The curse of (non)convexity: The case of an Optimization-Inspired Data Pruning algorithm

Anonymous Author(s)
Affiliation
Address
email

Abstract

Data pruning consists of identifying a subset of the training set that can be used for training instead of the full dataset. This pruned dataset is often chosen to satisfy some desirable properties. In this paper, we leverage some existing theory on importance sampling with Stochastic Gradient Descent (SGD) to derive a new principled data pruning algorithm based on Lipschitz properties of the loss landscape. The goal is to identify a training subset that accelerates training (compared to random data pruning). We call this algorithm `LiPrune`. We illustrate cases where `LiPrune` outperforms existing methods and discuss the limitations and failures of this algorithm in the context of deep learning.

1 Introduction

Data pruning aims at selecting a fraction of the most informative examples in a large training dataset. This is done once, usually at initialization of after a few iterations, and the coreset remains unchanged thereafter. It is an old topic that has amassed a large body of works [Welling, 2009, Chen et al., 2012, Feldman et al., 2011, Huggins et al., 2016, Campbell and Broderick, 2019]. The main objective of data pruning is to reduce the computational complexity and achieve faster training with a lower power consumption. Applications include: 1) neural architecture search (NAS) where models trained with a small fraction of the data serve as a proxy to quickly estimate the performance of a given choice of hyperparameters [Coleman et al., 2019]. 2) Continual/incremental learning: In the context of online learning, in order to avoid the forgetting problem, one keeps track of the most representative examples of past observations [Aljundi et al., 2019]. A variety of approaches can be used in order to select the relevant examples that will constitute the coreset. Examples include Error based approaches where the goal is to find the most 'difficult' examples defined as the ones that contribute the most to the error: keeping the most foregettable examples (that change the most often from being well classified to being misclassified during the course of the training, Toneva et al. [2018]), or the examples with highest expected gradient norm (GraND, EL2N scores Paul et al. [2021]). Another approach is the decision boundary based algorithms which find the examples near the decision boundary, the points for which the prediction has the highest variation (with respect to the input space, Ducoffe and Precioso [2018], Margatina et al. [2021]). In this work, we focus on a different question: *Can we prune the dataset in a way that accelerates training, as compared to e.g. random data pruning?* We answer this question by introducing a new data pruning algorithm, called `LiPrune` , based on the Lipschitz coefficients of the loss function evaluated on single datapoints. `LiPrune` is designed to accelerate the convergence of SGD in the context of convex optimization. We document cases where `LiPrune` succeeds at accelerating the training (convex case) and explain why this algorithm fails in the context of deep learning (non-convex case). A comprehensive discussion is provided at the end of the paper.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

2 SGD, Importance Sampling, and Stochastic data pruning

For some $m \geq 1$, consider an optimization problem of the form

$$\min_{w \in \mathbb{R}^m} F(w) \equiv \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{1}$$

where $(f_i)_{1 \leq i \leq n}$ is a sequence of functions from $\mathbb{R}^m$ to $\mathbb{R}$. Given a dataset $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, N\}$, we are particularly interested in two choices of the functions f_i:

- f_i as point loss: in this case $n = N$ and $f_i(w) = \ell(y_{out}(x_i), y_i)$, where y_{out} is a neural network and ℓ is some loss function, e.g. squared loss for regression, cross-entropy loss for classification.
- f_i as batch loss: in this case $N = n \times B$ where $B \in \mathbb{N}$ is the batch size and $f_i(w) = \frac{1}{B} \sum_{i=1}^{B} \ell(y_{out}(x_{\gamma(i)}), y_{\gamma(i)})$, and γ is a permutation of the set $\{1, \ldots, N\}$. This is the standard setting for neural networks training with SGD; batches are fixed at initialization.

When the number of functions f_i's is large, computing the full gradient $\nabla_w F$ might be expensive. Stochastic Gradient Descent (SGD) uses a randomly sampled index i from the set $\{1, \ldots, n\}$ to estimate the full gradient with the *noisy* version $\nabla_w f_i$[1]. At training iteration $t+1$, SGD updates the weights according to the following rule

$$w_{t+1} = w_t - \eta \nabla_w f_{i_t}(w_t),$$

where i_t is a randomly sampled index in $\{1, \ldots, n\}$, and η is the learning rate. The standard sampling distribution of i is the uniform distribution. Let us assume that there exists a global minimum $w^* = \text{argmin}_w F(w)$. We have the following result on the convergence of SGD.

Theorem 1 (Thm 2.1 and Cor 2.2 in Needell et al. [2014]) *Assume that each f_i is convex and that $\nabla_w f_i$ has Lipschitz constant L_i such that there exists a constant L_{sup} with $L_i \leq L_{sup}$ almost surely. Let $F(w) = \mathbb{E}_i[f_i(w)] = \frac{1}{n} \sum_{i=1}^{n} f_i(w)$ be $\mu-$strongly convex and $\sigma^2 = \mathbb{E}_i[\|\nabla_w f_i(w^*)\|_2^2]$. Given $\epsilon > 0$, and learning rate $\eta = \frac{\mu\epsilon}{2\mu\epsilon L_{sup} + 2\sigma^2}$, we have $\mathbb{E}\|w_t - w_*\|^2 \leq \epsilon$ after $t = 2\log(2\epsilon_0/\epsilon)\left(\frac{L_{sup}}{\mu} + \frac{\sigma^2}{\epsilon\mu^2}\right)$, where $\epsilon_0 = \mathbb{E}\|w_0 - w_*\|^2$.*

Now assume that we sample the index i according to some probability vector $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ such that $\mathbf{p}^T \mathbf{1} = \sum_{i=1}^{n} p_i = 1$ and $p_i \in (0, 1)$. The new SGD update rule is given by

$$w_{t+1} = w_t - \eta \nabla_w g_{i_t}(w_t),$$

where $g_i(w) = \frac{1}{n\, p_i} f_i(w)$. The normalization by $n\, p_i$ is necessary so that we have an unbiased estimate of the gradient, i.e. $\mathbb{E}[\nabla_w g_i(w)] = \nabla F(w)$. This is equivalent to the minimization problem Eq. (1) when f_i is replaced by g_i. Assuming that $\nabla_w f_i$ has Lipschitz constant L_i, the function g_i has Lipschitz constant $\frac{L_i}{np_i}$. Therefore, the new L_{sup} (which depends on $\mathbf{p}$) is given by $L_{sup}(\mathbf{p}) = \sup_i \frac{L_i}{np_i}$. From Theorem 1, the convergence rate is given by $1 - 2\eta\mu(1 - \eta L_{sup}(\mathbf{p}))$, and a smaller $L_{sup}(\mathbf{p})$ results in faster convergence. Hence, a natural choice of the probability vector $\mathbf{p}$ would be the one that minimizes $L_{sup}(\mathbf{p})$, i.e. $\mathbf{p}^* = \text{argmin}_{\mathbf{p}}\, L_{sup}(\mathbf{p})$ under the condition that $\mathbf{p}^T \mathbf{1} = 1$. This has a closed-form solution given by $p_i^* = \frac{L_i}{\bar{L}}$ where $\bar{L} = \frac{1}{n} \sum_{i=1}^{n} L_i$ (Needell et al. [2016]). Hence, choosing probabilities $p_i \propto L_i$ is optimal for *convergence speed* with importance sampling. How do we leverage this result to perform data pruning?

Interpretation. The Lipschitz constant L_i encodes some information about the second order geometry of the function f_i. Assuming that f_i has a second derivative, and that $w \in C \subset \mathbb{R}^m$ where C is a compact set, L_i represents the maximum norm eigenvalue of the Hessian matrix of f_i over the compact set C. Thus, L_i captures the *maximum* curvature of the function f_i over the set C. Hence,

[1] This SGD version is slightly more general than the standard setting where the functions f_i are assumed to be equal to the value of the loss function evaluated at a single datapoint (x_i, y_i).

sampling i according to $p_i \propto L_i$ entails giving more importance to indices i for which f_i has large curvatures. We can think of these indices as those corresponding to *hard* functions f_i. For instance, in the context of ordinary SGD with batch size 1 where $f_i(w) = \ell(y_{out}(x_i), y_i)$ for $(x_i, y_i) \in \mathcal{D}$, the optimal sampling scheme gives more importance to hard examples in the training procedure. For mini-batch SGD, this corresponds to giving more importance to hard batches.

Stochastic pruning. Now that we understand importance sampling in SGD, how do we incorporate data pruning in this framework? Intuitively, given some compression ratio $r \in (0, 1)$[2], we can define a stochastic variant of data pruning as performing the following SGD updates

$$w_{t+1} = w_t - \eta \zeta_t \nabla_w g_{i_t}(w_t), \tag{2}$$

where $\zeta_t \sim \mathcal{B}(r)$ is a Bernoulli variable with probability r, i.e. $\mathbb{P}(\zeta_t = 1) = 1 - \mathbb{P}(\zeta_t = 0) = r$, i_t is sampled according to some probability vector $\mathbf{p}$ and $g_i(w) \overset{def}{=} \frac{1}{rnp_i} f_i(w)$. With this update rule, when we train for n iterations, only a fraction r of the data will be sampled on average. This can be seen a stochastic data pruning algorithm. For this algorithm, we have the following convergence result, which is similar in flavour to that of Theorem 1.

