar bounds in the assumption on the second moment of $g(\mathbf{w}_k, \boldsymbol{\xi}_k)$. Note here that M is linked to batch size B as $M \propto 1/B$. In addition, if $g(\mathbf{w}_k, \boldsymbol{\xi}_k)$ is an unbiased estimate of $\nabla f(\mathbf{w}_k)$, which is the case for $\boldsymbol{\xi}_k$ being i.i.d. samples as in our experiments, then simply $\mu = 1$ [2]. In essence, this result shows that the average squared gradient norm on the left-hand side is bounded above by asymptotically decreasing quantity as per K, indicating a sublinear convergence rate of the method. We note further that the convergence rate for the mini-batch stochastic optimization of nonconvex loss functions is studied previously [6, 20], and yet, here we reconsider it to analyze the effects of data parallelism.

We now reformulate this result, such that it is translated into a form that matches our experiment settings and reveals the relationship between batch size and steps-to-result. We start by recognizing that the quantity on the left-hand side, the expected average squared norm of $\nabla f(\mathbf{w}_k)$ during the first K iterations, indicates the degree of convergence; for example, it gets smaller as training proceeds with increasing K. Thus, this quantity is directly related to a goal error to reach in our experiments, which is set to be fixed across different batch sizes for a given workload. This effectively means that training has stopped, and K will no longer contribute to decrease the bound of the quantity. Also, recall that we select the *optimal* learning rate $\bar{\eta}^\star$, out of extensive metaparameter search, to record the *lowest* number of steps to reach the given goal error, *i.e.*, steps-to-result $K^\star$. Next, notice that the only factors that constitute the inequality in Eq. 2 are the Lipschitz constant L and the variance bound M, and if they are assumed to be tight in the worst case, the inequality becomes tight. Now we are ready to provide the relationship between batch size (B) and steps-to-result ($K^\star$) as follows:

Proposition 3.1. Let $\varepsilon = \mathbb{E}\big[\frac{1}{K^\star}\sum_{k=1}^{K^\star} \|\nabla f(\mathbf{w}_k)\|_2^2\big]$ denote a degree of convergence achieved after the first $K^\star$ iterations and $\bar{\eta}^\star$ denote the optimal learning rate used to yield the lowest number of steps $K^\star$ to reach ε. Then,

$$K^\star \approx \frac{c_1}{B} + c_2\,, \qquad \text{where } c_1 = \frac{\Delta L\beta}{\mu^2\varepsilon^2} \text{ and } c_2 = \frac{\Delta}{\bar{\eta}^\star\mu\varepsilon}\,, \tag{3}$$

where $\Delta = 2(f(\mathbf{w}_1) - f_\infty)$, β is the initial variance bound at batch size $B = 1$ for a given workload.

Proof. This result is obtained by recasting Eq. 2 as outlined above. The proof is in Appendix A. □

This result precisely illustrates the relationship between batch size B and steps-to-result. For example, when B is small, $K^\star \approx \frac{c_1}{B}$, fitting the linear scaling regime (*e.g.*, $B \to 2B$ makes $K^\star \to (1/2)K^\star$), whereas when B is large and asymptotically dominates the right-hand side, $K^\star \approx c_2$, indicating the maximal data parallelism as $K^\star$ remains constant. In general, for moderate batch sizes, scaling $B \to 2^r B$ results in $K^\star \to \frac{1}{2^r}K^\star + (1 - \frac{1}{2^r})c_2$ (rather than $\frac{1}{2^r}K^\star$), indicating diminishing returns. Therefore, this result not only well accounts for the scaling trend observed in the experiments, but also describes it more precisely and generally. We note that the effect of data

[2] (simplified) (i) f is differentiable and satisfies $\|\nabla f(\mathbf{w}) - \nabla f(\bar{\mathbf{w}})\|_2 \leq L\|\mathbf{w} - \bar{\mathbf{w}}\|_2, \forall\{\mathbf{w}, \bar{\mathbf{w}}\} \subset \mathbb{R}^m$, and (ii) there exist scalars $M \geq 0$, $M_G \geq \mu^2 > 0$ such that $\mathbb{E}_{\boldsymbol{\xi}_k}\left[\|g(\mathbf{w}_k, \boldsymbol{\xi}_k)\|_2^2\right] \leq M + M_G\|\nabla f(\mathbf{w}_k)\|_2^2$.

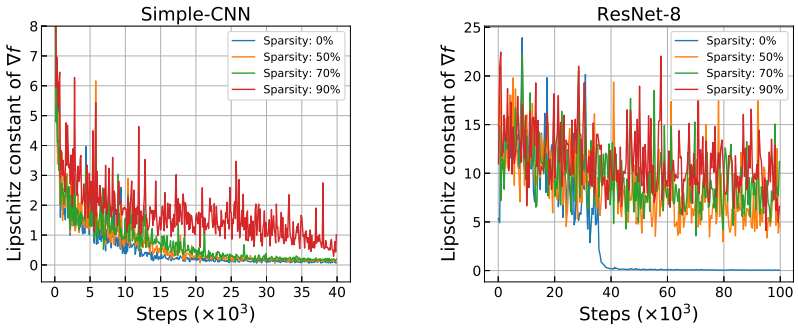


Figure 5: Lipschitz constant of ∇f measured locally over the course of training for networks with different sparsity levels. The more a network is pruned, the higher the Lipschitz constant becomes; *e.g.*, for 0, 50, 70, 90% sparsity levels, the average Lipschitz constants are 0.57, 0.72, 0.81, 1.76 for Simple-CNN and 3.87, 8.74, 9.54, 11.18 for ResNet-8, respectively. This indicates that pruning results in a network whose gradients are less smooth during training. We further provide additional training logs and explain how smoothness is measured in Appendix C.

parallelism, which was previously limited to serve as empirical evidence to support large-batch training, is now theoretically verified. We will further relate our result to sparse networks via smoothness analysis in Section 3.2.

3.2 Attributing Lipschitz smoothness to the difficulty of training sparse neural networks

Another distinct phenomenon observed in our experiments is that the number of steps required to reach the same goal error for sparse networks is consistently higher than that for dense networks regardless of batch size (*i.e.*, a whole data parallelism curve shifts upwards when introducing sparsity). This indicates the general difficulty of training sparse neural networks, and that sparsity degrades the training speed. In this section, we investigate what may cause this difficulty, and find a potential source of the problem by inspecting our theory of the effects of data parallelism to this end.

