Optimizing Learning Rate Schedules for Iterative Pruning of Deep Neural Networks

Shiyu Liu *shiyu_liu@u.nus.edu*
Department of Electrical and Computer Engineering
College of Design and Engineering
National University of Singapore

Rohan Ghosh *rghosh92@gmail.com*
Department of Electrical and Computer Engineering
College of Design and Engineering
National University of Singapore

John Chong Min Tan *johntancm@u.nus.edu.sg*
Department of Electrical and Computer Engineering
College of Design and Engineering
National University of Singapore

Mehul Motani *motani@nus.edu.sg*
Department of Electrical and Computer Engineering
College of Design and Engineering
N.1 Institute for Health
Institute of Data Science
Institute for Digital Medicine (WisDM)
National University of Singapore

Reviewed on OpenReview: *https://openreview.net/forum?id=nGW2Hotpq3*

Abstract

The importance of learning rate (LR) schedules on network pruning has been observed in a few recent works. As an example, *Frankle and Carbin (2019)* highlighted that winning tickets (i.e., accuracy preserving subnetworks) can not be found without applying a LR warmup schedule. *Renda, Frankle and Carbin (2020)* also demonstrated that rewinding the LR to its initial state at the end of each pruning cycle can improve pruning performance. In this paper, we go one step further by first providing a theoretical justification for the surprising effect of LR schedules. Next, we propose a LR schedule for network pruning called SILO, which stands for S-shaped Improved Learning rate Optimization. The advantages of SILO over existing LR schedules are two-fold: (i) SILO has a strong theoretical motivation and dynamically adjusts the LR during pruning to improve generalization. Specifically, SILO increases the LR upper bound (`max_lr`) in an S-shape. This leads to an improvement of 2% - 4% in extensive experiments with various types of networks (e.g., Vision Transformers, ResNet) on popular datasets such as ImageNet, CIFAR-10/100. (ii) In addition to the strong theoretical motivation, SILO is empirically optimal in the sense of matching an Oracle, which exhaustively searches for the optimal value of `max_lr` via grid search. We find that SILO is able to precisely adjust the value of `max_lr` to be within the Oracle optimized interval, resulting in performance competitive with the Oracle with significantly lower complexity.

1 Introduction

Network pruning is the process of simplifying neural networks by pruning weights, filters or neurons. (LeCun et al., 1990; Han et al., 2015b). Several state-of-the-art pruning methods (Renda et al., 2019; Frankle & Carbin, 2019) have demonstrated that a significant quantity of parameters can be removed without sacrificing accuracy. This greatly reduces the resource demand of neural networks, such as storage requirements and energy consumption (He et al., 2020; Wang et al., 2021; Good & et al, 2022; Vysogorets & Kempe, 2023).

The inspiring performance of pruning methods hinges on a key factor - Learning Rate (LR). Specifically, Frankle & Carbin (2019) proposed the Lottery Ticket Hypothesis and demonstrated that the winning tickets (i.e., the pruned network that can train in isolation to full accuracy) can not be found without applying a LR warmup schedule. In a follow-up work, Renda et al. (2019) proposed LR rewinding which rewinds the LR schedule to its initial state during iterative pruning and demonstrated that it can outperform standard fine-tuning. In summary, the results in both works suggest that, besides the pruning metric, LR also plays an important role in network pruning and could be another key to improving the pruning performance.

In this paper, we take existing studies one step further and aim to optimize the choice of LR for iterative network pruning. We explore a new perspective on adapting the LR schedule to improve the iterative pruning performance. In the following, we summarize the workflow of our study together with our contributions.

1. **Motivation and Theoretical Study.** In Section 3.1, we explore the optimal choice of LR during network pruning and find that the distribution of weight gradients tends to become narrower during pruning, suggesting that a larger value of LR should be used to retrain the pruned network. This finding is further verified by our theoretical development in Section 3.2. More importantly, our theoretical results suggest that the optimal increasing trajectory of LR should follow an S-shape.

2. **Proposed SILO.** In Section 4, we propose a novel LR schedule for network pruning called SILO, which stands for S-Shaped Improved Learning rate Optimization. Motivated by our theoretical development, SILO precisely adjusts the LR by increasing the LR upper bound (`max_lr`) in an S-shape. We highlight that SILO is method agnostic and works well with numerous pruning methods.

3. **Experiments.** In Section 5.2, we compare SILO to four LR schedule benchmarks via both classical and state-of-the-art (SOTA) pruning methods. We observe that SILO outperforms LR schedule benchmarks, leading to an improvement of 2% - 4% in extensive experiments with various networks (e.g., Vision Transformer (Dosovitskiy et al., 2020), ResNet (He et al., 2016), VGG (Simonyan & Zisserman, 2014), DenseNet-40 (Huang et al., 2017) and MobileNetV2 (Sandler et al., 2018)) on large-scale datasets such as ImageNet (Deng et al., 2009) and popular datasets such as CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009).

4. **Comparison to Oracle.** In Section 5.4, we examine the optimality of SILO by comparing it to an Oracle which exhaustively searches for the optimal value of `max_lr` via grid search. We find that SILO is able to precisely adjust `max_lr` to be within the Oracle's optimized `max_lr` interval at each pruning cycle, resulting in performance competitive with the Oracle, but with significantly lower complexity.

2 Background

In Section 2.1, we first review prior works on network pruning. Next, in Section 2.2, we highlight the important role of LR in network pruning and position our work in the context of network pruning.

2.1 Prior Works on Network Pruning

Network pruning is an established idea dating back to 1990 (LeCun et al., 1990). The motivation is that networks tend to be overparameterized and redundant weights can be removed with a negligible loss in accuracy (Arora et al., 2018; Allen-Zhu et al., 2019; Denil et al., 2013). Given a trained network, one **pruning cycle** consists of three steps: (i) Prune the network according to certain heuristics; (ii) Freeze pruned parameters as zero. (iii) Retrain the pruned network to recover the accuracy. Repeating the pruning cycle multiple times until the target sparsity or accuracy is met is known as **iterative pruning**. Doing so often results in better performance than **one-shot pruning** (i.e., perform only one pruning cycle) (Han et al., 2015b; Frankle & Carbin, 2019; Li et al., 2017). There are two types of network pruning - unstructured pruning and structured pruning - which will be discussed in detail below.

Unstructured Pruning removes individual weights according to certain heuristics, such as magnitude (Han et al., 2015b; Frankle & Carbin, 2019) or gradient (Hassibi & Stork, 1993; LeCun et al., 1990; Lee et al., 2019; Xiao et al., 2019; Theis et al., 2018). Examples are (LeCun et al., 1990), which performed pruning based on the Hessian Matrix, and (Theis et al., 2018), which used Fisher information to approximate the Hessian Matrix. Similarly, Han et al. (2015b) removed weights with the smallest magnitude, and this approach was further incorporated with the three-stage iterative pruning pipeline in (Han et al., 2015a).

Structured Pruning involves pruning weights in groups, neurons, channels or filters (Yang et al., 2019; Molchanov et al., 2017; 2019; Luo et al., 2017; Yu et al., 2018; Tan & Motani, 2020; Wang et al., 2020b; Lin et al., 2020; Zhang & Freris, 2023). Examples are (Hu et al., 2016), which removed neurons with high average zero output ratio, and (Li et al., 2017), which pruned neurons with the lowest absolute summation values of incoming weights. More recently, Yu et al. (2018) proposed the neuron importance score propagation algorithm to evaluate the importance of network structures. Molchanov et al. (2019) used Taylor expansions to approximate a filter's contribution to the final loss and Wang et al. (2020a) optimized the neural network architecture, pruning policy, and quantization policy together in a joint manner.