Theorem 2 (SGD Convergence with Stochastic Data Pruning) *Assume that each f_i is convex and that $\nabla_w f_i$ has Lipschitz constant L_i such that there exists a constant L_{sup} with $L_i \leq L_{sup}$. Let $F(w) = \mathbb{E}_i[f_i(w)]$ be $\mu-$strongly convex and $\sigma^2 = \mathbb{E}_i[\|\nabla_w f_i(w^*)\|_2^2]$. Consider the update rule given by Eq. (2), where $g_i(w) = \frac{1}{rnp_i} f_i(w)$, $r \in (0, 1)$ is the compression ratio, and i_t is sampled according to some probability vector $\mathbf{p}$. Suppose $\eta < r/L_{sup}(\mathbf{p})$ where $L_{sup}(\mathbf{p}) \overset{def}{=} \sup_i \frac{L_i}{np_i}$, then with probability at least $1 - e^{-\frac{r^2 t}{2}}$ over the distribution of $\zeta = (\zeta_1, \ldots, \zeta_t)$, we have that*

$$\mathbb{E}\|w_t - w^*\|_2^2 \leq \left(1 - 2\mu \frac{\eta}{r}\left(1 - \frac{\eta}{r} L_{sup}(\mathbf{p})\right)\right)^{r\,t/2} \|w_0 - w^*\|_2^2 + \frac{\eta \sigma^2(\mathbf{p})}{\mu r \left(1 - \frac{\eta}{r} L_{sup}(\mathbf{p})\right)},$$

where $\sigma^2(\mathbf{p}) = \frac{1}{n^2} \sum_{i=1}^n \frac{1}{p_i} \|\nabla_w f_i(w_*)\|_2^2$.

As a result, given some error threshold $\epsilon > 0$, by choosing $\eta = \frac{\mu r \epsilon}{2\sigma^2(\mathbf{p}) + 2\mu L_{sup}(\mathbf{p})\epsilon}$, with probability at least $1 - e^{-\frac{r^2 t}{2}}$ we have that $\mathbb{E}\|w_t - w_*\|^2 \leq \epsilon$ for

$$t = 4 \log\left(\frac{2\epsilon_0}{\epsilon}\right) \frac{1}{r} \left(\frac{L_{sup}(\mathbf{p})}{\mu} + \frac{\sigma^2(\mathbf{p})}{\mu^2 \epsilon}\right).$$

The first part of the proof of Theorem 2 is similar to that of Theorem 2.1 in Needell et al. [2016] and uses the same co-coercivity property. The second part is an application of Hoeffding's inequality to control the deviation of $\sum_{k \leq t} \zeta_k$ from its mean.

An important consequence of Theorem 2 is that the convergence rate of the stochastic pruning algorithm Eq. (2) is controlled by the term $\lambda = 1 - 2\mu \frac{\eta}{r}\left(1 - \frac{\eta}{r} L_{sup}(\mathbf{p})\right)$; the smaller the quantity λ, the faster the convergence. In the case of $r = 1$, Needell et al. [2016] characterizes the choice of the probability vector $\mathbf{p}$ that minimizes the growth factor λ, which is given by $p_i \propto L_i$. It is straightforward to see that this choice of $\mathbf{p}$ is optimal for any choice of compression ratio r.

<u>Conclusion</u>: *To maximize the factor λ, the probability vector $\mathbf{p}$ should be chosen as $p_i = \frac{L_i}{\sum_j L_j}$.*

Theorem 2 shows that sampling according to L_i's results in faster convergence of the stochastic data pruning algorithm given by Eq. (2). However, our goal in data pruning is to reduce the size of the dataset (deterministic pruning) and the stochastic pruning algorithm (Eq. (2)) does not fit in this category (each index i has a non-zero sampling probability). Can we derive a detemrministic version of algorithm Eq. (2) with the optimal choice of $\mathbf{p}$? As straightforward deterministic variant is obtained by using the Lipschitz coefficients L_i's as pruning scores. This <u>consists of keeping the samples i w</u>ith the largest L_i's. We define this algorithm in the next section.

[2]The compression ratio is the fraction of datapoints kept after pruning.

3 Deterministic pruning

The deterministic alternative of stochastic data pruning (Eq. (2)) is to rank the functions f_i based on the Lipschitz constants L_i since it represents *how hard* the function f_i is to train. We call this algorithm `LiPrune` , and we describe it in Algorithm 1. `LiPrune` requires access to the Lipschitz coefficients $L_i's$, which are generally unknown. However, approximations of L_i's can be obtained in different ways:

Algorithm 1 `LiPrune`

Require: Functions $(f_i)_{1\leq i\leq n}$ with Lipschitz constants L_i's, compression ratio $r \in (0,1)$.
$s_i \leftarrow L_i$ (scores)
$\hat{s}_i \leftarrow \text{sorted}(s)_i$ (i^{th} largest score)
$T_r \leftarrow \hat{s}_{r\times n}$ (threshold)
$\mathcal{I} = \emptyset$ (to be populated with indices)
for $i = 1, \dots, n$ **do**
  if $s_i \geq T_r$ **then**
    $\mathcal{I} = \mathcal{I} \cup \{i\}$
  end if
end for
return $\mathcal{I}$

- **Using the Gradient.** Paul et al. [2021] introduced the GraNd algorithm which uses the gradient norm to rank samples. Let $\mathcal{D} = \{(x_i, y_i), i = 1, \dots, n\}$ be a dataset and $f_i(w) = \ell(y_{out}(x_i, w), y_i)$, where ℓ is some loss function and $y_{out}(x_i, w)$ is the network output, and w are the weights of the network. GraNd then computes approximations of the gradient norm given by $\chi_i = \mathbb{E}_w[\|\nabla_w f_i\|]$ where the expectation is taken over random initializations, and uses the scores $(\chi_i)_{1\leq i\leq n}$ as a measure of sample importance. This gradient computation is usually performed at a fixed iteration t, e.g. pruning at initialization where $t = 0$, or during training where $t > 0$. Recall that `LiPrune` uses the Lipschitz constant L_i's as sample importance scores. This Lipschitz constant satisfies $|\nabla_w f_i(w) - \nabla_w f_i(w_*)| \leq L_i\|w - w_*\|$ where $w_* = \text{argmin}_w F(w)$. Hence, given some weight w, we can use the fraction $|\nabla_w f_i(w) - \nabla_w f_i(w_*)|/\|w - w_*\|$ as an approximation of L_i. Assuming that $f_i(w_*) \approx 0$ for all i, GraNd can be seen as an approximate `LiPrune` .
- **Using the Hessian.** A better estimate of the Lipschitz constants L_i's is the norm of the Hessian matrix. Indeed, locally, we have that $|\nabla_w f_i(w') - \nabla_w f_i(w)| \leq \|\nabla_2 f_i(w)\|\|w' - w\|$, and the constant $\|\nabla_2 f_i(w)\|$ is tight. We propose to use the Hessian matrix norm to estimate the Lipschitz constants L_i's. However, a major drawback of this methods is the computation cost of the Hessian matrix norm. This cannot be performed for each example (x_i, y_i) in the dataset. Hence, we restrict our analysis to batch pruning, and consider the second scenario where $f_i(w) = \frac{1}{B}\sum_{x_i\in B_i} \ell(y_{out}(x_i), y_i)$. Intuitively, using `LiPrune` in batch pruning should accelerate SGD convergence as compared to random batch pruning. We summarize this in the following hypothesis.

Hypothesis. *By using the Hessian norm to estimate the Lipschitz constants L_i's in batch pruning, we can accelerate the convergence speed with minimal loss in performance.*

In the next section, we empirically verify our hypothesis by pruning and training models until convergence. We observe that while the loss in performance might be indeed minimal, the convergence speed remains relatively unchanged. We conjecture that this is due to the non-convex nature of the loss landscape, and explain why `LiPrune` might not be the best pruning algorithm in this case.

4 Experiments

4.1 Linear model

We consider a linear model where the data is generated using the following rule ($i \in [N]$, $N = 1000$)

$$y_i = w_* x_i + 0.2\,\varepsilon_i, \quad \varepsilon_i \overset{iid}{\sim} \mathcal{N}(0,1), \tag{3}$$

where $w_* \in \mathbb{R}$ is fixed and $x_i \sim \mathcal{U}([-5,5])$ (uniform distribution). See Fig. 4 for an illustration of the dataset. We would like to fit a linear regression model to the data $\mathcal{D} = \{(x_i, y_i), i = 1, \dots, N\}$ by minimizing the mean squared loss. Is this case, the functions f_i are given by $f_i(w) = \frac{1}{2}(w\,x_i - y_i)^2$, and the Lipschitz constants are given by $L_i = |x_i|^2$. Using `LiPrune` in this case implies choosing the samples (x_i, y_i) with the largest $|x_i|^2$. To understand why this would accelerate convergence, let us look at what happens with SGD. In this case

the update rule is given by $w_{k+1} = w_k - \eta(w_k x_{i_k} - y_{i_k})x_{i_k} = (1 - \eta x_{i_k}^2)w_k + \eta x_{i_k} y_{i_k}$, and hence the convergence speed is controlled by the *contraction* factor $(1 - \eta x_{i_k}^2)$. Choosing x_{i_k} such that $x_{i_k}^2$ is large results in faster convergence (on average). Fig. 1 shows the empirical results of data pruning using `LiPrune` vs. Random pruning for different compression ratios r. Notice that the initial loss value is large with `LiPrune` since we fix the initial point of SGD to $w_0 = 0$. The results confirm our theoretical predictions that `LiPrune` accelerates convergence. More experiments with different hyper-parameters are provided in Appendix B.