Let us begin with our result for the effect of data parallelism in Proposition 3.1. Notice that it is the coefficient c_1 ($= \Delta L \beta / \mu^2 \varepsilon^2$) that can shift a whole data parallelism curve vertically, by the same factor across different batch sizes. Taking a closer look, we realize that it is the Lipschitz constant L that can vary quite significantly by introducing sparsity and hence affect c_1; ε and μ are invariable, and Δ and β can change by sparsity in a relatively minor manner (we explain this in detail in Appendix C). Specifically, L refers to the bound on the rate of change in ∇f and is by definition a function of f. Also, sparsity introduced by pruning changes the prediction function h which is linked to f via the loss function l. Therefore, we posit that a sparse neural network obtained by pruning will be *less smooth* (with a higher Lipschitz constant) than the non-pruned dense network. To verify our hypothesis, we empirically measure the Lipschitz constant for networks with different sparsity levels over the course of the entire training process. The results are presented in Figure 5.

As we can see, it turns out that the Lipschitz constant increases as with increasing sparsity level, and further, is consistently higher for sparse networks than for the dense network throughout training. This means that *pruning results in sparse networks whose gradient changes relatively too quickly* compared to the dense network; in other words, the prediction function h becomes *less smooth* after pruning. This is potentially what hinders training progress, and as a result, sparse networks require more time (*i.e.*, steps-to-result) to reach the same goal error. We note that our findings of increased Lipschitz constant for sparse networks are consistent with the literature on over-parameterized networks such as [14], which can be seen as the opposite of sparsity. The more input weights a neuron has, the less likely it is that a single parameter significantly changes the resulting activation pattern, and that wide layers exhibit convexity-like properties in the optimization landscape [4]. This even extends to non-smooth networks with ReLU activations, which are still shown to exhibit pseudo-smoothness in the overparameterized regime [14]. We further show that our theory precisely explains the difficulty of training sparse networks due to decreased smoothness based on a quantitative analysis in Appendix C.

4 Discussion and future work

In this work, we studied the effects of data parallelism and sparsity on neural network training. Despite the potential synergy between sparsity and data parallelism, little has been known about the behavior of sparse networks for this aspect. To this end, we first conducted rigorous experiments to accurately measure the effects, and further developed a theoretical analysis to precisely account for the general characteristics of data parallelism and sparsity in training neural networks. We find that there exists a universal trend in the relationship between batch size and number of training steps to convergence irrespective of sparsity levels, and that sparse neural networks can potentially benefit from large-batch training. While our findings render positive impacts to practitioners and theorists alike, there remain several challenges to address. First, our experiments are bounded by available computing resources; the cost of experiments increases exponentially for more complex workloads. Also, the lack of general convergence guarantees for existing momentum schemes in nonconvex settings hinders a further theoretical analysis. We hypothesize that ultimate understanding of the effects of data parallelism should be accompanied by a study of the generalization capability of optimization methods, however, these are beyond the scope of our current work, and we intend to explore this direction further as future work.

Broader Impact

A fundamental aim of this research paper is linked to the idea of improving the efficiency of deep learning technologies by studying the general behaviors of sparse neural network models in regard to the effects of data scaling in their training. Using large models has been a key to success in the development of highly intelligent systems in recent years, however, it often has limited accessibility, and thus simultaneously poses significant challenges to those who may not have enough computing resources to employ such models. A part of our results indicates that neural networks in a compressed form can utilize given computing resources more effectively during training. Hence, we believe that our study can potentially render a positive impact in the general computing research community without incentivizing a particular group of individuals. Our results can also lead to a development of more efficient hardwares in new ways, which is believed to have a long-term benefit to society. There could always be negative implications of scientific and technological advancements, however, it is our current belief that this work, being theoretical and focusing on understanding neural networks, is distant from having negative impacts or foreseeable potential risks in the near future.

References

[1] Léon Bottou. Stochastic gradient learning in neural networks. *Neuro Nîmes*, 1991.

[2] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

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

[4] Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *ICLR*, 2019.

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

[6] Saeed Ghadimi, Guanghui Lan, and Hongchao Zhang. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, 155 (1-2):267–305, 2016.

[7] 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.

[8] 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.

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

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

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

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

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

[14] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. *NeurIPS*, pages 8157–8166, 2018.

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

[16] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[17] 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*, 2019.

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

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

[20] Weiran Wang and Nathan Srebro. Stochastic nonconvex optimization with large minibatches. *arXiv preprint arXiv:1709.08728*, 2017.

[21] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *NeurIPS*, 2019.

[22] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *ICLR*, 2020.

A The relationship between batch size and steps-to-result

In this section, we provide the full derivation of Proposition 3.1 in Section 3. We start from the convergence rate result in Eq. 2. We first recognize that the expected average squared gradient norm on the left-hand side indicates the degree of convergence. Then, this quantity is directly related to the concept of goal error to reach in our experiments, and hence, reduces to be a small constant ϵ as soon as it is implemented as a pre-defined goal error for a given workload. Thus, it follows that

$$\epsilon \leq \frac{\bar{\eta}LM}{\mu} + \frac{2(f(\mathbf{w}_1) - f_\infty)}{K\mu\bar{\eta}} \,. \tag{4}$$

Notice that by fixing $\epsilon = \mathbb{E}\big[\frac{1}{K}\sum_{k=1}^{K} \|\nabla f(\mathbf{w}_k)\|_2^2\big]$, it effectively means that the training process has stopped, and therefore, K on the right-hand side will no longer contribute to decrease the bound of the quantity for a particular learning rate $\bar{\eta}$ and batch size B.

Also, we select the *optimal*[3] learning rate $\bar{\eta}^\star$, out of extensive metaparameter search, to record the *lowest* number of steps to reach the given goal error, which we denote as steps-to-result $K^\star$. Plugging in these yields the following:

$$\epsilon \leq \frac{\bar{\eta}^\star LM}{\mu} + \frac{2(f(\mathbf{w}_1) - f_\infty)}{K^\star\mu\bar{\eta}} \,. \tag{5}$$

Next, notice that the only factors that constitute the inequality in Eq. 2 come from the assumptions made to derive the convergence rate result, which are on the Lipschitz constant L and the variance bound M, and if they are assumed to be tight in the worst case, the inequality becomes tight. Then, after making algebraic manipulation and taking the first-order Taylor approximation while substituting $M = \beta/B$ since the variance bound M is related to batch size B as $M \propto 1/B$ [2], we obtain the following result:

$$\begin{aligned} K^\star &\approx \frac{\Delta}{\bar{\eta}^\star\mu\epsilon - (\bar{\eta}^\star)^2 LM} \\ &\approx \frac{\Delta}{\bar{\eta}^\star\mu\epsilon} + \frac{\Delta LM}{\mu^2\epsilon^2} \\ &= \frac{\Delta L\beta}{\mu^2\epsilon^2 B} + \frac{\Delta}{\bar{\eta}^\star\mu\epsilon} \\ &= \frac{c_1}{B} + c_2 \,, \qquad \text{where } \; c_1 = \frac{\Delta L\beta}{\mu^2\epsilon^2} \; \text{ and } \; c_2 = \frac{\Delta}{\bar{\eta}^\star\mu\epsilon} \,. \end{aligned} \tag{6}$$

Here, $\Delta = 2(f(\mathbf{w}_1) - f_\infty)$, β is the initial variance bound at batch size $B = 1$ for a given workload. Notice that ϵ, Δ, L, β, μ $\bar{\eta}^\star$ all are constant or become fixed for a given workload. Therefore, this result precisely illustrates the relationship between batch size B and steps-to-result $K^\star$. Please refer to the main paper for the analysis of this result. We also note that the degree of metaparameter search quality is assumed to be the same across different batch sizes, and hence, the result can be reliably used to interpret the relationshp between batch size and steps-to-result.