Other Works. In addition to works mentioned above, several other works also share some deeper insights in network pruning (Liu et al., 2019b; Wang et al., 2020c; Li & et al, 2022; Wang & et al, 2022). For example, Liu et al. (2019a) demonstrated that training-from-scratch on the right sparse architecture yields better results than pruning from pre-trained models. Similarly, Wang et al. (2020c) suggested that the fully-trained network could reduce the search space for the pruned structure. More recently, Luo & Wu (2020) addressed the issue of pruning residual connections with limited data and Ye et al. (2020) theoretically proved the existence of small subnetworks with lower loss than the unpruned network. You & et al (2022) motivated the use of the affine spline formulation of networks to analyze recent pruning techniques. Liu et al. (2022) applied the network pruning technique in graph networks and approximated the subgraph edit distance. One milestone paper (Frankle & Carbin, 2019) pointed out that re-initializing with the original parameters (known as weight rewinding) plays an important role in pruning and helps to further prune the network with negligible loss in accuracy. Some follow-on works (Zhou et al., 2019; Renda et al., 2019; Malach et al., 2020) investigated this phenomenon more precisely and applied this method in other fields (e.g., transfer learning (Mehta, 2019) and natural language processing (Yu et al., 2020)).

2.2 The Important Role of Learning Rate

Several recent works (Renda et al., 2019; Frankle & Carbin, 2019) have noticed the important role of LR in network pruning. For example, Frankle & Carbin (2019) demonstrated that training VGG-19 with a LR warmup schedule (i.e., increase LR to 1e-1 and decrease it to 1e-3) and a constant LR of 1e-2 results in comparable accuracy for the unpruned network. However, as the network is iteratively pruned, the LR warmup schedule leads to a higher accuracy (see Fig.7 in (Frankle & Carbin, 2019)). In a follow-up work, Renda et al. (2019) further investigated this phenomenon and proposed a retraining technique called LR rewinding which can always outperform the standard retraining technique called fine-tuning (Han et al., 2015b). The difference is that fine-tuning trains the unpruned network with a LR warmup schedule, and retrains the pruned network with a constant LR (i.e., the final LR of the schedule) in subsequent pruning cycles (Liu et al., 2019b). LR rewinding retrains the pruned network by rewinding the LR warmup schedule to its initial state, namely that LR rewinding uses the same schedule for every pruning cycle. As an example, they demonstrated that retraining the pruned ResNet-50 using LR rewinding yields higher accuracy than fine-tuning (see Figs.1 & 2 in (Renda et al., 2019)). In (Liu & et al, 2021), the authors also suggest that when pruning happens during the training phase with a large LR, models can easily recover from pruning than using a smaller LR. Overall, The results in these works suggest that, besides the pruning metric, LR also plays an important role in network pruning and could be another key to improving network pruning.

Our work. In this paper, we explore a new perspective on adapting the LR schedule to improve the iterative pruning performance of ReLU-based networks. The proposed LR schedule is method agnostic and can work well with numerous pruning methods. We mainly focus on iterative pruning of ReLU-based networks for two

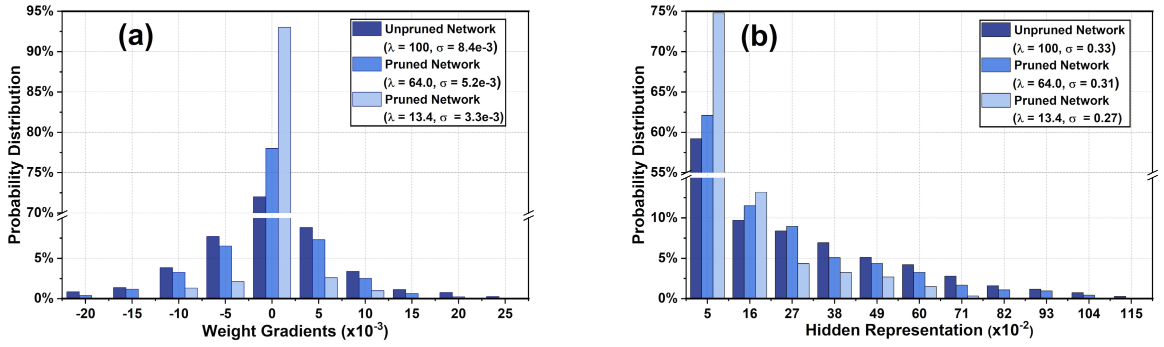


Figure 1: (a) The distribution of all weight gradients when iteratively pruning a fully connected ReLU network using global magnitude, where λ is the percent of weights remaining and σ is the standard deviation of the distribution. (b) The corresponding distribution of hidden representations.

reasons: (i) Iterative pruning tends to provide better pruning performance than one-shot pruning as reported in the literature (Frankle & Carbin, 2019; Renda et al., 2019). (ii) ReLU has been widely used in many classical neural networks (e.g., ResNet, VGG, DenseNet) which have achieved outstanding performance in various tasks (e.g., image classification, object detection) (He et al., 2016; Simonyan & Zisserman, 2014).

3 A New Insight on Network Pruning

In Section 3.1, we first provide a new insight in network pruning using experiments. Next, in Section 3.2, we provide a theoretical justification for our observed new insight and present some relevant theoretical results.

3.1 Weight Gradients during Iterative Pruning

(1) Experiment Setup. To exclude the influence of other factors, we start from a simple fully connected ReLU-based network with three hidden layers of 256 neurons each (results of other popular networks are summarized later). We train the network using the training dataset of CIFAR-10 via SGD (Ruder, 2016) (momentum = 0.9 and a weight decay of 1e-4) with a batch size of 128 for 500 epochs. All hyperparameters are tuned for performance via grid search (e.g., LR from 1e-4 to 1e-2). We apply the global magnitude (Han et al., 2015b) (i.e., remove weights with the smallest magnitude anywhere in the network) with a pruning rate of 0.2 (i.e., prune 20% of the remaining parameters) to iteratively prune the network for 10 pruning cycles and plot the distribution of all weight gradients when the network converges in Fig. 1(a), where λ is the percent of weights remaining. In Fig. 1(a), there are 10 visible bins estimated by the Sturges' Rule (Scott, 2009) and each bin consists of three values (i.e., the probability distribution of networks with different λ). The edge values range from -0.022 to 0.027 with a bin width of 0.004.

(2) Experiment Results. In Fig. 1(a), we observe that the distribution of weight gradients tends to become narrower, i.e., the standard deviation of weight gradients σ reduces from 8.4e-3 to 3.3e-3 when the network is iteratively pruned to $\lambda = 13.4$. As an example, the unpruned network ($\lambda = 100$) has more than 7% of weight gradients with values greater than 0.008 (rightmost 4 bars) or less than -0.012 (leftmost 2 bars), while the pruned network ($\lambda = 13.4$) has less than 1% of weight gradients falling into those regions. It suggests that the magnitude of weight gradients tends to decrease as the network is iteratively pruned.

(3) New Insight. During the backpropagation, the weight update of w_i is $w_i \leftarrow w_i + \alpha\frac{\partial\mathcal{L}}{\partial w_i}$, where α is the LR and $\mathcal{L}$ is the loss function. Assume that α is well-tuned to ensure the weight update (i.e., $\alpha\frac{\partial\mathcal{L}}{\partial w_i}$) is sufficiently large to prevent the network from getting stuck in local optimal points (Bengio, 2012; Goodfellow et al., 2016). As shown in Fig. 1(a), the magnitude of the weight gradient (i.e., $\frac{\partial\mathcal{L}}{\partial w_i}$) tends to decrease as the network is iteratively pruned. To preserve the same weight updating size and effect as before, a gradually larger value of LR (i.e., α) should be used to retrain the pruned network during iterative pruning.