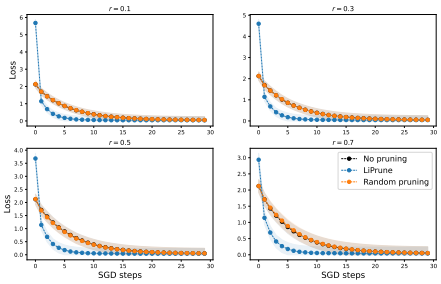
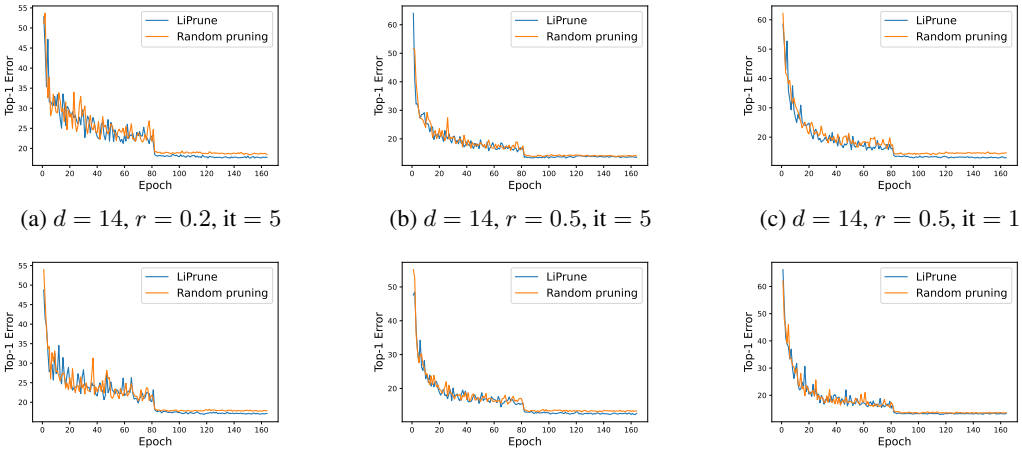


Figure 1: Data pruning using `LiPrune` Vs. Random pruning in the case of the linear model Eq. (3). Test loss (with 400 test samples) with 30 iterations of SGD with batch size=10, learning rate=0.01, $w_0 = 0$, and compression ratios $r \in \{0.1, 0.3, 0.5, 0.7\}$. 95% confidence intervals are shown (based on 100 runs). The curve corresponding to 'No pruning' does not show clearly since it is almost the same as with random pruning.

4.2 Non-convex case: Neural Networks

To evaluate `LiPrune` on deep learning applications, we consider a ResNet architecture and the CIFAR10 dataset. We use batch pruning (based on the estimation of the Hessian norm) at different training iterations (pruning at initialization or later in training). We train the model with SGD with batch size 128 and varying learning rates for 164 epochs. In Fig. 2, we report the test error corresponding to using `LiPrune` with different network depths, compression ratios, and pruning iterations. The learning rate in this case is 0.01 which is divided by 10 at epoch 80 and 120. Experiments with different learning rates are provided in Appendix B.5. From these empirical results, we conclude that there is no (training acceleration) advantage of using `LiPrune` (in the current form of batch pruning) instead of random pruning.



(a) $d = 14$, $r = 0.2$, it $= 5$ (b) $d = 14$, $r = 0.5$, it $= 5$ (c) $d = 14$, $r = 0.5$, it $= 1$

(d) $d = 20$, $r = 0.2$, it $= 5$ (e) $d = 20$, $r = 0.5$, it $= 5$ (f) $d = 20$, $r = 0.5$, it $= 1$

Figure 2: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.2, 0.5\}$, and pruning iteration it $\in \{1, 5\}$ (it $= 1$ corresponds to pruning at initialization, and it $= 5$ corresponds to pruning at the beginning of the fifth epoch).

5 Discussion

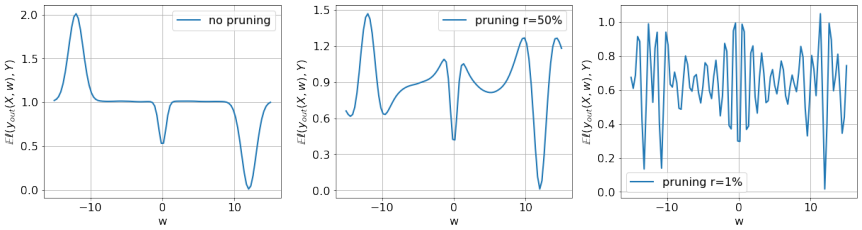
The gains of the data pruning methods are usually less remarkable in deep learning. In particular, they lead to limited improvement compared to naively prioritizing the samples uniformly at random. For example, Guo et al. [2022] conduct extensive experiments with 12 of the most popular data pruning methods on Cifar10 and imagenet datasets with popular architectures (ResNet-18, VGG-16,

Inception-v3) for variaous pruning levels, keeping from 0.1% to 100% of the data. One of the main conclusions is that random pruning is a strong baseline: no method can outperform it systematically across the different levels of pruning; Random pruning achieves the best test accuracy when pruning 90% or 70% of ImageNet data, for example. As seen in the experiment section, our method suffers from the same caveat. We believe that two different and potentially concurrent effects can explain these behaviors: i) modification of the asymptotic loss and ii) modification of the learning dynamic

Modification of the asymptotic loss. In essence, any pruning algorithm aims to find a better subset than random pruning; therefore, it transforms the data distribution, i.e. the distribution of (X, Y). This typically results in a modification of the loss landscape. We illustrate this point using the example developed in Appendix B.4. The data generating process is given by

$$Y = \sin(w_* X) + 0.1\,\varepsilon, \quad \varepsilon \overset{iid}{\sim} \mathcal{N}(0,1), \quad X \overset{iid}{\sim} \mathcal{N}(0,1), \tag{4}$$

where $w_* = 12$, and $y_{out}(x; w) = \sin(wx)$. In Figure 3, we compare the loss landscape that is minimized without pruning (left figure), when keeping only 50%, and 1% of the data (middle and right figures) with LiPrune. For $r = 0.5$, we can still 'easily' identify the minimizer w_*. With r=0.01, the task is more challenging. Though not always mentioned, we believe this behavior is common to many popular pruning algorithms, if not most. For example, the authors of Kawaguchi and Lu [2020], are able to derive a bound on the loss transformation induced by their pruning algorithm in the slightly different setting of dynamic pruning. The authors prove that their algorithm asymptotically returns a global minimum for convex losses. However, there are no such results for general non-convex losses.



(a) No pruning (b) Pruning with $r = 0.5$ (c) Pruning with $r = 0.01$

Figure 3: Asymptotic loss landscape modification due to pruning.

Modification of the learning dynamic. In the convex setting, it is sufficient to reduce the stochastic gradient variance to converge faster, which is the base ingredient of accelerated SGD with importance sampling. In the highly multimodal landscapes of neural network problems, the noise of the gradient plays a pivotal role in the training dynamic and the final performance of the model. Several lines of work argue, for instance, that stochastic procedures such as dropout or stochastic depth improve the generalization results because they introduce additional noise that acts as an implicit regularizer [Wei et al., 2020, Hayou and Ayed, 2021], smoothing the loss landscape. It has long been believed that the noise in SGD favors flat minima [Heskes and Kappen, 1993]. In Xie et al. [2020], the authors adopt a diffusion theory perspective to study the training dynamics and prove, under certain simplifying assumptions, that the noise in SGD exponentially favors flat minima. A different line of work [Ma et al., 2018, Zhang et al., 2019] observes that SGD has two regimes, the *noise dominated* (small batches/large learning rate), characterized by the gradient noise, and *curvature dominated* (large batches/small learning rate). In Smith et al. [2020], the authors empirically verify the existence of the two regimes and confirm that the small batch size (larger noise) leads to test accuracy improvements when compared to large batch sizes (smaller noise). The noise is particularly determinant in the early phase of the training. We argue that a successful data pruning method should consider the changes it induces in the training dynamics and the noise of the gradient. In Johnson and Guestrin [2018], for example, a key component of the method is to adapt the learning rate by rescaling it by the inverse of the estimated gradient noise reduction.

References

Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128, 2009.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. *arXiv preprint arXiv:1203.3472*, 2012.

Dan Feldman, Matthew Faulkner, and Andreas Krause. Scalable training of mixture models via coresets. *Advances in neural information processing systems*, 24, 2011.

Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. *Advances in Neural Information Processing Systems*, 29, 2016.