B Scale of our experiments

For a given workload of {data set, network model, optimization algorithm} and for a study setting of {batch size, sparsity level}, we execute 100 training runs with different metaparameters to measure steps-to-result. At each run, we evaluate the intermediate models at every 16 (for MNIST) or 32 (for CIFAR-10) iterations, on the *entire* validation set, to check if it reached a goal error. This means that, in order to plot the results for the workload of {MNIST, Simple-CNN, SGD} for example, one would need to perform, 14 (batch sizes; 2^1 to 2^{14}) $\times$ 4 (sparsity levels; $0, 50, 70, 90\%$) $\times$ 100 (runs) $\times$ $40,000$ (max training iteration preset) $/$ 16 (evaluation interval) $= 14,000,000$ number of evaluations. Assuming that evaluating the Simple-CNN model on the entire MNIST validation set takes only a *second* on a modern GPU, it will take $14,000,000$ (evaluations) $\times$ 1 (second per evaluation) $/$ 3600 (second per hour) ≈ 3888 hours or 162 days.

[3]Here, *optimal* simply refers to the sense of yielding the *lowest* steps-to-result.

Of course, there are multiple ways to reduce this cost; for instance, we may decide to stop as soon as the run hits the goal error without running until the max training iteration limit. Or, simply reducing any factor listed above that contributes to increasing the experiment cost (*e.g.*, number of batch sizes) can help to reduce the time, however, in exchange for the quality of experiments. We should also point out that this is only for one workload where we assumed that the evaluation takes only a second. The cost can increase quite drastically if the workload becomes more complex and requires more time for evaluation and training (*e.g.*, CIFAR-10). Not to mention, we have tested for multiple workloads besides the above example, in order to confirm the generality of the findings in this work.

C More on Lipschitz smoothness analysis

We measure the local Lipschitz constant of ∇f based on a Hessian-free method as used in [22]. Precisely, the local smoothness (or Lipschitz constant of the gradient) at iteration k is estimated as in the following:

$$\hat{L}(\mathbf{w}_k) = \max_{\gamma \in \{\delta, 2\delta, .., 1\}} \frac{\|\nabla f(\mathbf{w}_k + \gamma \boldsymbol{d}) - \nabla f(\mathbf{w}_k)\|_2}{\|\gamma \boldsymbol{d}\|_2}, \tag{7}$$

where $\boldsymbol{d} = \mathbf{w}_{k+1} - \mathbf{w}_k$ and $\delta \in (0, 1)$ for which we set to be $\delta = 0.1$. The expected gradient ∇f is computed on the entire training set, and we measure $\hat{L}(\mathbf{w}_k)$ at every 100 iterations throughout training. This method searches the maximum bound on the smoothness along the direction between $\nabla f(\mathbf{w}_{k+1})$ and $\nabla f(\mathbf{w}_k)$ based on the intuition that the degree of deviation of the linearly approximated objective function is bounded by the variation of gradient between $\mathbf{w}_{k+1}$ and $\mathbf{w}_k$. Furthermore, while ReLU networks (*e.g.*, Simple-CNN, ResNet-8) can only be piecewise smooth, the smoothness can still be measured for the same reason that we measure gradient (*i.e.*, it only requires differentiability).

In addition to our main results on the effect of data parallelism, we provide in Figure 6 the training logs of the networks used for the Lipschitz smoothness analysis in Section 3.2, in order to show the correlation between the Lipschitz smoothness of a network and its training performance; *i.e.*, sparsity incurs low smoothness of gradients (high L; see Figure 5) and hence the poor training performance.

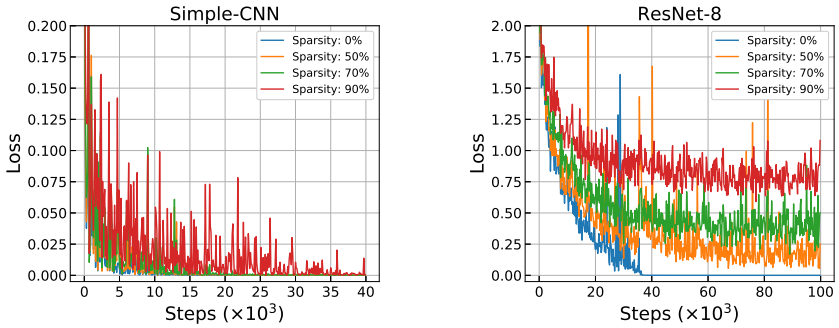


Figure 6: Training logs of the networks Simple-CNN and ResNet-8 used for the smoothness analysis in Section 3.2. The sparse networks that recorded high Lipschitz constants show worse training performance, indicating that low smoothness may be the potential cause of hampering the training of sparse neural networks.

We also empirically measure the changes in Δ and β by introducing sparsity. Recall that these are the other elements in c_1 that can be affected by sparsity along with Lipschitz constant L. When we measure these quantities for Simple-CNN, we obtain the following results: $\Delta_s/\Delta_d \approx 4.68/4.66 \approx 1.00$, and $\beta_s/\beta_d \approx 107.39/197.06 \approx 0.54$; more precisely, Δ does not change much since neither $f(\mathbf{w}_1)$ or $f(\mathbf{w}_\infty)$ changes much, and β_s/β_d can be measured by the ratio of the variances of gradients between sparse and dense networks at $B = 1$. We have already provided L_s, L_d in Figure 5, which makes $L_s/L_d \approx 1.76/0.57 \approx 3.09$. Here, s and d denote sparse (90%) and dense, respectively. Notice that if we combine all the changes in Δ, β, L due to sparsity, and compute $c_{1,s}/c_{1,d}$, it becomes $1.00 \times 0.54 \times 3.09 \approx 1.67$. Importantly, $c_{1,s}/c_{1,d} > 1$ means that c_1 has increased by sparsity, and since the increase in Lipschitz constant L played a major role therein, these results indicate that the general difficulty of training sparse networks is indeed caused by