(4) Result Analysis. We now provide an explanation for the change in the distribution of weight gradients. First, we assume each $x_i w_i$ (i.e., $x_i \in \mathbb{R}$ is the neuron input and $w_i \in \mathbb{R}$ is the associated weight) is an i.i.d. random variable. Then, the variance of the neuron's pre-activation output ($\sum_{i=1}^{n} x_i w_i$, n is the number of inputs) will be $\sum_{i=1}^{n} \mathrm{Var}(x_i w_i)$. Pruning the network is equivalent to reducing the number of inputs from n to $n-k$. This results in a smaller variance of $\sum_{i=1}^{n-k} \mathrm{Var}(x_i w_i)$, leading to a smaller standard deviation. Hence, the distribution of the pre-activation output after pruning is narrower. Since ReLU returns its raw input if the input is non-negative, the distribution of hidden representations (output of hidden layers) becomes narrower as well. This can be verified from Fig. 1(b), where we plot the distribution of hidden representations from the previous experiment. The key is that the weight gradient $\frac{\partial \mathcal{L}}{\partial w_i}$ is proportional to the hidden representation x_i that associates with w_i (i.e., $\frac{\partial \mathcal{L}}{\partial w_i} \propto x_i$). As the network is iteratively pruned, the distribution of hidden representations becomes narrower, leading to a narrower distribution of weight gradients. As a result, a larger LR should be used to retrain the pruned network.

(5) More Generalized Results. (i) Effect of Batch Normalization (BN) (Ioffe & Szegedy, 2015): BN is a popular technique to reformat the distribution of hidden representations, so as to address the issue of internal covariate shift. We note that similar performance trends can be observed after applying BN as well (see Fig. 4 in the Appendix). (ii) Popular CNN Networks and Pruning Methods: In addition to the global magnitude used before, two unstructured pruning methods (i.e., layer magnitude, global gradient) suggested by (Blalock et al., 2020) and one structured pruning method (`L1` norm pruning) (Li et al., 2017) are examined as well. Those methods are used to iteratively prune AlexNet (Krizhevsky & Hinton, 2010), ResNet-20 and VGG-19 using CIFAR-10. The results using these popular neural networks largely mirror those in Figs. 1(a) & (b) as well. We refer the interested reader to Figs. 5 - 7 in the Appendix.

3.2 Theoretical Study and Motivation

In this subsection, we theoretically investigate how network pruning can influence the value of the desired LR. The proofs of the results given here are provided in the Appendix. First, we present some definitions.

Definition 1. Average Activation Norm (E_{AA}): Given a network with fixed weights, input X from a distribution P, and a layer $H = \{h_1(X), ..., h_N(X)\}$ with N nodes where $h_i(X)$ represents the function at the i^{th} node. Then $E_{AA}(H) = \mathbb{E}_X[\frac{1}{N}\sum_i h_i(X)^2]$. This quantity reflects the average strength of the layer's activations.

Definition 2. Average Gradient Norm (E_{WG}): Let $W = [w_1, ..., w_k]$ and $W' = [w'_1, ..., w'_k]$ represent the flattened weight vector before and after one epoch of training (via backpropagation). Then E_{WG} is the average change in weight magnitude in a single training epoch (for the active unpruned weights), i.e., $E_{WG}(W, W') = \mathbb{E}_i[(w_i - w'_i)^2]$. E_{WG} quantifies how much the weights change after one epoch of training.

We now demonstrate the impact of network pruning on the *average activation norm* of hidden layers.

Theorem 1. Consider a ReLU activated neural network represented as $X \xrightarrow{W_1} H \xrightarrow{W_2} Y$, where $X \in \mathbb{R}^d$ is the input, $H = \{H_1(X), H_2(X), ..., H_N(X)\}$ is of infinite width ($N = \infty$), and Y is the network output. W_1 and W_2 represent network parameters (weights, biases). Furthermore, let $X \sim \mathcal{N}(0, \sigma_x^2 I)$ and $W_1 \sim \mathcal{N}(0, \sigma_w^2 I)$, where I is the identity matrix and σ_x, σ_w are scalars. Now, let us consider an iterative pruning method, where in each iteration a fraction $0 \le p \le 1$ of the smallest magnitude weights are pruned (layer-wise pruning). Then, after k iterations of pruning, it holds that

$$4E_{AA}(H) \ge \sigma_W^2 + d\sigma_X^2\sigma_W^2\Big((1-p)^k + \sqrt{\frac{4}{\pi}}\,\mathrm{erf}^{-1}\Big(1-(1-p)^k\Big)e^{-\left(\mathrm{erf}^{-1}\left(1-(1-p)^k\right)\right)^2}\Big) \tag{1}$$

where $\mathrm{erf}^{-1}(.)$ is the inverse error function.

Next, based on the above result, the following theorem establishes how the *average gradient norm* depends on the LR of the neural network, and the pruning iteration.

Theorem 2. In the setting of Theorem 1, we consider a single epoch of weight update for the network across a training dataset $S = \{(X_1, Y_1), .., (X_n, Y_n)\}$ using the cross-entropy loss, where $Y_i \in \{0, 1\}$. Let

α denote the learning rate. Let us denote the R.H.S of equation 1 by $C(\sigma_X, \sigma_W, p, k)$. Let the final layer weights before and after one training epoch be W_2 and W_2' respectively. We have,

$$\mathop{\mathbb{E}}_{W_2 \sim \mathcal{N}_k(0,\sigma_W^2 I)} \left[E_{WG}\left(W_2, W_2'\right)\right] \geq \alpha^2 \gamma C(\sigma_X, \sigma_W, p, k), \tag{2}$$

for some constant γ, where $\mathcal{N}_k(0, \sigma_w^2 I)$ represents the distribution of W_2 after being initialized as the Gaussian $\mathcal{N}(0, \sigma_w^2 I)$ and pruned for k iterations.

Remark 1. (Pruning and LR) Theorems 1 and 2 together establish how the choice of LR influences the lower bound of *average gradient norm*. Theorem 1 shows that the lower bound of *activation norm* of the hidden layer decreases as the network is pruned, and as Theorem 2 shows, this also reduces the lower bound of *average gradient norm* per epoch. Thus, to counter this reduction, it is necessary to increase the learning rate α as the number of pruning cycles grows, in order to ensure that the R.H.S of equation 2 remains fixed.

Remark 2. (S-shape of LR During Iterative Pruning) We fix the average gradient norm to negate the impact of weight gradients reducing while pruning (see Fig. 1). Theorem 2 implies that to maintain a fixed *average gradient norm* of $\mathbb{E}_{W_2 \sim \mathcal{N}_k(0,\sigma_W^2 I)}[E_{WG}(W_2, W_2')] = K$, we must have the learning rate $\alpha \leq (K/\gamma C(\sigma_X, \sigma_W, p, k))^{1/2}$. We find that this upper bound resembles an S-shape trajectory during iterative pruning (see the red line in Fig. 2).

Remark 3. (Additional Results) Note that we extend Theorem 1's result to the case of fully connected neural networks of arbitrary depth in Proposition 1 of the Appendix. Similarly, we extend Theorem 2's result to the case of networks of arbitrary depth in Corollary 1 of the Appendix. Although Theorem 1 assumes 2-layer neural networks of infinite width, we extend it to the case of finite hidden neurons (see Proposition 2 of Appendix), yielding a probabilistic bound of the same form as Theorem 1. Note that the results in both Theorems are empirically verified in Section B of the Appendix.

4 A New Learning Rate Schedule

In Section 4.1, we shortlist four LR benchmarks for comparison. In Section 4.2, we introduce SILO and highlight the difference with existing works. In Section 4.3, we detail the algorithm for SILO.

4.1 LR Schedule Benchmarks

Learning rate is the most important hyperparameter in training neural networks (Goodfellow et al., 2016). The LR schedule is to adjust the value of LR during training by a pre-defined schedule. Three common LR schedules are summarized as follows.