Trevor Campbell and Tamara Broderick. Automated scalable bayesian inference via hilbert coresets. *The Journal of Machine Learning Research*, 20(1):551–588, 2019.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*, 2019.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. 2021.

Melanie Ducoffe and Frederic Precioso. Adversarial active learning for deep networks: a margin based approach. *arXiv preprint arXiv:1802.09841*, 2018.

Katerina Margatina, Giorgos Vernikos, Loïc Barrault, and Nikolaos Aletras. Active learning by acquiring contrastive examples. *arXiv preprint arXiv:2109.03764*, 2021.

Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Advances in neural information processing systems*, 27, 2014.

D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Math. Program*, 155:549–573, 2016.

Chengcheng Guo, Bo Zhao, and Yanbing Bai. Deepcore: A comprehensive library for coreset selection in deep learning. *arXiv preprint arXiv:2204.08499*, 2022.

Kenji Kawaguchi and Haihao Lu. Ordered sgd: A new stochastic optimization framework for empirical risk minimization. In *International Conference on Artificial Intelligence and Statistics*, pages 669–679. PMLR, 2020.

Colin Wei, Sham Kakade, and Tengyu Ma. The implicit and explicit regularization effects of dropout. In *International Conference on Machine Learning*, pages 10181–10192. PMLR, 2020.

Soufiane Hayou and Fadhel Ayed. Regularization in resnet with stochastic depth. *Advances in Neural Information Processing Systems*, 34, 2021.

Tom M Heskes and Bert Kappen. On-line learning processes in artificial neural networks. In *North-Holland Mathematical Library*, volume 51, pages 199–233. Elsevier, 1993.

Zeke Xie, Issei Sato, and Masashi Sugiyama. A diffusion theory for deep learning dynamics: Stochastic gradient descent exponentially favors flat minima. *arXiv preprint arXiv:2002.03495*, 2020.

Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3325–3334. PMLR, 2018.

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. *Advances in neural information processing systems*, 32, 2019.

Samuel Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. In *International Conference on Machine Learning*, pages 9058–9067. PMLR, 2020.

Tyler B Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. *Advances in Neural Information Processing Systems*, 31, 2018.

A Proof of Theorem 2

Theorem 2 (SGD Convergence with Stochastic Data Pruning). *Assume that each f_i is convex and that $\nabla_w f_i$ has Lipschitz constant L_i such that there exists a constant L_{sup} with $L_i \leq L_{sup}$ almost surely. Let $F(w) = \mathbb{E}_i[f_i(w)]$ be $\mu-$strongly convex and $\sigma^2 = \mathbb{E}_i[\|\nabla_w f_i(w^*)\|_2^2]$. Consider the following SGD update*

$$w_{t+1} = w_t - \eta\zeta_t \nabla_w g_{i_t}(w_t), \tag{5}$$

where $g_i(w) = \frac{1}{rnp_i} f_i(w)$, $r \in (0,1)$ is the compression ratio, and i is sampled according to some probability vector $\mathbf{p}$. Suppose $\eta < r/L_{sup}(\mathbf{p})$ where $L_{sup}(\mathbf{p}) \overset{def}{=} \sup_i \frac{L_i}{np_i}$, then with probability at least $1 - e^{-\frac{r^2 t}{2}}$ over the distribution of $\zeta = (\zeta_1, \ldots, \zeta_t)$, we have that

$$\mathbb{E}\|w_t - w^*\|_2^2 \leq \left(1 - 2\mu\frac{\eta}{r}\left(1 - \frac{\eta}{r}L_{sup}(\mathbf{p})\right)\right)^{r\,t/2} \|w_0 - w^*\|_2^2 + \frac{\eta\sigma^2(\mathbf{p})}{\mu r\left(1 - \frac{\eta}{r}L_{sup}(\mathbf{p})\right)},$$

where $\sigma^2(\mathbf{p}) = \frac{1}{n^2}\sum_{i=1}^n \frac{1}{p_i}\|\nabla_w f_i(w_*)\|_2^2$.

As a result, given some error threshold $\epsilon > 0$, by choosing $\eta = \frac{\mu r \epsilon}{2\sigma^2(\mathbf{p}) + 2\mu L_{sup}(\mathbf{p})\epsilon}$, with probability at least $1 - e^{-\frac{r^2 t}{2}}$ we have that $\mathbb{E}\|w_t - w_*\|^2 \leq \epsilon$ for

$$t = 4\log\left(\frac{2\epsilon_0}{\epsilon}\right)\frac{1}{r}\left(\frac{L_{sup}(\mathbf{p})}{\mu} + \frac{\sigma^2(\mathbf{p})}{\mu^2\epsilon}\right).$$

The first part of the proof of Theorem 2 is similar to that of Theorem 2.1 in Needell et al. [2016] in the sense that it uses the same co-coercivity property. The second part is an application of Hoeffding's inequality to control the deviation of $\sum_{k\leq t}\zeta_k$ from its mean.

Lemma 1 (Co-coercivity, Lemma A.1 in Needell et al. [2016]) *Let f be a smooth function with Lipschitz constant L, then for all x, y*

$$\|\nabla f(x) - \nabla f(y)\|_2^2 \leq L\langle x - y, \nabla f(x) - \nabla f(y)\rangle.$$

Now let $\eta_k \overset{def}{=} \eta\zeta_k$. We have that

$$\begin{aligned}
\|w_{t+1} - w_*\|_2^2 &= \|w_t - w_* - \eta_t \nabla_w g_{i_t}(w_t)\|_2^2 \\
&= \|w_t - w_*\|_2^2 - 2\eta_t\langle w_t - w_*, \nabla_w g_{i_t}(w_t)\rangle + \eta_t^2\|\nabla_w g_{i_t}(w_t)\|_2^2 \\
&\leq \|w_t - w_*\|_2^2 - 2\eta_t\langle w_t - w_*, \nabla_w g_{i_t}(w_t)\rangle + 2\eta_t^2\|\nabla_w g_{i_t}(w_t) - \nabla_w g_{i_t}(w_*)\|_2^2 \\
&+ 2\eta_t^2\|\nabla_w g_{i_t}(w_*)\|_2^2 \quad \text{(Jensen's inequality)} \\
&\leq \|w_t - w_*\|_2^2 - 2\eta_t\langle w_t - w_*, \nabla_w g_{i_t}(w_t)\rangle \\
&+ 2\eta_t^2\frac{L_{i_t}}{rnp_{i_t}}\langle w_t - w_*, \nabla_w g_{i_t}(w_t) - \nabla_w g_{i_t}(w_*)\rangle + 2\eta_t^2\|\nabla_w g_{i_t}(w_*)\|_2^2 \quad \text{(Co-coercivity)}.
\end{aligned}$$

Taking the expectation with respect to i_t, and letting $\sigma^2(\mathbf{p}) = \frac{1}{n^2}\sum_{i=1}^n \frac{1}{p_i}\|\nabla_w f_i(w_*)\|_2^2$, we obtain

$$\begin{aligned}
\mathbb{E}\|w_{t+1} - w_*\|_2^2 &\leq \|w_t - w_*\|_2^2 - 2\frac{\eta_t}{r}\langle w_t - w_*, \nabla_w F(w_t)\rangle \\
&\quad + 2\eta_t^2 \mathbb{E}\frac{L_{i_t}}{rnp_{i_t}}\langle w_t - w_*, \nabla_w g_{i_t}(w_t) - \nabla_w g_{i_t}(w_*)\rangle + 2\frac{\eta_t^2}{r^2}\sigma^2(\mathbf{p}) \\
&\leq \|w_t - w_*\|_2^2 - 2\frac{\eta_t}{r}\langle w_t - w_*, \nabla_w F(w_t)\rangle \\
&\quad + 2\eta_t^2 \frac{1}{r} L_{sup}(\mathbf{p})\mathbb{E}\langle w_t - w_*, \nabla_w g_{i_t}(w_t) - \nabla_w g_{i_t}(w_*)\rangle + 2\frac{\eta_t^2}{r^2}\sigma^2(\mathbf{p}) \\
&\leq \|w_t - w_*\|_2^2 - 2\frac{\eta_t}{r}\langle w_t - w_*, \nabla_w F(w_t)\rangle \\
&\quad + 2\eta_t^2 \frac{1}{r^2} L_{sup}(\mathbf{p})\langle w_t - w_*, \nabla_w F(w_t)\rangle + 2\frac{\eta_t^2}{r^2}\sigma^2(\mathbf{p}).
\end{aligned}$$

Using strong convexity of F we have

$$\mathbb{E}\|w_{t+1} - w_*\|_2^2 \leq \left(1 - 2\mu\frac{\eta_t}{r}\left(1 - \frac{\eta_t}{r}L_{sup}(\mathbf{p})\right)\right)\|w_t - w_*\|_2^2 + 2\frac{\eta_t^2}{r^2}\sigma^2(\mathbf{p}).$$