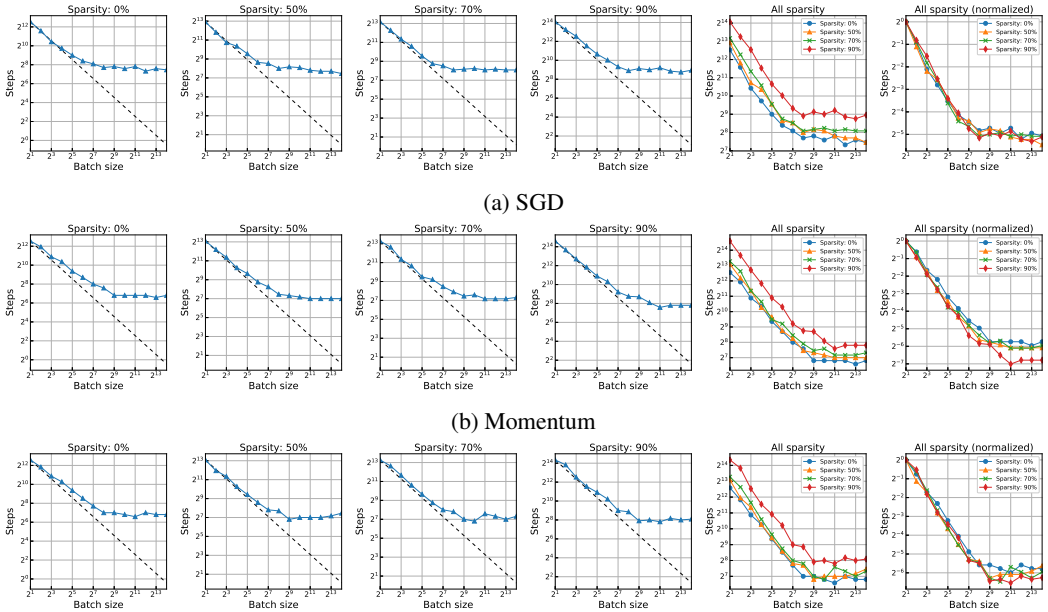
reduced smoothness. We further note that this degree of change fits roughly the range of $k_s^\star/k_d^\star$ as shown in Figure 12.

Evidence of increased Lipschitz constant for sparse networks can be found further in metaparameter search results such as in Figure 4. Notice that for each batch size, the size of the range for successful learning rate $\bar{\eta}$ decreases when switching from 0 to 90% sparsity level. This is potentially because the learning rate bound satisfying the convergence rate theory becomes $0 < \bar{\eta} \leq 1/L$ for a fixed batch size, and increased L due to sparsity shrinks the range of $\bar{\eta}$.

D Additional results

In this section, we provide additional experiemental results that are not included in the main paper. In Section D.1, we supplement more results for the effects of data parallelism and sparsity in Figures 7, 8, 9, 10, 11. In Figure 12 we present the difference in ratio between sparse (90%) and dense networks across different batch sizes for all workloads presented in this work. This result shows how much increase in steps-to-result is induced by introducing sparsity, and therefore, is used to study the general difficulty of training sparse neural networks. In Section D.2, we provide metaparameter search results for a subset of workloads studied in this work.

D.1 Effects of data parallelism and sparsity

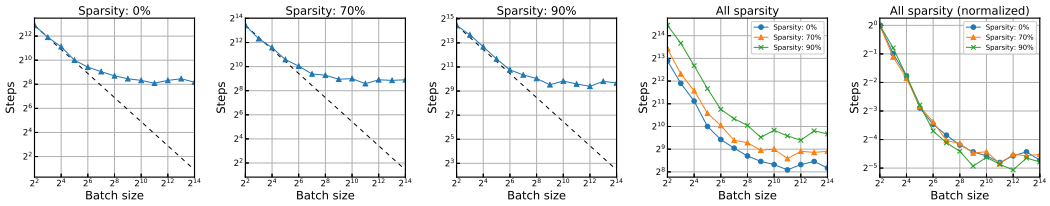


(a) SGD

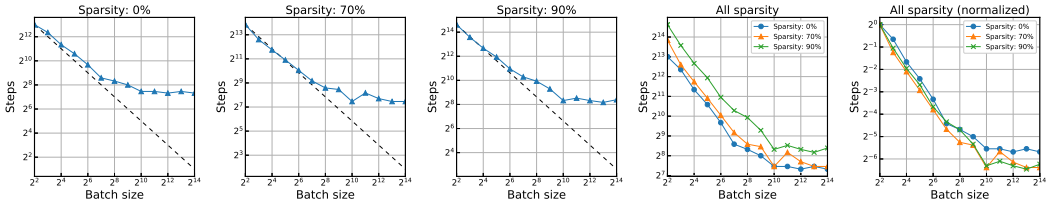
(b) Momentum

(c) Nesterov

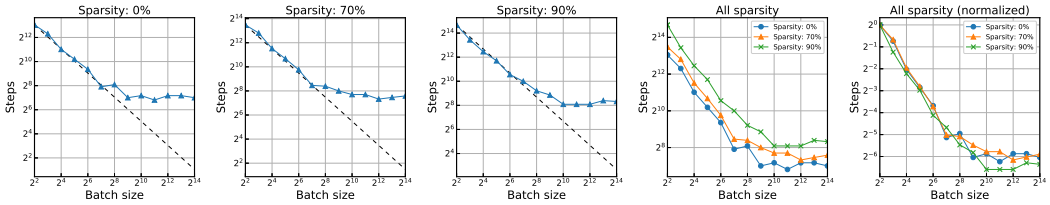
Figure 7: Results for the effects of data parallelism for the workloads of {MNIST, Simple-CNN, SGD/Momentum/Nesterov} with a constant learning rate.