1. **LR Decay** starts with a large LR and linearly decays it by a certain factor after a pre-defined number of epochs. Several recent works (You et al., 2019; Ge et al., 2019; An et al., 2017) have demonstrated that decaying LR helps the neural network to converge better and avoids undesired oscillations in optimization.

2. **LR Warmup** is to increase the LR to a large value over certain epochs and then decreases the LR by a certain factor. It is a popular schedule used by many practitioners for transfer learning (He et al., 2019) and network pruning (Frankle & Carbin, 2019; Frankle et al., 2020).

3. **Cyclical LR** (Smith, 2017) varies the LR cyclically between a pre-defined lower and upper bound. It has been widely used in many tasks (You et al., 2019).

All the three LR schedules described above and constant LR will be used as **benchmarks for performance comparison**. We note that, in addition to LR schedules which vary LR by a pre-defined schedule, adaptive LR optimizers such as AdaDelta (Zeiler, 2012) and Adam (Kingma & Ba, 2014) provide heuristic based approaches to adaptively vary the step size of weight update based on observed statistics of the past gradients. All of them are sophisticated optimization algorithms and much work (Gandikota et al., 2021; Jentzen et al., 2021) has been done to investigate their mechanisms. In this paper, the performance of all benchmarks and SILO will be evaluated using SGD with momentum = 0.9 and a weight decay of 1e-4 (same as (Renda et al., 2019; Frankle & Carbin, 2019)). The effect of those adaptive LR optimizers on SILO will be discussed in Section 6.

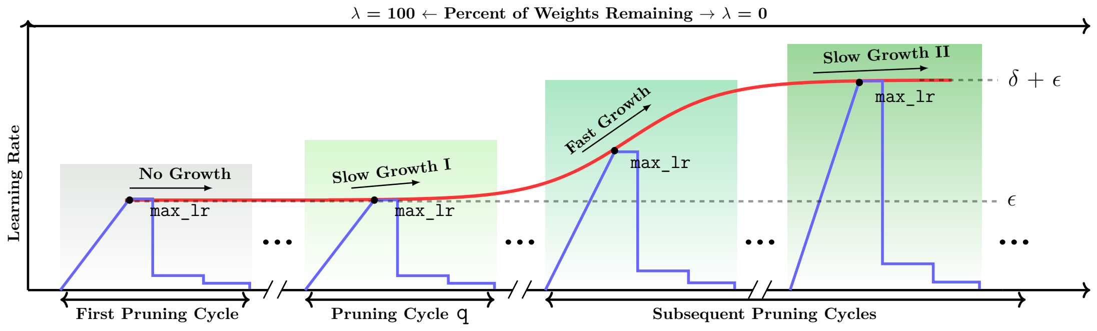


Figure 2: Illustration of SILO during pruning. The S-shape red line is motivated from Theorem 2.

4.2 SILO Learning Rate Schedule

To ensure the pruned network is properly trained during iterative pruning, we propose an S-shaped Improved Learning rate Schedule, called SILO, for iterative pruning of networks. As illustrated in Fig. 2, the main idea of the proposed SILO is to apply the LR warmup schedule at every pruning cycle, with a gradual increase of the LR upper bound (i.e., `max_lr`) in an S-shape as the network is iteratively pruned. This LR warmup schedule is meant to be flexible and can change depending on different networks and datasets.

The S-shape in SILO is inspired by Theorem 2 (see Remark 2) and will be further verified by comparing to an Oracle. We divide the S-shape into four phases and provide the intuition behind each phase as follows.

1. Phase-1: **No Growth**, SILO does not increase `max_lr` until the pruning cycle `q` (see Fig. 2). It is because the unpruned network often contains a certain amount of weights with zero magnitude. Those parameters are likely to be pruned at the first few pruning cycles, and removing such weights has negligible effect on the distribution of weight gradients.
2. Phase-2: **Slow Growth I**, the pruning algorithm has removed most zero magnitude weights and started pruning weights with small magnitude. Pruning such weights has a small effect on distribution of weight gradients. Hence, we slightly increase `max_lr` after pruning cycle `q`.
3. Phase-3: **Fast Growth**, SILO greatly increases `max_lr`. It is because the pruning algorithm now starts removing weights with large magnitude and the distribution of weight gradients becomes much narrower. This requires a much larger LR for meaningful weight updates.
4. Phase-4: **Slow Growth II**, the network is now heavily pruned and very few parameters left in the network. By using the same pruning rate, a very small portion of the weights will be pruned. This could cause a marginal effect on the distribution of weight gradients. Hence, SILO slightly increases `max_lr`.

We note that SILO is designed based on the assumption that existing pruning methods tend to prune weights with small magnitude. **The key difference with existing LR schedules** (e.g., LR warmup) is that SILO is adaptive and able to **precisely increase** the value of `max_lr` as the network is iteratively pruned, while existing LR schedules do not factor in the need to change `max_lr` during different pruning cycles.

4.3 Implementation of SILO

As for the implementation of SILO, we designed a function to estimate the value of `max_lr` as shown below.

$$\texttt{max\_lr} = \frac{\delta}{1 + (\frac{\gamma}{1-\gamma})^{-\beta}} + \epsilon, \tag{3}$$

where $\gamma = 1 - (1 - \texttt{p})^{m-\texttt{q}}$ is the input of the function and `max_lr` is the output of the function. When computing the value of γ, the parameter `p` is the pruning rate and `m` is the number of completed pruning cycles. In equation 3, the parameters β and `q` are used to control the shape of the S curve. The larger the β, the later the curve enters the Fast Growth phase. The parameter `q` determines at which pruning cycle SILO enters the Slow Growth I phase. When $q = 0$, the No Growth phase will be skipped and γ will be

Algorithm 1 Algorithm for the proposed S-shaped Improved Learning rate Optimization (SILO)

```
Input: lower bound $\epsilon$, upper bound $\delta + \epsilon$, pruning rate p, number of pruning cycles L, number of training
    epochs t, S-shape control term $\beta$, delay term q.
1: for m = 0 to L do
2:     if m $\leq$ q then
3:         max_lr = $\epsilon$
4:     else
5:         max_lr = $\frac{\delta}{1+(\frac{\gamma}{1-\gamma})^{-\beta}} + \epsilon$, $\gamma = 1 - (1 - \mathtt{p})^{m-\mathtt{q}}$
6: for i = 0 to t do
7:     (1) linearly warmup the LR to max_lr
8:     (2) drop the value of LR by 10 at certain epochs
```

the proportion of pruned weights at the current pruning cycle. The parameters ϵ and δ determine the value range of `max_lr`. As the network is iteratively pruned, γ increases and `max_lr` increases from ϵ to $\epsilon + \delta$ accordingly. The details of the SILO algorithm are summarized in Algorithm 1.

Parameter Selection for SILO. Algorithm 1 requires several inputs for implementation. The value of ϵ can be tuned using the validation accuracy of the unpruned network while the value of δ can be tuned using the validation accuracy of the pruned network with targeted sparsity. The pruning rate `p` and pruning cycles `L` are chosen to meet the target sparsity. The number of training epochs `t` should be large enough to guarantee the network convergence. Let $\mathtt{q} = 1$ and $\beta = 5$ could be a good choice and yield promising results as we demonstrate in Section 5. Furthermore, based on our experience, the value of `q` and β could be tuned in the range of [0, 3] and [3, 6], respectively. Lastly, we note that the sensitivity of parameters in the proposed SILO will be further examined in Section 5.3.

5 Performance Evaluation

In Sections 5.1, we first summarize the experiment setup. In Section 5.2, compare the performance of the proposed SILO to four LR schedule benchmarks. In Section 5.3, we examine the sensitivity of parameters in SILO. In Section 5.4, we present the value of `max_lr` estimated by the proposed SILO at each pruning cycle and compare it to an Oracle which exhaustively searches for the optimal `max_lr` via grid search.