Let $\lambda = 1 - 2\mu\frac{\eta}{r}\left(1 - \frac{\eta}{r}L_{sup}(\mathbf{p})\right)$ and $\chi_t = \sum_{j=1}^t \zeta_j$. Using this recursively, we obtain for all t

$$\begin{aligned}
\mathbb{E}\|w_{t+1} - w_*\|_2^2 &\leq \lambda^{\chi_t}\|w_0 - w_*\|_2^2 + 2\frac{\eta_t^2}{r^2}\sigma^2(\mathbf{p})\sum_{j=1}^t \lambda^{\chi_j} \\
&\leq \lambda^{\chi_t}\|w_0 - w_*\|_2^2 + 2\frac{\eta^2\sigma^2(\mathbf{p})}{r^2(1-\lambda)}.
\end{aligned}$$

Lemma 2 (Hoeffding's inequality) *Let $X_1, \ldots, X_n$ be independent random variables such that $a_i \leq X_i \leq b_i$ almost surely. Let $S_n = X_1 + \cdots + X_n$, then for all $z > 0$, we have that*

$$\mathbb{P}(|S_n - \mathbb{E}S_n| \geq t) \leq 2\exp\left(-\frac{2z^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

Using Hoefdding's inequality (Lemma 2) with $z = rt/2$, we have that with probability at least $1 - 2\exp(-\frac{r^2 t}{2})$

$$\chi_t \geq rt - rt/2 = rt/2.$$

Hence, with probability at least $1 - 2\exp(-\frac{r^2 t}{2})$, we have that

$$\mathbb{E}\|w_{t+1} - w_*\|_2^2 \leq \lambda^{rt/2}\|w_0 - w_*\|_2^2 + 2\frac{\eta^2\sigma^2(\mathbf{p})}{r^2(1-\lambda)}. \tag{6}$$

which concludes the first part of the proof.

Now let $\epsilon > 0$ and consider the following choice of the learning rate η

$$\eta = \frac{\frac{\mu\epsilon}{r}}{2\frac{1}{r^2}\sigma^2(\mathbf{p}) + 2\frac{\mu\epsilon}{r^2}L_{sup}(\mathbf{p})}.$$

With this choice, we have that

$$2\frac{\eta^2\sigma^2(\mathbf{p})}{r^2(1-\lambda)} = \frac{\eta\sigma^2(\mathbf{p})}{\mu r d\left(1 - \frac{\eta}{r}L_{sup}(\mathbf{p})\right)} \leq \frac{\epsilon}{2}.$$

Now it suffices to control the first term of Eq. (6) to conclude. Let $\epsilon_0 = \|w_0 - w_*\|_2^2$. In order to to have $\lambda^{rt/2}\epsilon_0 \leq \epsilon/2$, it is sufficient to have

$$\frac{rt}{2} \times \log(\lambda) \leq \log(\epsilon/2\epsilon_0).$$

As in Needell et al. [2016], we leverage the inequality $\frac{-1}{\log(1-x)} \leq 1/x$ with $x = 1 - \lambda$ to obtain a sufficient condition for this statement to hold. Indeed, using this inequality, it is straightforward that $\frac{rt}{2} \times \log(\lambda) \leq \log(\epsilon/2\epsilon_0)$ is satisfied whenever

$$t \geq 4 \log\left(\frac{2\epsilon_0}{\epsilon}\right) \frac{1}{r} \left(\frac{L_{sup}(\mathbf{p})}{\mu} + \frac{\sigma^2(\mathbf{p})}{\mu^2 \epsilon}\right).$$

B Additional experiments

B.1 Data distribution in the linear model

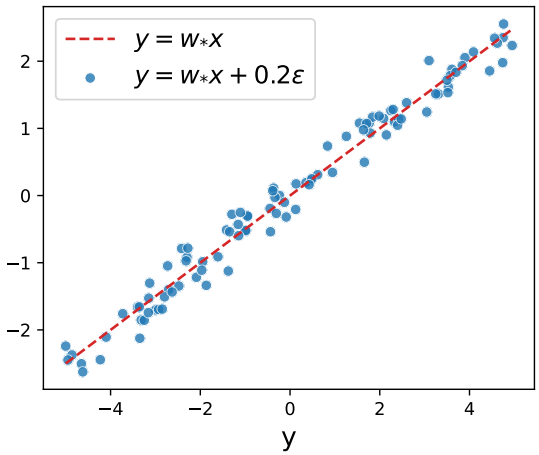


Figure 4: Samples from the dataset generated with Eq. (3).

B.2 Different learning rates

We show in Fig. 5 and Fig. 6 the results of `LiPrune` for different compression ratios with a different learning rates. The results show consistent training acceleration with `LiPrune` , compared to random pruning. In Fig. 6, the convergence is fast for both `LiPrune` and random pruning, with a small advantage for `LiPrune` .

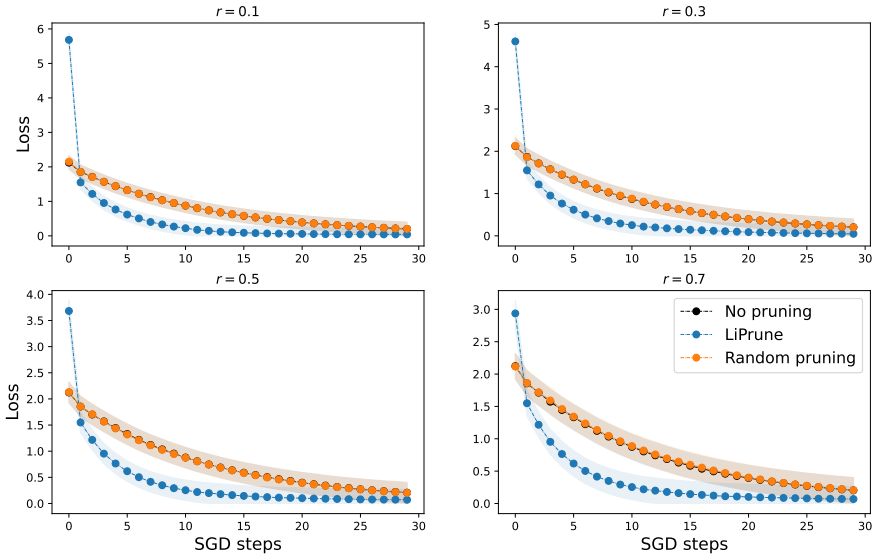


Figure 5: Experiments with the linear model (Eq. (3)) with batchsize$= 10$, learning rate=0.005.

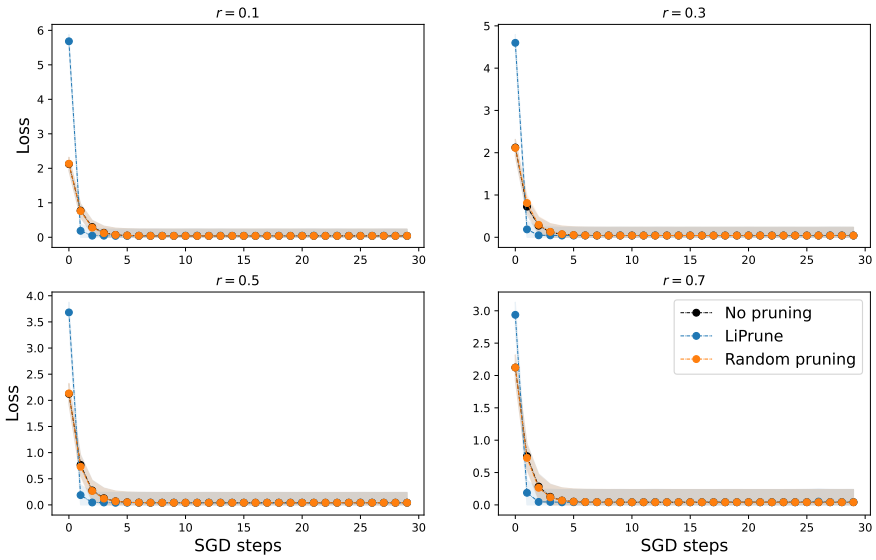


Figure 6: Experiments with the linear model (Eq. (3)) with batchsize$= 10$, learning rate=0.05.

B.3 Different batch sizes

In Fig. 7, Fig. 8, Fig. 9, we show the empirical results of `LiPrune` (Vs random pruning) for different choices of the batch size. `LiPrune` seems to be robust to batch size as well, in the case of the linear model. As the batch size increases (Fig. 9), the empirical advantage of `LiPrune` over random pruning becomes less significant, which is expected since the pruned sets obtained using `LiPrune` and random pruning have more and more shared samples.

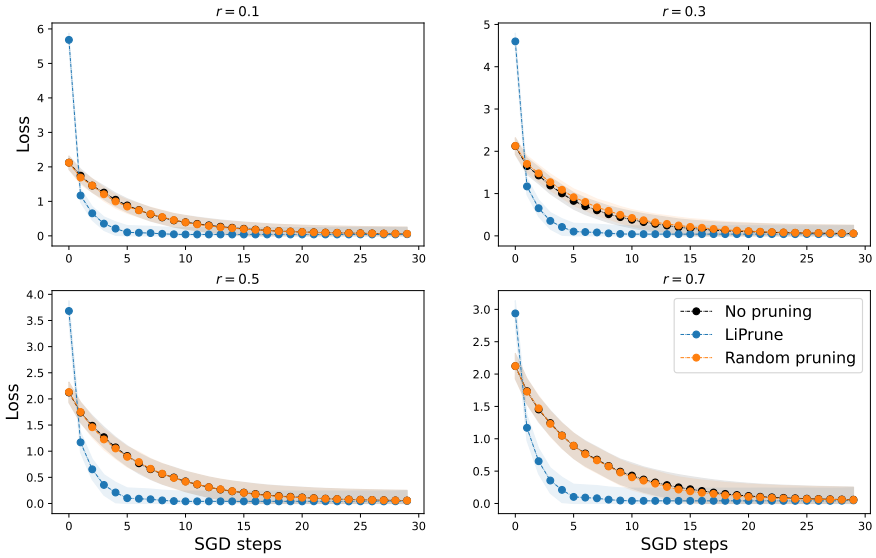


Figure 7: Experiments with the linear model (Eq. (3)) with batchsize$= 1$, learning rate=0.01.