(a) SGD

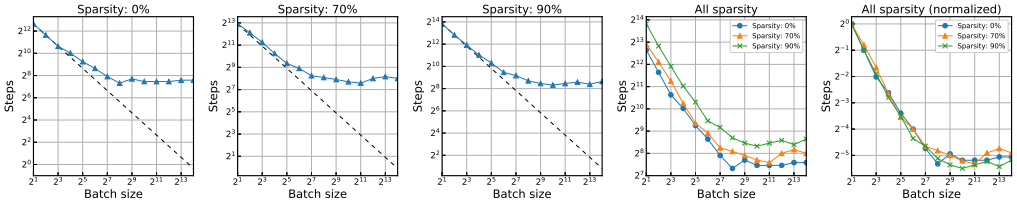


(b) Momentum

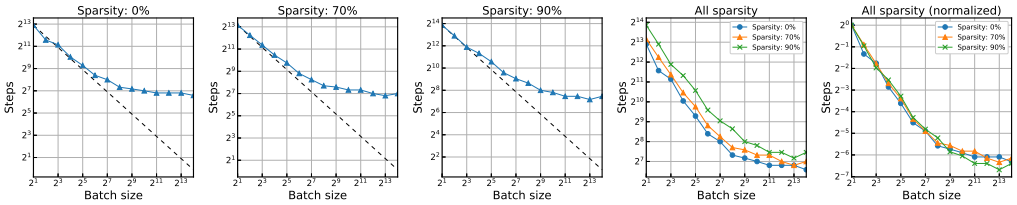


(c) Nesterov

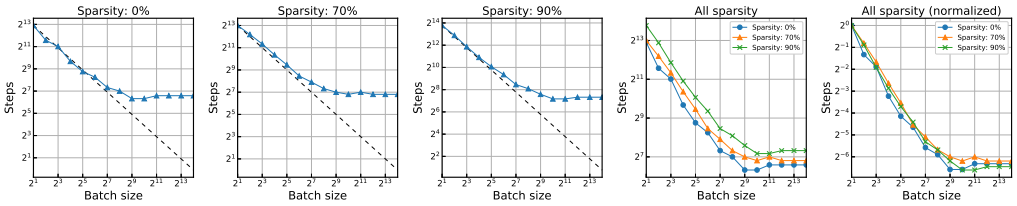
Figure 8: Results for the effects of data parallelism for the workloads of {Fashion-MNIST, Simple-CNN, SGD/Momentum/Nesterov} with a constant learning rate.



(a) SGD

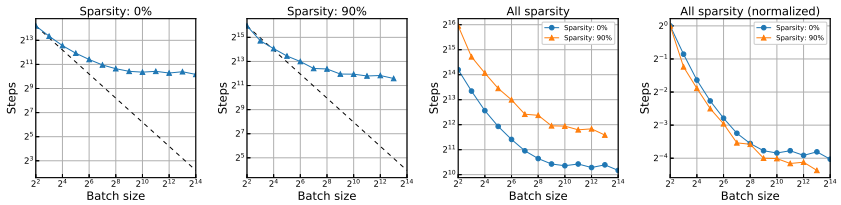


(b) Momentum

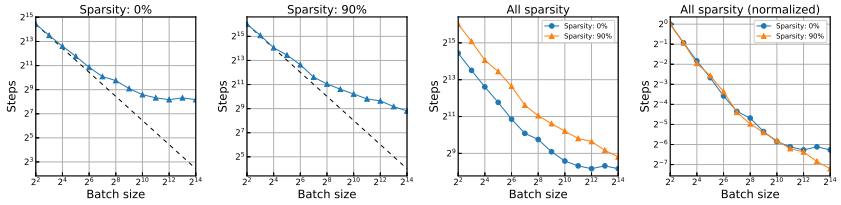


(c) Nesterov

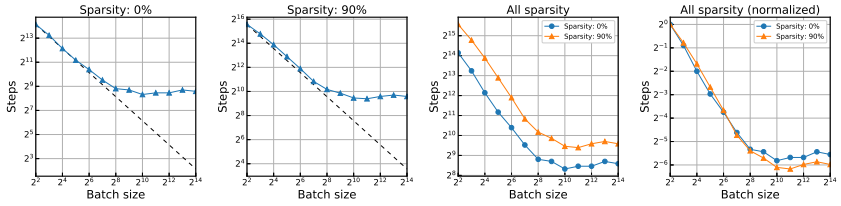
Figure 9: Results for the effects of data parallelism for the workloads of {Fashion-MNIST, Simple-CNN, SGD/Momentum/Nesterov} with a constant learning rate and the goal error of 0.14.



(a) SGD

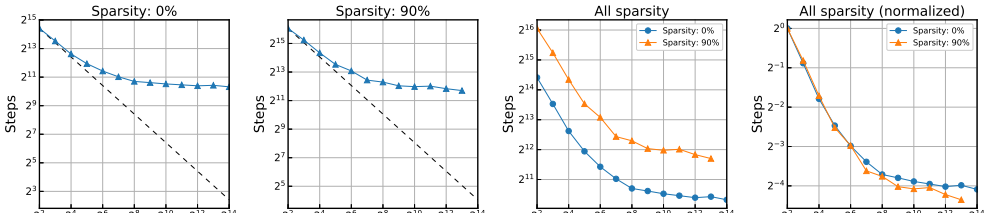


(b) Momentum

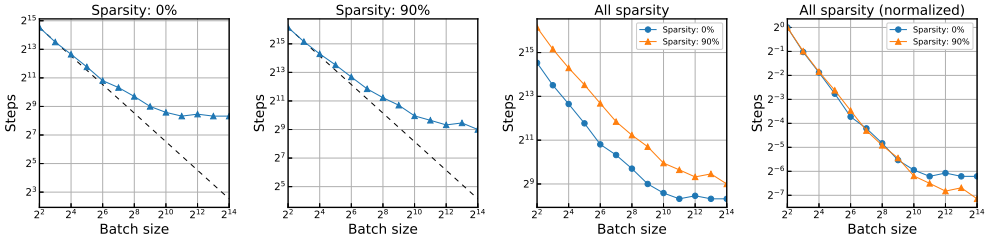


(c) Nesterov

Figure 10: Results for the effects of data parallelism for the workloads of {CIFAR-10, ResNet-8, SGD/Momentum/Nesterov} with a linear learning rate decay.

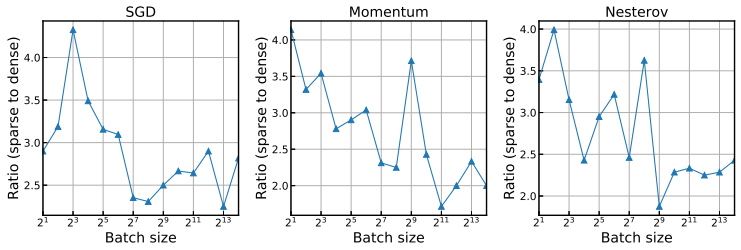


(a) SGD

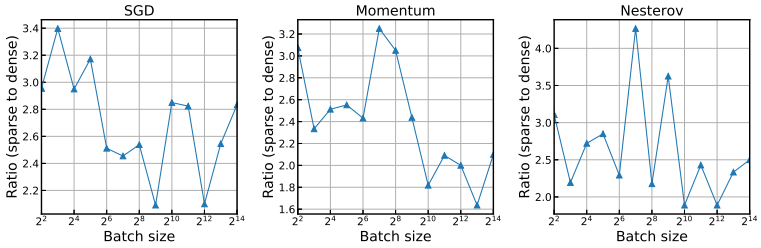


(b) Momentum

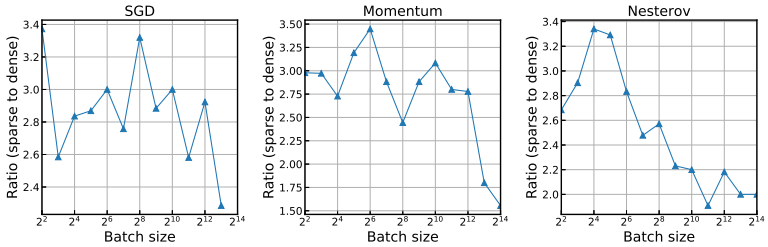
Figure 11: Results for the effects of data parallelism for the workloads of {CIFAR-10, ResNet-8, SGD/Momentum} with a constant learning rate.