5.1 Experimental Setup

We demonstrate that SILO can work well with different pruning methods across a wide range of networks and datasets. We shortlist two classical pruning methods (global weight, global gradient) suggested by (Blalock et al., 2020) and three state-of-the-art pruning method (Iterative Magnitude Pruning (IMP) (Frankle & Carbin, 2019), Layer-adaptive Magnitude-based Pruning (LAMP) (Lee et al., 2020)) and Lookahead Pruning (LAP) (Park et al., 2020). The details for each experiment are as follows.

1. Pruning ResNet-20 (He et al., 2016) on CIFAR-10 via global magnitude.
2. Pruning VGG-19 (Simonyan & Zisserman, 2014) on CIFAR-10 via global gradient.
3. Pruning DenseNet-40 (Huang et al., 2017) on the CIFAR-100 dataset using LAMP.
4. Pruning MobileNetV2 (Sandler et al., 2018) on the CIFAR-100 dataset using LAP.
5. Pruning ResNet-50 on ImageNet (i.e., ImageNet-1000) (Deng et al., 2009) using IMP.
6. Pruning Vision Transformer (ViT-B-16) (Dosovitskiy et al., 2020) on CIFAR-10 using IMP.

In each experiment, **to demonstrate the robustness of parameters in the proposed SILO, we compare SILO with a fixed value of ($\mathtt{q} = 1$, β =5) to constant LR and the three shortlisted LR schedules**: (i) LR decay, (ii) cyclical LR and (iii) LR warmup (described in Section 4.1). The implementation details of each LR schedule are summarized in Table 1.

Schedule	**Description** (Iters: Iterations)
LR decay (`a`, `b`)	linearly decay the value of LR from `a` over `b` Iters.
cyclical LR (`a`, `b`, `c`)	linearly vary between `a` and `b` with a step size of `c` Iters.
LR warmup (`a`, `b`, `c`, `d`, `e`)	increase to `a` over `b` Iters, 10x drop at `c`, `d`, `e` Iters.
SILO (ϵ, δ, `b`, `c`, `d`, `e`)	LR warmup (`max_lr`, `b`, `c`, `d`, `e`), where `max_lr` increases from ϵ to $\epsilon + \delta$ during iterative pruning (see equation 3).

Table 1: Descriptions of LR schedule benchmarks and the proposed SILO.

(1) Methodology. We train the network using the training dataset via SGD with momentum = 0.9 and a weight decay of 1e-4 (same as (Renda et al., 2019; Frankle & Carbin, 2019)). Next, we prune the trained network with a pruning rate of 0.2 (i.e., 20% of remaining weights are pruned) in 1 pruning cycle. We repeat 25 pruning cycles in 1 run and use early-stop top-1 test accuracy (i.e., the corresponding test accuracy when early stopping criteria for validation error is met) to evaluate the performance. The results are averaged over 5 runs and the corresponding standard deviation are summarized in Tables 2 - 7, where the results of pruning ResNet-20, VGG-19, DenseNet-40, MobileNetV2, ResNet-50 and Vision Transformer (ViT-B-16) are shown, respectively. **Some additional details (e.g., training epochs, optimizer, batch size, etc) and results for more values of λ are given in Tables 16 - 21 in the Appendix**.

(2) SOTA LR Schedules. To ensure fair comparison against prior SOTA LR schedules, we use LR schedules reported in the literature. Specifically, LR schedules (i.e., LR-warmup) from Table 2 - 6 are from (Frankle & Carbin, 2019), (Frankle et al., 2020), (Zhao et al., 2019), (Chin et al., 2020) and (Renda et al., 2019), respectively. The LR schedule (i.e., cosine decay) in Table 7 is from (Dosovitskiy et al., 2020).

(3) Parameters for other LR schedules. For the other schedules without a single "best" LR in the literature, we tune the value of LR for each of them via a grid search with a range from 1e-4 to 1e-1 using the validation accuracy. Other related parameters (e.g., step size) are also tuned in the same manner. Lastly, we highlight that all LR schedules used, including SILO, are rewound to the initial state at the beginning of each pruning cycle, which is the same as the LR rewinding in (Renda et al., 2019).

(4) Source Code & Devices: We use Tesla V100 devices for our experiments, and the source code is available at `https://github.com/Martin1937/SILO`.

5.2 Performance Comparison

(1) Reproducing SOTA results. By using the implementations reported in the literature, we have correctly reproduced SOTA results. For example, the benchmark results of LR warmup in our Tables 2 - 7 are comparable to Fig. 11 and Fig. 9 of (Blalock et al., 2020), Table 4 in (Liu et al., 2019b), Fig. 3 in (Chin et al., 2020), Fig. 10 in (Frankle et al., 2020), Table 5 in (Dosovitskiy et al., 2020), respectively.

(2) SILO outperforms SOTA results. The key innovation of SILO is that the LR precisely increases as the network is pruned, by increasing `max_lr` in an S-shape as λ decreases. This results in a much higher accuracy than all LR schedule benchmarks studied. For example, in Table 2, the top-1 test accuracy of SILO is 1.8% higher than the best performing schedule (i.e., LR-warmup) at $\lambda = 5.72$. SILO also obtains the best performance when using larger models in Table 3 (i.e., 2.6% higher at $\lambda = 5.72$) and using more difficult datasets in Table 4 (i.e., 4.0% higher at $\lambda = 5.72$).

(3) Performance on ImageNet. In Table 6, we show the performance of SILO using IMP (i.e., the lottery ticket hypothesis pruning method) via ResNet-50 on ImageNet which contains over 1.2 million images from 1000 different classes. We observe that SILO still outperforms the best performing LR schedule benchmark (LR-warmup) by 1.9% at $\lambda = 8.59$. This improvement increases to 3.2% when λ reduces to 5.72.

(4) Performance on SOTA networks (Vision Transformer). Several recent works (Liu et al., 2021; Yuan et al., 2021) demonstrated that transformer based networks tend to provide excellent performance

Original Top-1 Test Accuracy = 91.7% (λ = 100)				
λ	32.8	26.2	8.59	5.72
constant LR	88.1 ± 0.9	87.5 ± 0.7	82.8 ± 0.9	79.1 ± 0.8
LR decay	89.8 ± 0.4	89.0 ± 0.7	83.9 ± 0.6	79.8 ± 0.7
cyclical LR	89.7 ± 0.6	88.2 ± 0.7	84.1 ± 0.8	80.3 ± 0.7
LR-warmup	90.3 ± 0.4	89.8 ± 0.6	85.9 ± 0.9	81.2 ± 1.1
SILO (Ours)	**90.8 ± 0.5**	**90.3 ± 0.4**	**87.5 ± 0.8**	**82.7 ± 1.2**

Table 2: Top-1 test accuracy ± standard deviation of pruning ResNet-20 on CIFAR-10 via global magnitude.

Original Top-1 Test Accuracy = 92.2% (λ = 100)				
λ	32.8	26.2	8.59	5.72
constant LR	88.8 ± 0.6	87.4 ± 0.7	82.2 ± 1.4	73.7 ± 1.3
LR decay	89.4 ± 0.4	88.6 ± 0.5	83.3 ± 0.8	75.4 ± 0.9
cyclical LR	89.8 ± 0.5	89.1 ± 0.6	83.7 ± 1.0	75.7 ± 1.2
LR-warmup	90.2 ± 0.5	89.8 ± 0.8	84.5 ± 0.9	76.5 ± 1.0
SILO (Ours)	**90.6 ± 0.6**	**90.3 ± 0.6**	**86.1 ± 0.8**	**78.5 ± 1.0**

Table 3: Top-1 test accuracy ± standard deviation of pruning VGG-19 on CIFAR-10 using global gradient.