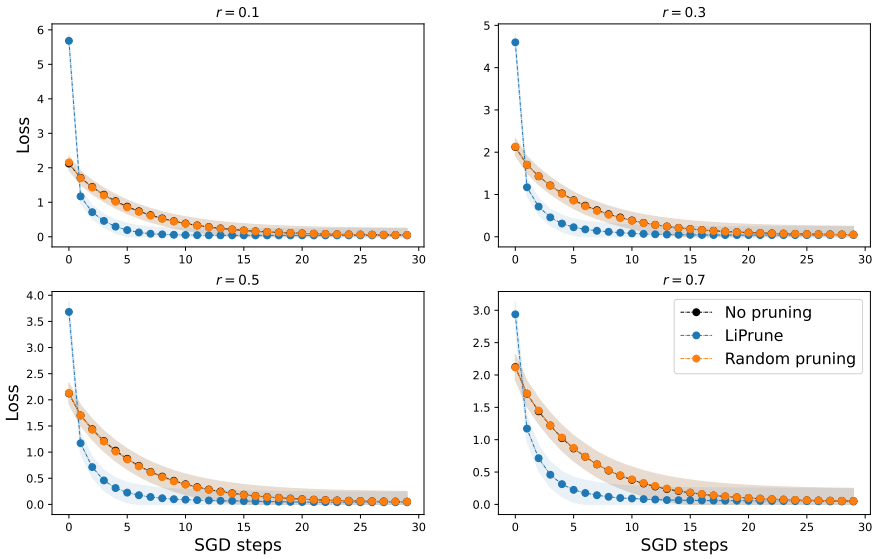


Figure 8: Experiments with the linear model (Eq. (3)) with batchsize$= 20$, learning rate=0.01.

B.4 Non-Convex case: a simple example

We consider a simple non-linear model where the data is generated using the following rule (for $i \in [N]$, $N = 1000$)

$$y_i = \sin(w_* x_i) + 0.1\,\varepsilon_i, \quad \varepsilon_i \overset{iid}{\sim} \mathcal{N}(0,1), \tag{7}$$

where $w_* = 12$ is fixed and $x_i \sim \mathcal{N}(0,1)$ (standard normal distribution). See Fig. 10 for an illustration of the dataset. We would like to fit a regression model of the form $y = \sin(wx)$ to

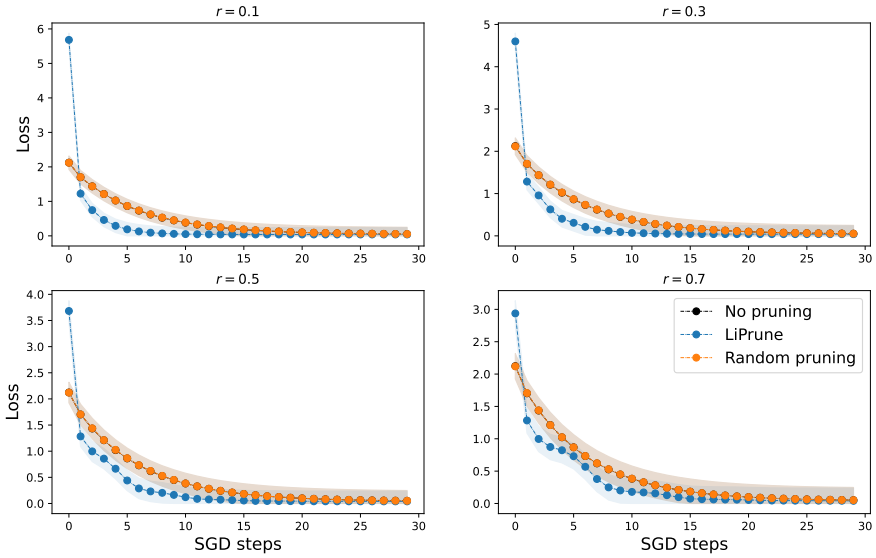


Figure 9: Experiments with the linear model (Eq. (3)) with batchsize$= 100$, learning rate=0.01.

the data $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, N\}$ by minimizing the mean squared loss. We show in Fig. 11 the empirical loss landscape with $N = 1000$ samples. The figure illustrates, as expected, a highly non-convex shape.

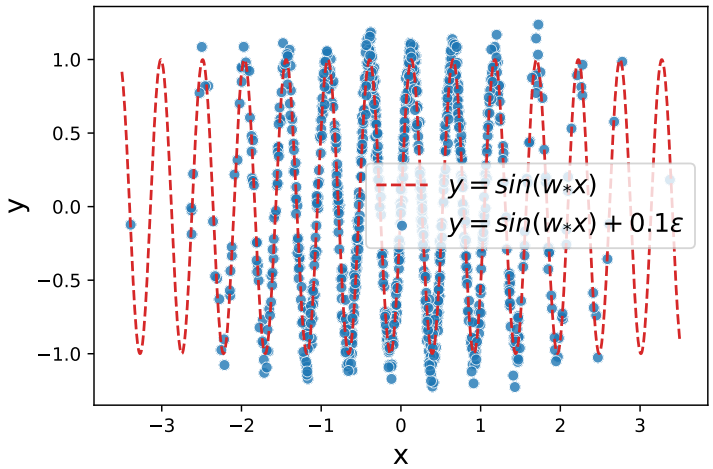


Figure 10: Samples from the data generation process given by Eq. (7)

in Fig. 12, we show the training curves obtained with `LiPrune` and random pruning. In this case, there is no clear (acceleration) advantage of `LiPrune` .

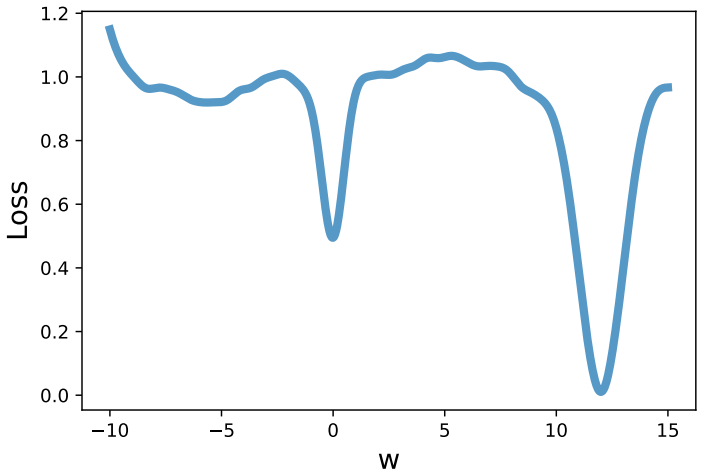


Figure 11: The empirical loss function obtained with $N = 1000$ samples from the generation process given by Eq. (7)

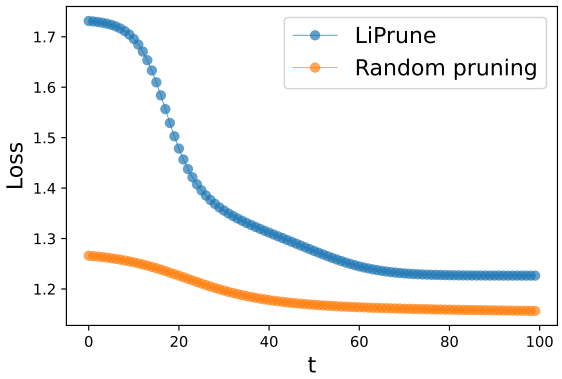
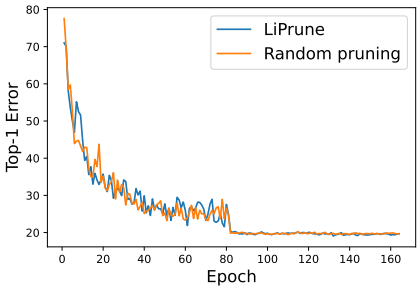


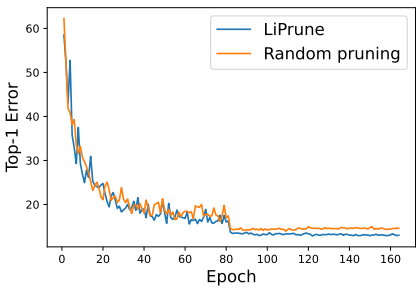
Figure 12: Comparison of the training loss of `LiPrune` and random pruning. The model is trained with 100 steps of SGD with batch size $= 10$, learning rate$= 0.2$, and initial weight $w_0 = 8$.