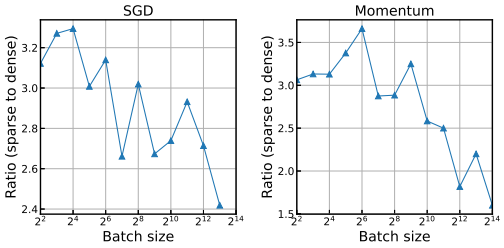
(a) MNIST, Simple-CNN, SGD/Momentum/Nesterov with a constant learning rate



(b) Fashion-MNIST, Simple-CNN, SGD/Momentum/Nesterov with a constant learning rate



(c) CIFAR-10, ResNet-8, SGD/Momentum/Nesterov with a linear learning rate decay



(d) CIFAR-10, ResNet-8, SGD/Momentum with a constant learning rate

Figure 12: Differences in ratio between (90%) sparse network's steps-to-result to dense network's, across different batch sizes for all workloads presented in this work. The difference ranges between (1.5, 4.5) overall. Note that the ratio difference > 1 indicates that it requires more number of training iterations (*i.e.*, steps-to-result) for sparse network compared to dense network. Also, the difference seems to decrease as batch size increases, especially for Momentum based optimizers. This potentially indicates that sparse neural networks can benefit from large batch training, despite the general difficulty therein.

D.2 Metaparameter search results

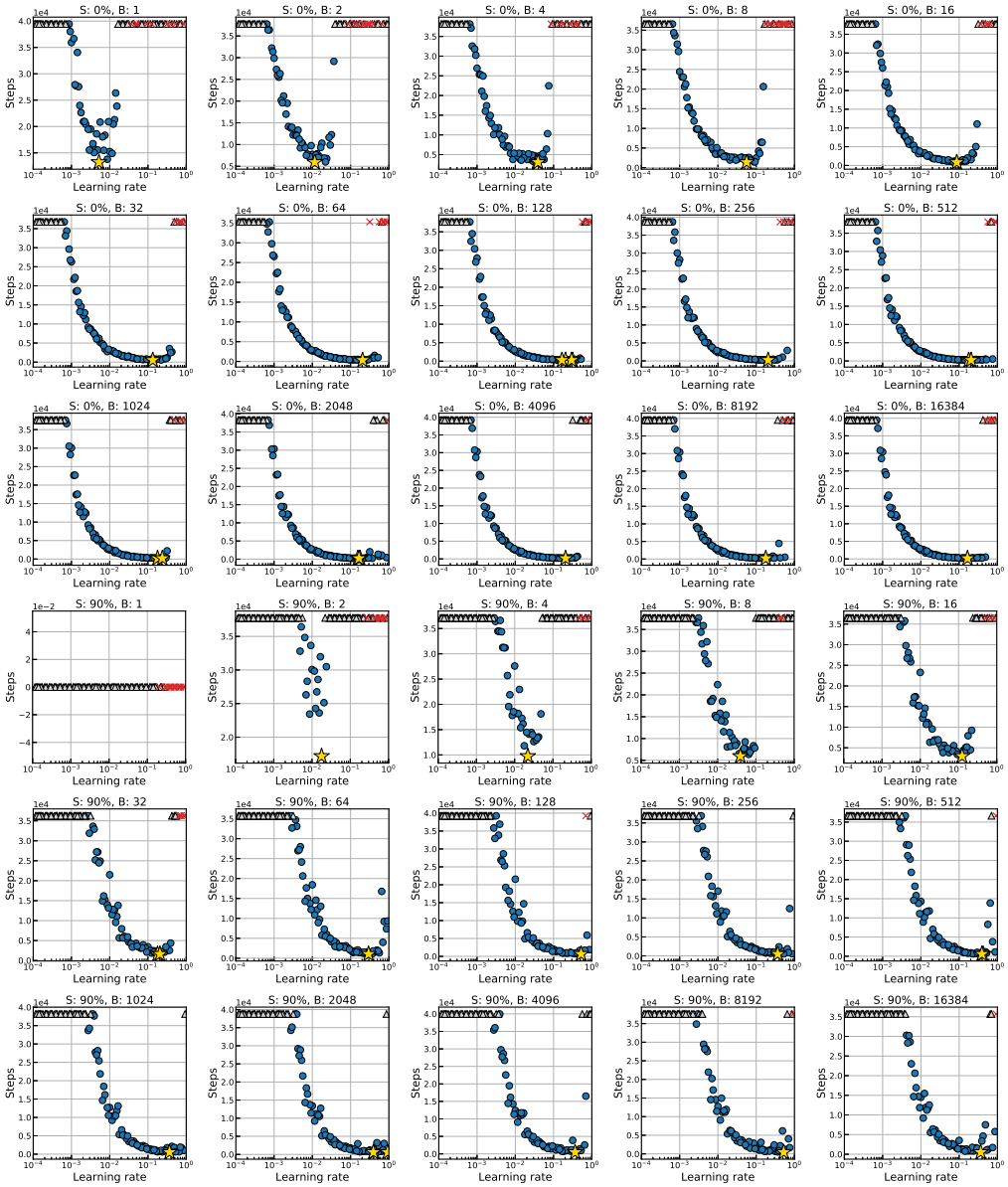


Figure 13: Meataparameter search results for the workloads of {MNIST, Simple-CNN, SGD} with a constant learning rate.