Original Top-1 Test Accuracy = 74.6% (λ = 100)				
λ	32.8	26.2	8.59	5.72
constant LR	70.3 ± 0.8	68.1 ± 0.7	60.8 ± 1.1	59.1 ± 1.2
LR decay	71.2 ± 0.8	69.0 ± 0.6	62.6 ± 1.2	60.3 ± 1.4
cyclical LR	70.9 ± 0.6	69.4 ± 0.6	63.0 ± 1.1	60.8 ± 1.3
LR-warmup	71.5 ± 0.7	69.6 ± 0.8	63.9 ± 1.0	61.2 ± 0.9
SILO (Ours)	**72.4 ± 0.7**	**70.8 ± 0.8**	**65.7 ± 1.2**	**63.7 ± 1.0**

Table 4: Top-1 test accuracy ± standard deviation of pruning DenseNet-40 on CIFAR-100 using LAMP.

Original Top-1 Test Accuracy = 73.7% (λ = 100)				
λ	32.8	26.2	8.59	5.72
constant LR	69.8 ± 1.1	68.2 ± 0.9	63.8 ± 1.1	62.1 ± 1.2
LR decay	70.9 ± 1.0	69.4 ± 0.6	65.1 ± 0.8	64.0 ± 1.1
cyclical LR	71.5 ± 0.7	69.6 ± 0.6	65.3 ± 1.1	64.3 ± 1.2
LR-warmup	72.1 ± 0.8	70.5 ± 0.9	66.2 ± 1.1	64.8 ± 1.5
SILO (Ours)	**72.5 ± 0.6**	**71.0 ± 0.7**	**68.8 ± 0.8**	**66.8 ± 1.4**

Table 5: Top-1 test accuracy ± standard deviation of pruning MobileNetV2 on CIFAR-100 using LAP.

in computer vision tasks (e.g., classification). We now examine the performance of SILO using Vision Transformer (i.e., ViT-B16 with a resolution of 384). We note that the ViT-B16 uses Gaussian Error Linear Units (GELU, GELU(x) = x$\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function) as the activation function. We note that both ReLU and GELU have the unbounded output, suggesting that SILO could be helpful for pruning GELU based models as well.

We repeat the same experiment setup as above and compare the performance of SILO to other LR schedules using ViT-B16 in Table 7. We observe that SILO is able to outperform the standard implementation (cosine decay, i.e., decay the learning rate via the cosine function) by 1.3% at $\lambda = 8.59$ in top-1 test accuracy. This improvement increases to 1.6% when λ reduces to 5.72.

Original Top-1 Test Accuracy = 77.0% (λ = 100)				
λ	32.8	26.2	8.59	5.72
constant LR	74.2 ± 0.8	73.9 ± 0.7	70.5 ± 0.6	69.2 ± 0.9
LR decay	75.6 ± 0.5	75.1 ± 0.5	72.7 ± 0.8	70.5 ± 0.6
cyclical LR	76.5 ± 0.5	75.5 ± 0.6	73.4 ± 0.8	71.2 ± 0.7
LR-warmup	76.6 ± 0.2	75.8 ± 0.3	73.8 ± 0.5	71.5 ± 0.4
SILO (Ours)	**76.8 ± 0.4**	**76.1 ± 0.7**	**75.2 ± 0.8**	**73.8 ± 0.6**

Table 6: Top-1 test accuracy ± standard deviation of pruning ResNet-50 on ImageNet using IMP.

Original Top-1 Test Accuracy = 98.0% (λ = 100)				
λ	32.8	26.2	8.59	5.72
constant LR	96.4 ± 0.5	96.0 ± 0.7	83.0 ± 0.9	80.1 ± 0.8
cosine decay	97.2 ± 0.2	96.5 ± 0.6	84.1 ± 1.0	81.6 ± 1.1
cyclical LR	97.0 ± 0.2	96.5 ± 0.6	83.4 ± 0.6	81.0 ± 1.1
LR-warmup	97.3 ± 0.6	96.8 ± 0.7	84.4 ± 0.8	82.1 ± 0.9
SILO (Ours)	**97.7 ± 0.5**	**97.4 ± 0.6**	**85.5 ± 0.9**	**83.4 ± 0.8**

Table 7: Top-1 test accuracy ± standard deviation of pruning Vision Transformer on CIFAR-10 using IMP.

Percent of Weights Remaining, λ	32.8	26.2	8.59	5.72
LR-warmup (benchmark)	90.3 ± 0.4	89.8 ± 0.6	85.9 ± 0.9	81.2 ± 1.2
SILO (q = 1, $\beta = 5$)	90.8 ± 0.5	90.3 ± 0.4	87.5 ± 0.8	82.7 ± 1.2
SILO (q = 2, $\beta = 5$)	90.5 ± 0.6	90.1 ± 0.2	86.9 ± 0.4	82.4 ± 1.1
SILO (q = 3, $\beta = 4$)	90.9 ± 0.4	90.2 ± 0.5	87.1 ± 0.9	82.2 ± 1.4
SILO (q = 1, $\beta = 3$)	90.5 ± 0.9	90.6 ± 0.8	87.5 ± 0.6	83.1 ± 1.6
SILO (q = 1, $\beta = 7$)	90.4 ± 0.2	89.2 ± 0.7	85.1 ± 1.2	80.8 ± 1.9
SILO (q = 5, $\beta = 5$)	90.2 ± 0.3	89.8 ± 0.5	85.5 ± 1.0	80.5 ± 1.6

Table 8: Performance (Top-1 test accuracy ± standard deviation) of SILO with different values of q and β when pruning ResNet-20 on CIFAR-10 via global magnitude.

Percent of Weights Remaining, λ	32.8	26.2	8.59	5.72
LR-warmup (benchmark)	90.2 ± 0.5	89.8 ± 0.8	84.5 ± 0.9	76.5 ± 1.0
SILO (q = 1, $\beta = 5$)	90.6 ± 0.6	90.3 ± 0.6	86.1 ± 0.8	78.5 ± 1.0
SILO (q = 2, $\beta = 5$)	90.7 ± 0.5	90.0 ± 0.4	86.5 ± 0.9	79.2 ± 1.3
SILO (q = 3, $\beta = 4$)	90.3 ± 0.2	90.4 ± 0.3	85.8 ± 0.6	78.8 ± 0.9
SILO (q = 1, $\beta = 3$)	90.8 ± 0.8	90.1 ± 0.6	85.5 ± 0.9	78.2 ± 1.1
SILO (q = 1, $\beta = 7$)	90.0 ± 0.4	88.5 ± 0.9	82.1 ± 1.4	75.3 ± 1.3
SILO (q = 5, $\beta = 5$)	90.1 ± 0.5	89.1 ± 0.6	83.2 ± 1.1	75.7 ± 1.5

Table 9: Performance (Top-1 test accuracy ± standard deviation) of SILO with different values of q and β when pruning VGG-19 on CIFAR-10 via global gradient.

5.3 Sensitivity of Parameters in SILO

In Section 5.2, we demonstrate the robustness of parameters in SILO by using a fixed value of q = 1, $\beta = 5$ and compare it to benchmark schedulers using different networks, datasets and pruning methods. We now repeat the setup in Tables 2 - 3 and examine the sensitivity of these two parameters in Tables 8 - 9.

Percent of Weights Remaining, λ	100	51.3	32.9	21.1	5.72
Oracle optimized `max_lr` ($\times 10^{-2}$)	4	4.6	9.0	9.8	10.2
Oracle optimized interval ($\times 10^{-2}$)	[3.6, 4.2]	[4.2, 5.4]	[8.0, 9.6]	[9.2, 10.4]	[9.8, 10.6]
SILO's estimated `max_lr` ($\times 10^{-2}$)	4	4.32	9.2	9.9	9.99

Table 10: Comparison between Oracle optimized `max_lr`, Oracle optimized interval (both obtained via grid search) and the value of `max_lr` estimated by SILO when iteratively pruning VGG-19 on CIFAR-10.