B.5 ResNet experiments

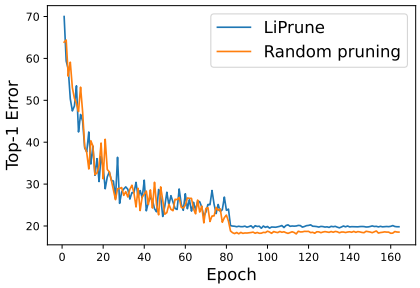
B.6 Experiments with learning rate 0.01, pruning iteration $it = 1$



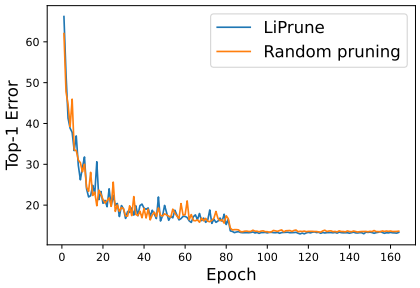
(a) $d = 14$, $r = 0.2$



(b) $d = 14$, $r = 0.5$



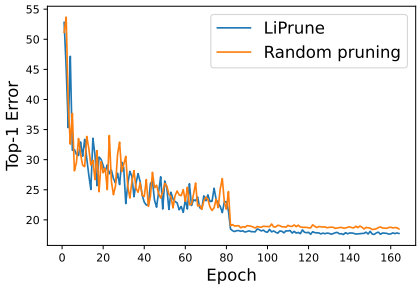
(c) $d = 20$, $r = 0.2$



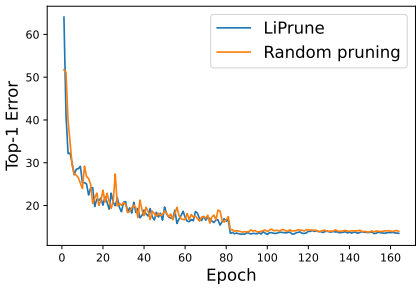
(d) $d = 20$, $r = 0.5$

Figure 13: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.2, 0.5\}$, and pruning iteration $\text{it} = 1$ ($\text{it} = 1$ corresponds to pruning at initialization).

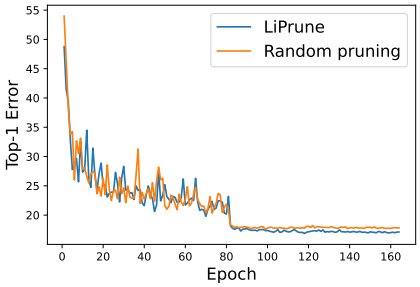
B.7 Experiments with learning rate 0.01, pruning iteration $it = 5$



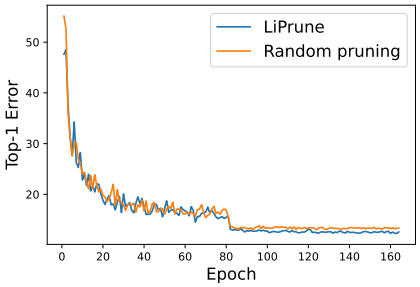
(a) $d = 14$, $r = 0.2$



(b) $d = 14$, $r = 0.5$



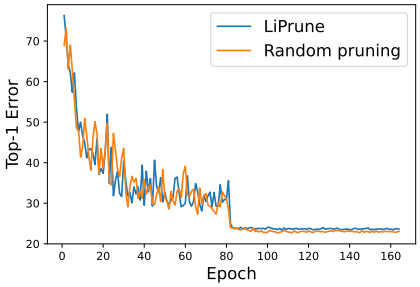
(c) $d = 20$, $r = 0.2$



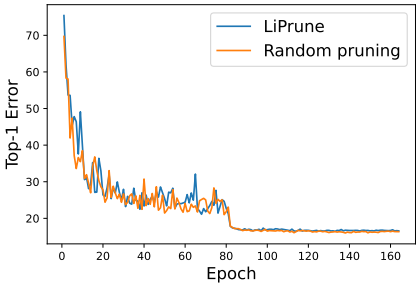
(d) $d = 20$, $r = 0.5$

Figure 14: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.2, 0.5\}$, and pruning iteration it $= 5$.

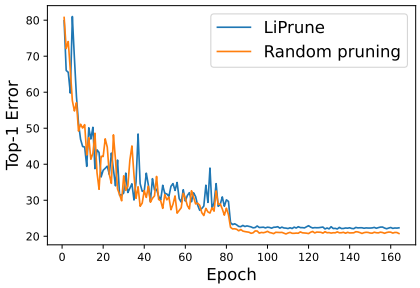
B.8 Experiments with learning rate 0.1, pruning iteration $it = 1$



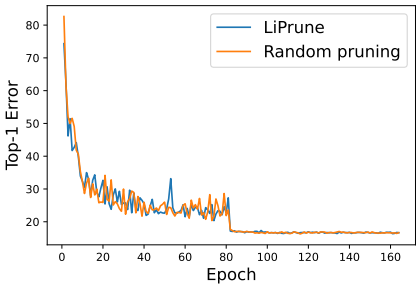
(a) $d = 14$, $r = 0.1$



(b) $d = 14$, $r = 0.2$

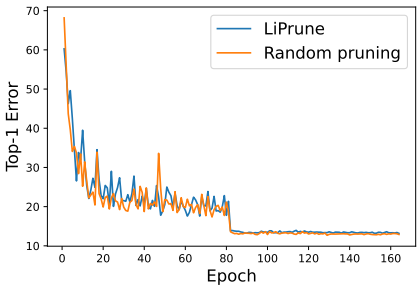


(c) $d = 20$, $r = 0.1$

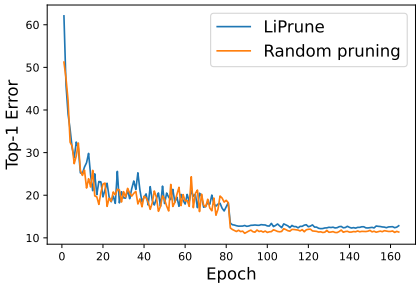


(d) $d = 20$, $r = 0.2$

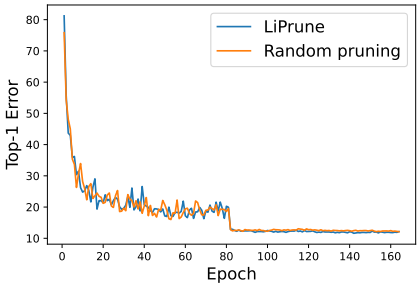
Figure 15: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.1, 0.2\}$, and pruning iteration $\text{it} = 1$ ($\text{it} = 1$ corresponds to pruning at initialization).



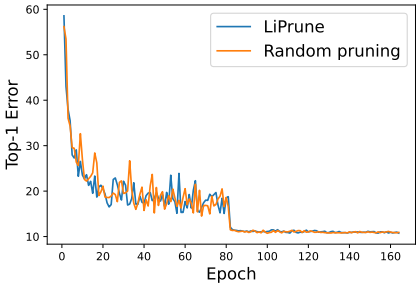
(a) $d = 14$, $r = 0.4$



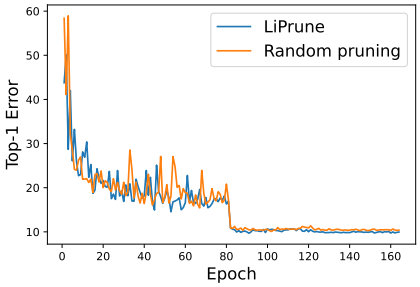
(b) $d = 14$, $r = 0.4$



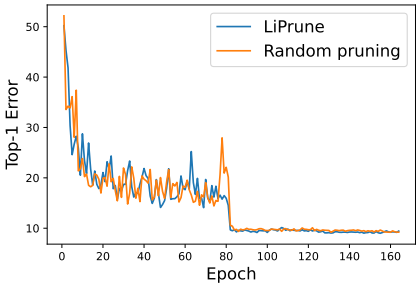
(c) $d = 20$, $r = 0.4$



(d) $d = 20$, $r = 0.5$



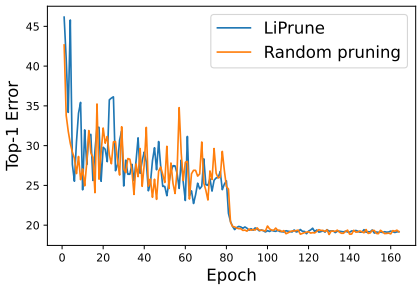
(e) $d = 14$, $r = 0.7$



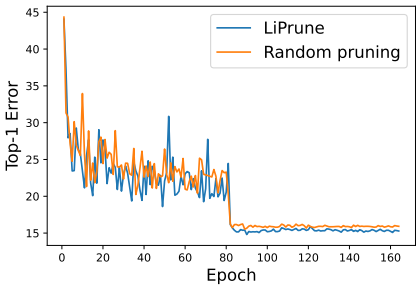
(f) $d = 20$, $r = 0.7$

Figure 16: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.4, 0.5, 0.7\}$, and pruning iteration $\text{it} = 1$ ($\text{it} = 1$ corresponds to pruning at initialization).