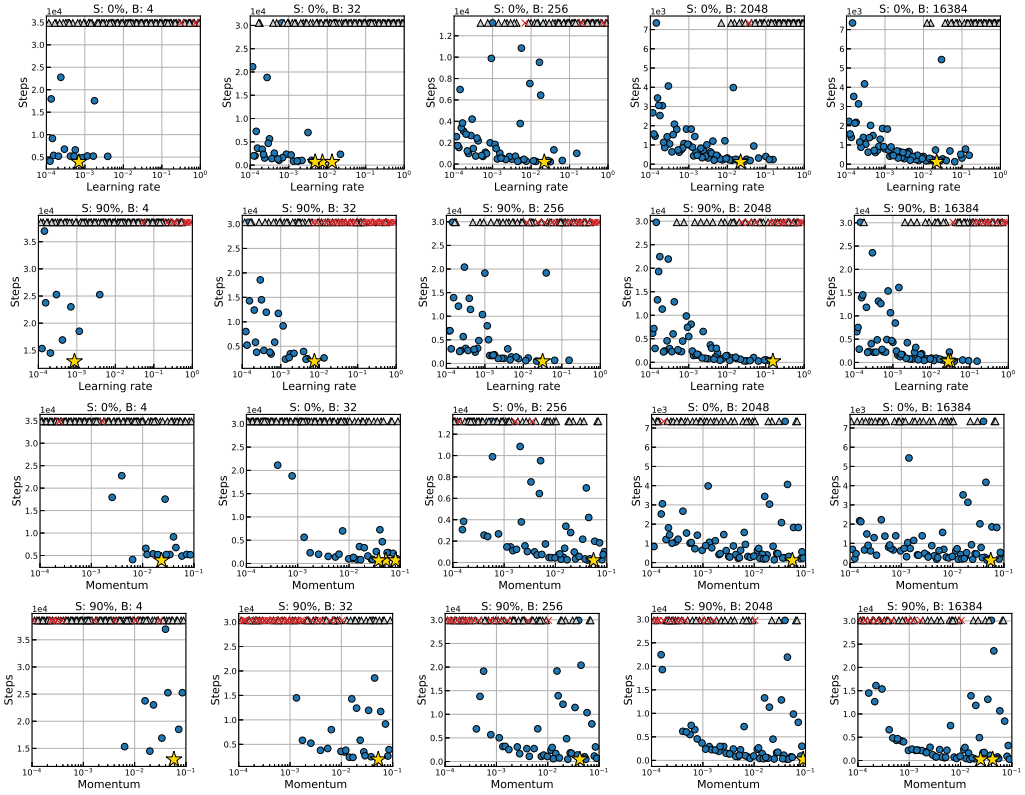


Figure 14: Meataparameter search results for the workloads of {MNIST, Simple-CNN, Momentum} with a constant learning rate.

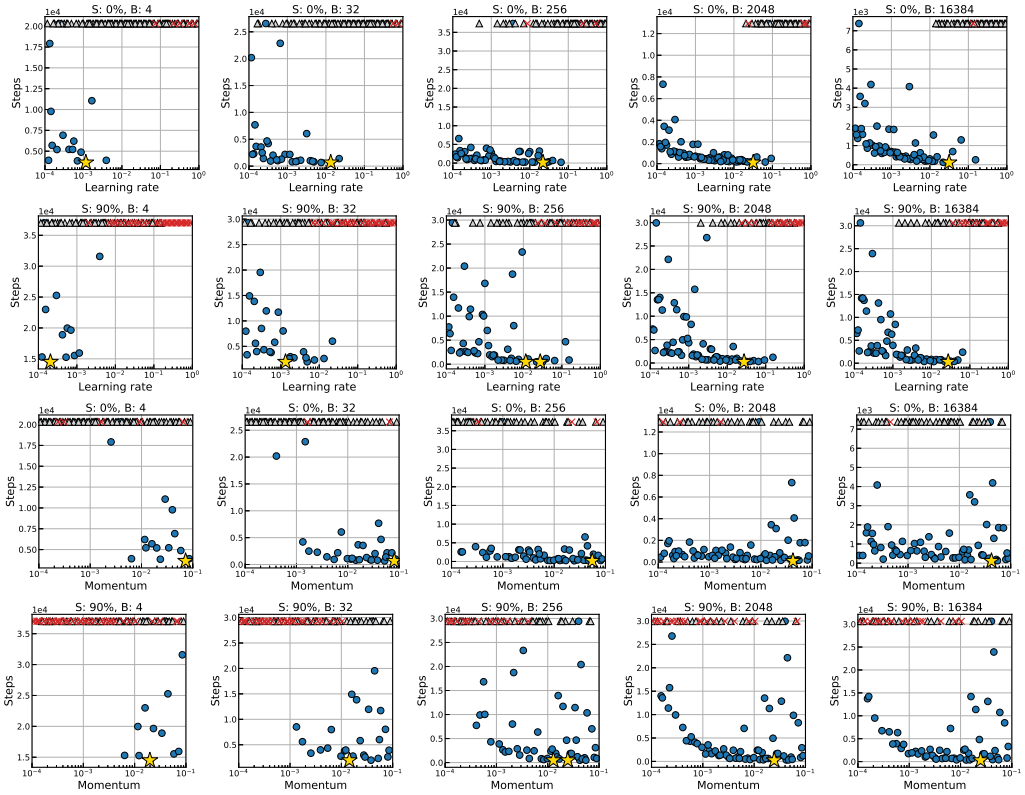


Figure 15: Meataparameter search results for the workloads of {MNIST, Simple-CNN, Nesterov} with a constant learning rate.

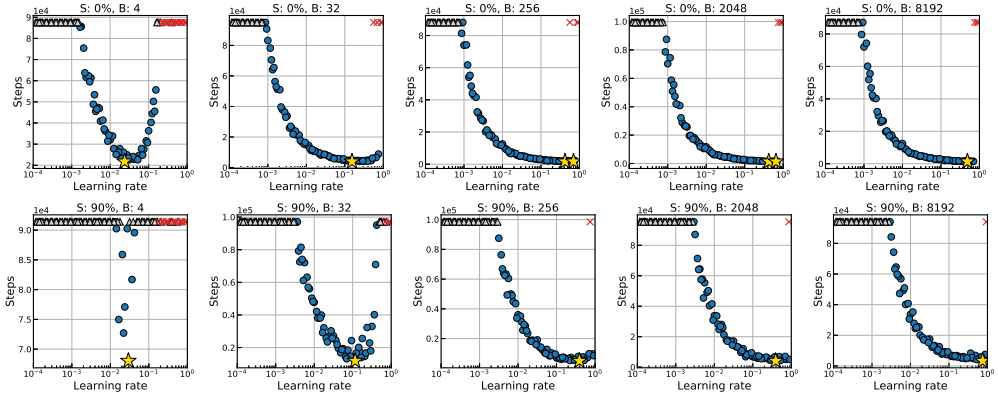


Figure 16: Meataparameter search results for the workloads of {CIFAR-10, ResNet-8, SGD} with a constant learning rate.