Percent of Weights Remaining, λ	100	51.3	41.1	32.9	21.1
Oracle optimized `max_lr` ($\times 10^{-2}$)	3.4	3.8	4.6	5.6	6.2
Oracle optimized interval ($\times 10^{-2}$)	[2.8, 3.6]	[3.4, 4.2]	[3.8, 5.2]	[5.4, 6.6]	[5.4, 6.8]
SILO estimated `max_lr` ($\times 10^{-2}$)	3	3.2	4.7	6.4	6.9

Table 11: Comparison between Oracle tuned `max_lr`, Oracle optimized `max_lr` interval (both obtained via grid search) and `max_lr` estimated by SILO when iteratively pruning ResNet-20 on the CIFAR-10 dataset.

The takeaway message from Tables 8 - 9 is three-fold: (i) The parameters of SILO can provide promising results (i.e., outperform benchmark schedulers) within the suggested value range, i.e., $0 \leq q \leq 3$, $3 \leq \beta \leq 6$ (see first 5 rows in Tables 8 - 9). (ii) When the value of q and β fall outside the suggested range, SILO may fail to outperform benchmarks (see last 2 rows in Tables 8 - 9). (iii) It also suggests that q and β could be tuned within a relatively small range and does not require as much effort as tuning the hyperparameters.

5.4 Comparing SILO to an Oracle

Our new insight suggests that, due to the change in distribution of hidden representations during iterative pruning, LR should be re-tuned at each pruning cycle. SILO provides a method to adjust the `max_lr` in an S-shape, which is backed up by a theoretical result (see Theorem 2). We now further examine the S-shape trajectory of SILO by comparing SILO's estimated `max_lr` to an Oracle, which uses the same LR warmup structure as SILO but exhaustively searches for the optimal value of `max_lr` at each pruning cycle. The Oracle's `max_lr` at the current pruning cycle is chosen by grid search ranging from `1e-4` to `1e-1` and the best performing value (i.e., determined by validation accuracy) is used to train the network. The results of `max_lr` determined this way when iteratively pruning a VGG-19 on CIFAR-10 using the global magnitude are detailed in Table 10 via two metrics:

1. **Oracle optimized `max_lr`**: The value of `max_lr` that provides the best validation accuracy.
2. **Oracle optimized interval**: The range of `max_lr` which provides comparable performance to Oracle optimized `max_lr` (i.e., within 0.5% of the best validation accuracy).

SILO vs Oracle (Performance): In Table 10, the `max_lr` estimated by SILO falls in the Oracle optimized `max_lr` interval at each pruning cycle, suggesting that SILO can precisely adjust `max_lr` to provide competitive performance with Oracle. This further verifies the S-shape trajectory of `max_lr` used in SILO.

SILO vs Oracle (Complexity): The process of finding the Oracle tuned `max_lr` requires a significantly larger computational complexity in tuning due to the grid search. Assume that `max_lr` is searched from a sampling space of $[\theta_1, \cdots, \theta_n]$ for k pruning cycles. Hence, the complexity of the Oracle will be $\mathcal{O}(n^k)$. On the other hand, SILO controls the variation of `max_lr` at each pruning cycle via four parameters: ranges of `max_lr`: $[\epsilon, \epsilon + \delta]$, delay term `q` and S-shape control term β. Similar to the Oracle, both ϵ and δ can be searched from a range of n values. As we have recommended before, `q` and β can be tuned in the range of [0, 3], [3, 6], respectively. As a result, SILO has a complexity of $\mathcal{O}(n^2)$, which is exponentially less complex than the Oracle's complexity, but with competitive performance. Lastly, we highlight that similar performance trends can be observed using ResNet-20 via global magnitude (see Table 11).

Params: 227K; **Train Steps**: 63K Iters; **Batch**: 128; **Pruning Rate**: 0.2					
λ	100	32.9	21.1	5.72	2.03
constant LR	88.4±0.4	84.8±0.6	83.5±0.6	75.5±1.2	67.1±1.7
LR decay	88.6±0.3	87.1±0.7	83.7±0.9	76.1±0.8	66.0±1.3
cyclical LR	88.9±0.3	86.9±0.5	84.1±0.3	77.0±0.9	64.4±1.1
LR-warmup	89.1±0.3	87.2±0.4	84.5±0.6	75.2±1.1	65.1±1.9
SILO	**89.2±0.2**	**87.9±0.3**	**86.3±0.5**	**79.5±1.7**	**71.7±2.3**

Table 12: Performance comparison (averaged top-1 test accuracy ± std over 5 runs) of iteratively pruning ResNet-20 on CIFAR-10 dataset using global magnitude and Adam optimizer (Kingma & Ba, 2014).

Params: 227K; **Train Steps**: 63K Iters; **Batch**: 128; **Pruning Rate**: 0.2					
λ	100	32.9	21.1	5.72	2.03
constant LR	87.9±0.3	83.4±0.4	81.5±0.9	65.5±1.9	55.1±2.3
LR decay	88.4±0.2	84.8±0.6	77.8±0.9	67.1±1.4	58.3±1.6
cyclical LR	88.1±0.3	84.7±0.5	81.9±0.7	67.5±0.9	56.3±1.7
LR-warmup	**88.9±0.2**	85.1±0.5	81.7±0.4	67.3±1.3	57.1±1.4
SILO	88.7±0.3	**86.1±0.4**	**83.1±0.6**	**72.5±1.3**	**63.5±1.9**

Table 13: Performance comparison (top-1 test accuracy ± std over 5 runs) of iteratively pruning ResNet-20 on CIFAR-10 dataset using global magnitude and RMSProp optimizer (Tieleman & Hinton, 2012).

6 Reflections

In summary, SILO is an adaptive LR schedule for network pruning with theoretical justification. SILO outperforms existing benchmarks by 2.1% - 3.2% via classical networks and datasets. For SOTA networks (e.g., Vision Transformer) and large scale datasets (e.g., ImageNet), it leads to an improvement of 3% - 5.6%. More importantly, via the S-shape trajectory, SILO can obtain comparable performance to Oracle with significantly lower complexity. We now conclude the paper by presenting some reflections.

(1) The gain of SILO. As compared to existing LR schedules, the advantage of SILO is three-fold: (i) SILO is specially designed for network pruning which gradually increases the value of LR in an S-shape as the network is gradually pruned. (ii) The S-shape trajectory is theoretically justified and empirically examined by comparing to an Oracle. The results suggest that SILO can obtain a comparable performance to the Oracle with a significantly lower complexity. (iii) As for its performance, SILO outperforms existing LR schedules by 2% - 4%, which is demonstrated via extensive experiments.

(2) Connection to Prior Work. Our work explores the important role of LR in network pruning and provides a new insight – *as the ReLU-based network is iteratively pruned, a larger LR should be used to retrain the pruned network.* This new insight could be used to explain the surprising effect of LR schedules observed in prior works. Specifically, Frankle & Carbin (2019) highlighted that they can only find winning tickets after applying a LR warmup schedule. Using insights from our analysis, we attribute this to LR warmup increasing the LR to a large value (e.g., Frankle & Carbin (2019) increases LR to 1e-1 when training VGG-19) which is better for pruned networks. Similarly, Renda et al. (2019) proposed LR rewinding and demonstrated that it outperforms standard fine-tuning. We attribute this to LR rewinding ensuring that a relatively larger LR (as compared to fine-tuning) is used to better train pruned networks in every pruning cycle.

(3) Performance using Adaptive LR Optimizers. In the main paper, we only evaluate the performance of SILO using SGD. We note that the weight update mechanism is different for other adaptive learning rate optimizers, which may potentially affect the performance of SILO. In Tables 12 - 13, we conduct a similar performance comparison using Adam (Kingma & Ba, 2014) and RMSprop (Tieleman & Hinton, 2012), and SILO still outperforms all LR schedule benchmarks studied.