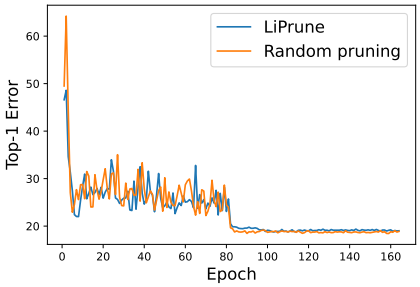
B.9 Experiments with learning rate 0.1, pruning iteration $it = 5$



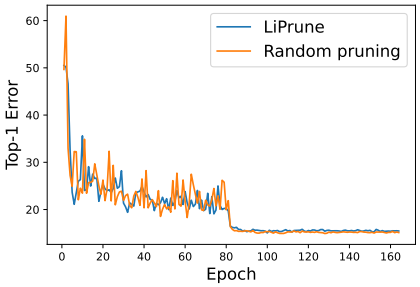
(a) $d = 14$, $r = 0.1$



(b) $d = 14$, $r = 0.2$

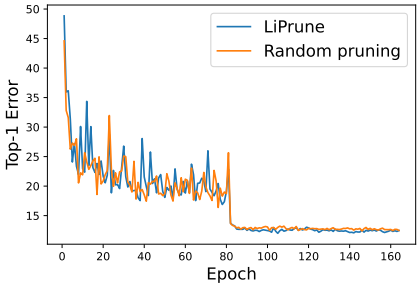


(c) $d = 20$, $r = 0.1$

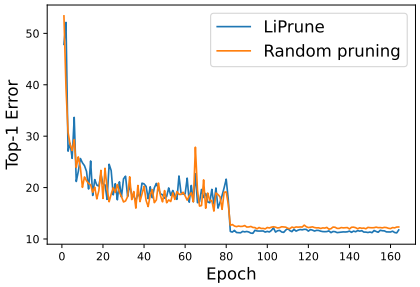


(d) $d = 20$, $r = 0.2$

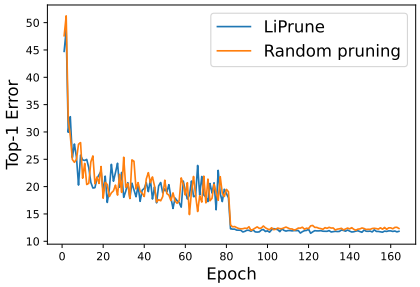
Figure 17: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.1, 0.2\}$, and pruning iteration it $= 5$.



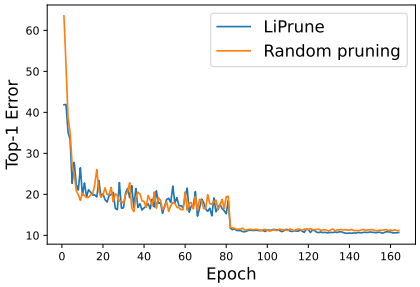
(a) $d = 14$, $r = 0.4$



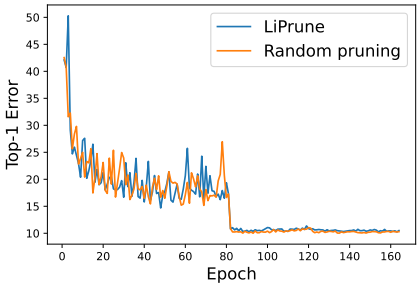
(b) $d = 14$, $r = 0.4$



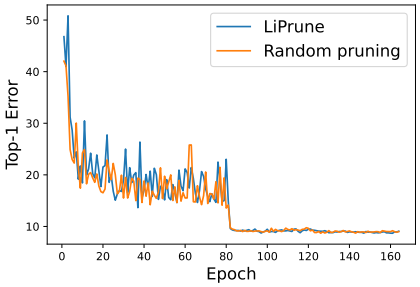
(c) $d = 20$, $r = 0.4$



(d) $d = 20$, $r = 0.5$



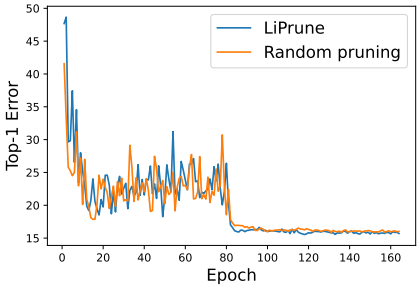
(e) $d = 14$, $r = 0.7$



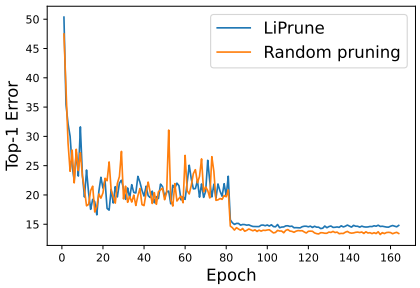
(f) $d = 20$, $r = 0.7$

Figure 18: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.4, 0.5, 0.7\}$, and pruning iteration $\text{it} = 5$.

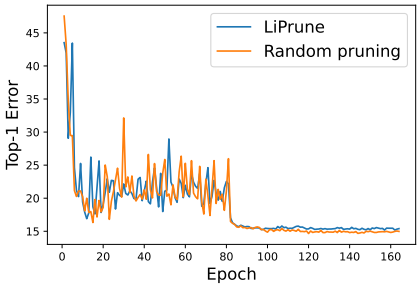
B.10 Experiments with learning rate 0.1, pruning iteration $it = 15$



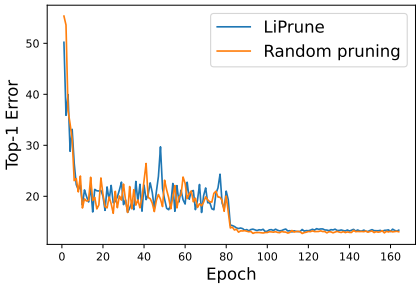
(a) $d = 14$, $r = 0.1$



(b) $d = 14$, $r = 0.2$

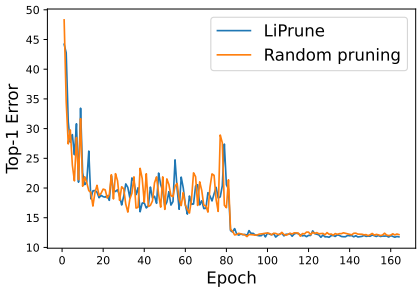


(c) $d = 20$, $r = 0.1$

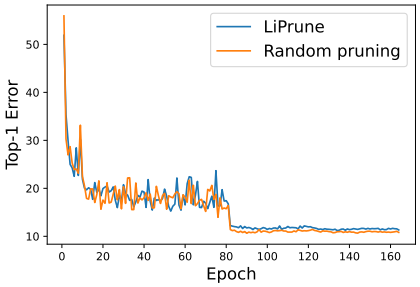


(d) $d = 20$, $r = 0.2$

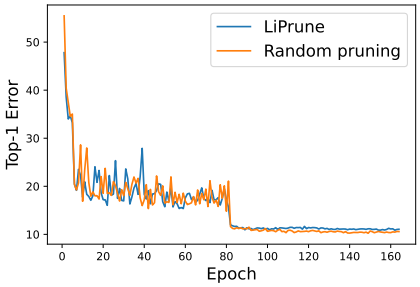
Figure 19: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.1, 0.2\}$, and pruning iteration $\text{it} = 15$.



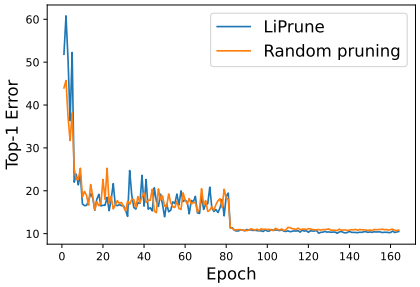
(a) $d = 14$, $r = 0.4$



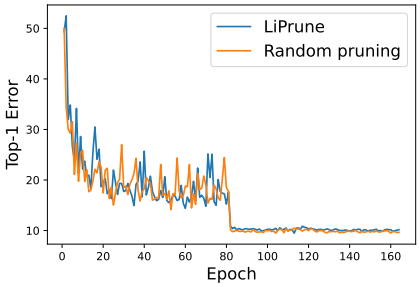
(b) $d = 14$, $r = 0.4$



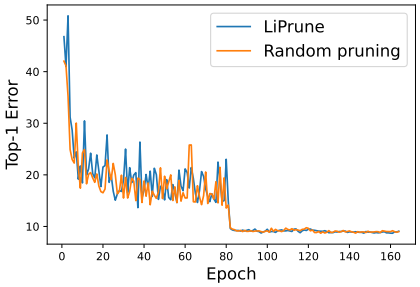
(c) $d = 20$, $r = 0.4$



(d) $d = 20$, $r = 0.5$



(e) $d = 14$, $r = 0.7$



(f) $d = 20$, $r = 0.7$

Figure 20: Empirical evaluation of `LiPrune` (versus Random pruning) on CIFAR10 dataset with ResNet architecture for different depths $d \in \{14, 20\}$, compression ratios $r \in \{0.4, 0.5, 0.7\}$, and pruning iteration $\text{it} = 15$.