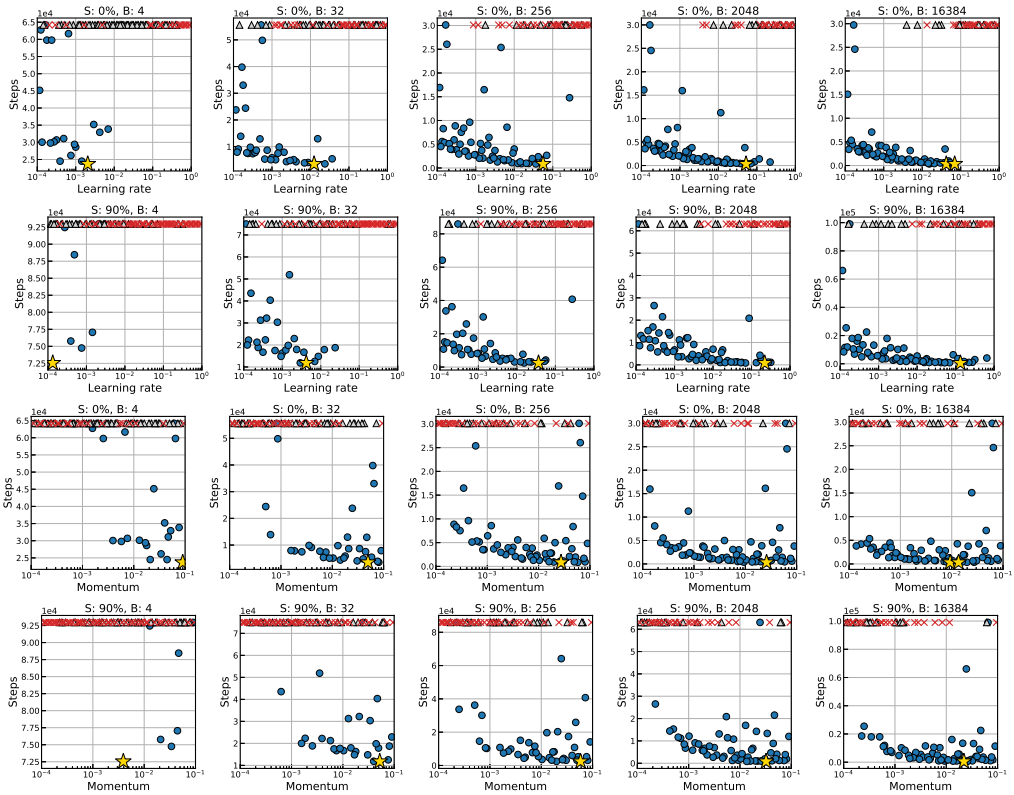


Figure 17: Meataparameter search results for the workloads of {CIFAR-10, ResNet-8, Momentum} with a constant learning rate.

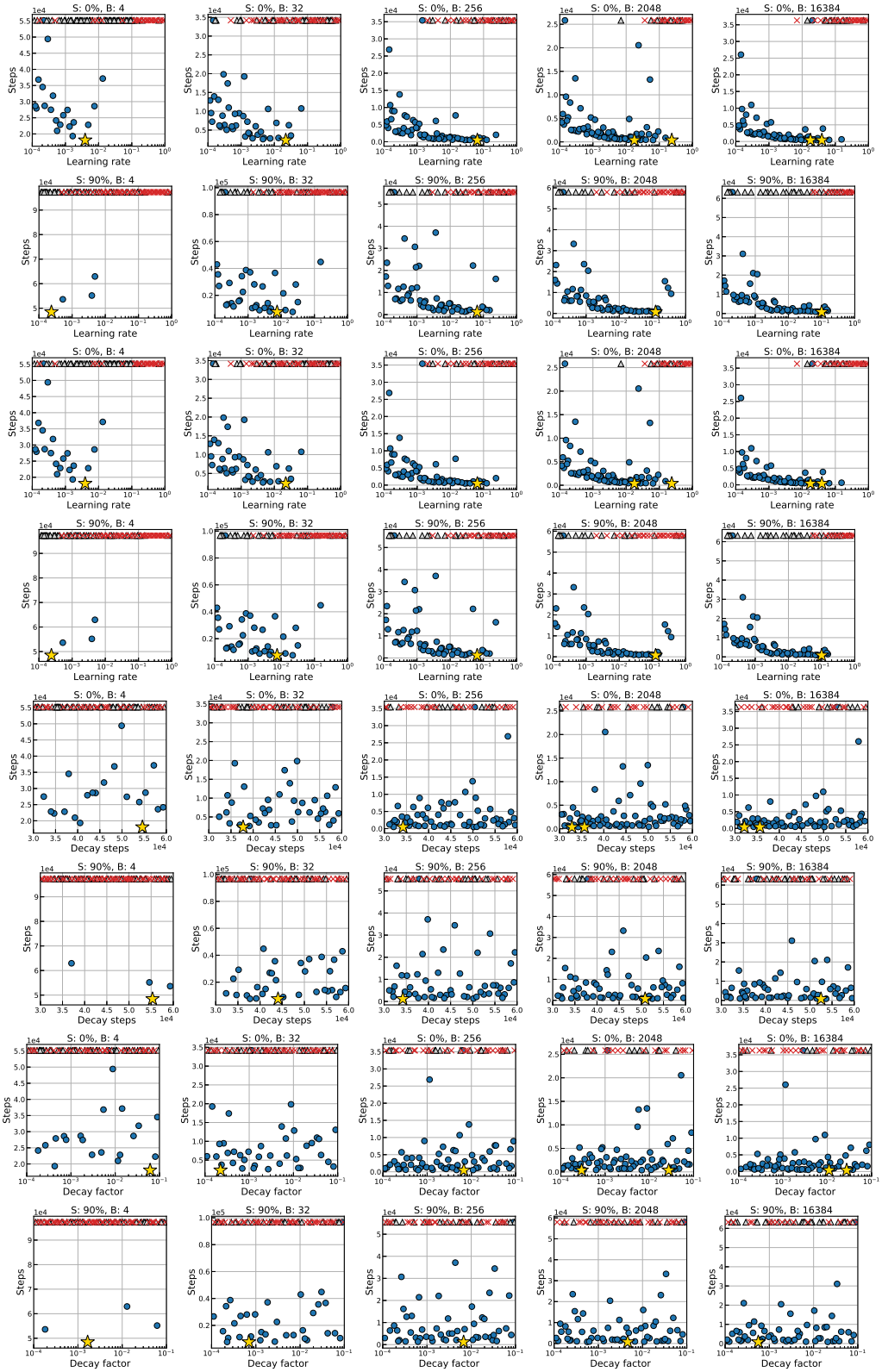


Figure 18: Meataparameter search results for the workloads of {CIFAR-10, ResNet-8, Nesterov} with a linear learning rate decay.