Percent of Weights Remaining, λ	30.4	24.7	7.29
(i) LR-Warmup	88.5 ± 0.9	87.1 ± 1.2	82.2 ± 1.4
(ii) SILO (q = 1, $\beta = 3$)	89.4 ± 0.8	88.5 ± 1.4	83.9 ± 1.7
(iii) LR-Warmup	87.4 ± 0.7	85.2 ± 0.9	80.8 ± 1.6
(iv) SILO (q = 1, $\beta = 3$)	88.1 ± 0.6	87.1 ± 1.1	82.3 ± 1.4

Table 14: Performance comparison between SILO and LR-warmup when pruning VGG-19 (rows (i) and (ii)) and ResNet-20 (rows (iii) and (iv)) using L1 Norm filter pruning (Li et al., 2017) on CIFAR-10 dataset.

(4) Extension of SILO to Structured Pruning. In Section 3.1, we have examined the distribution of weight gradients using structured pruning and shown that structured pruning also suffer from the issue of decreasing weight gradients (see Fig. 7 in the Appendix for more details). This suggests that SILO could be applicable to structured pruning as well.

In Table 14, we compare the performance of SILO to LR-warmup using L1 Norm filter pruning (Li et al., 2017) on ResNet-20 and VGG-19. We observe that SILO still outperforms the benchmark LR schedule, leading to an improvement of 2.0% (compare SILO at row (ii) to LR-warmup at row (i) when $\lambda = 7.29$). We note that the performance improvement is not as significant as that of working with unstructured pruning. We suspect it is due to that the distribution of weight gradients may change differently when the network is structurely pruned. In such a case, SILO may need to be re-designed for a larger performance gain.

(5) Effect of SILO on Weight Update and Generalization. We also examine the effect of SILO on distribution of weight update (weight gradient $\times$ learning rate) and the experimental results demonstrate that, with SILO, the distribution of weight update of pruned networks becomes less centralized, suggesting that SILO helps to mitigate the issue of decreasing weight gradients. As a result, the pruned network is better trained with SILO, leading to a much better generalization performance. We refer the interested reader to Figs. 8 - 10 in the Appendix.

(6) Future Research. (i) We demonstrate the performance of SILO on ReLU-based networks (ResNet, VGG) and GELU-based networks (Vision Transformer). We note that similar LR schedules could be used for networks with other activation functions (e.g., PReLU) and we plan to explore this in future research. (ii) Moreover, the main motivation for SILO is that the distribution of weight gradients tends to become narrower after pruning. An approach to automatically determine the value of `max_lr` from the distribution of weight gradients is definitely worth deeper thought. (iv) As mentioned above, when it comes to the case of structured pruning, the improvement of SILO is not as large as that of working with unstructured pruning. This could be due to that the distribution of weight gradients may change differently when the network is structurely pruned. In such a case, SILO may need to be re-designed for a larger performance gain.

Acknowledgements

This research is supported by A*STAR, CISCO Systems (USA) Pte. Ltd and the National University of Singapore under its Cisco-NUS Accelerated Digital Economy Corporate Laboratory (Award I21001E0002). Additionally, we would like to thank the members of the Kent-Ridge AI research group at the National University of Singapore for helpful feedback and interesting discussions on this work.

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 97, pp. 242–252, 2019.

Wangpeng An et al. Exponential decay sine wave learning rate for fast deep neural network training. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pp. 1–4. IEEE, 2017.

Larry C. Andrews. *Special functions of mathematics for engineers*, volume 49. Spie Press, 1998.

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 80, pp. 244–253, 2018.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the trade*, pp. 437–478. Springer, 2012.

Davis Blalock et al. What is the state of neural network pruning? In *Proceedings of the Machine Learning and Systems (MLSys)*, 2020.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1518–1528, 2020.

Jia Deng et al. Imagenet: A large-scale hierarchical image database. In *2009 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

Misha Denil et al. Predicting parameters in deep learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2148–2156, 2013.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

Jonathan Frankle et al. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 3259–3269, 2020.

Venkata Gandikota et al. vqsgd: Vector quantized stochastic gradient descent. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 2197–2205, 2021.

Itai Gat, Yossi Adi, Alex Schwing, and Tamir Hazan. On the importance of gradient norm in pac-bayesian bounds. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:16068–16081, 2022.

Rong Ge et al. The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares. *arXiv preprint arXiv:1904.12838*, 2019.

Aidan Good and et al. Recall distortion in neural network pruning and the undecayed pruning algorithm. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:32762–32776, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han et al. Learning both weights and connections for efficient neural network. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1135–1143, 2015b.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 164–171, 1993.

Kaiming He et al. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Tong He et al. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 558–567, 2019.

Yang He et al. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Hengyuan Hu et al. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Gao Huang et al. Densely connected convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 37, pp. 448–456, 2015.

Arnulf Jentzen et al. Strong error analysis for stochastic gradient descent optimization algorithms. *IMA Journal of Numerical Analysis*, 41(1):455–492, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 2015.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 598–605, 1990.

Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. In *International Conference on Learning Representations (ICLR)*, 2020.

Namhoon Lee et al. SNIP: Single-shot network pruning based on connection sensitivity. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Hao Li et al. Pruning filters for efficient convnets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Yawei Li and et al. Revisiting random channel pruning for neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 191–201, 2022.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1529–1538, 2020.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019a.

Linfeng Liu, Xu Han, Dawei Zhou, and Liping Liu. Towards accurate subgraph similarity computation via neural graph pruning. *Transactions on Machine Learning Research (TMLR)*, 2022. URL `https://openreview.net/forum?id=CfzIsWWBlo`.

Shiwei Liu and et al. Sparse training via boosting pruning plasticity with neuroregeneration. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 9908–9922, 2021.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10012–10022, 2021.

Zhuang Liu et al. Rethinking the value of network pruning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019b.

Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1458–1467, 2020.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 5058–5066, 2017.

Eran Malach et al. Proving the lottery ticket hypothesis: Pruning is all you need. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 6682–6691, 2020.

Rahul Mehta. Sparse transfer learning via winning lottery tickets. In *Proceedings of the Advances in Neural Information Processing Systems Workshop on Learning Transferable Skills*, 2019.

Pavlo Molchanov et al. Pruning convolutional neural networks for resource efficient inference. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Pavlo Molchanov et al. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11264–11272, 2019.

Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. Lookahead: A far-sighted alternative of magnitude-based pruning. 2020.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations (ICLR)*, 2019.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

David W Scott. Sturges' rule. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(3):303–306, 2009.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472. IEEE, 2017.

Chong Min John Tan and Mehul Motani. Dropnet: Reducing neural network complexity via iterative pruning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 9356–9366. PMLR, 2020.

Lucas Theis et al. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Artem Vysogorets and Julia Kempe. Connectivity matters: Neural network pruning through the lens of effective sparsity. *Journal of Machine Learning Research*, 24(99):1–23, 2023.

Huan Wang and et al. Recent advances on neural network pruning at initialization. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, Vienna, Austria*, pp. 23–29, 2022.

Tianzhe Wang et al. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020a.

Yulong Wang et al. Dynamic network pruning with interpretable layerwise channel selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6299–6306, 2020b.

Yulong Wang et al. Pruning from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 12273–12280, 2020c.

Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14913–14922, June 2021.

Xia Xiao, Zigeng Wang, and Sanguthevar Rajasekaran. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 13681–13691, 2019.

He Yang et al. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4335–4344, 2019.

Mao Ye et al. Good subnetworks provably exist: Pruning via greedy forward selection. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 10820–10830. PMLR, 2020.

Haoran You and et al. Max-affine spline insights into deep network pruning. *Transactions on Machine Learning Research (TMLR)*, 2022. ISSN 2835-8856.

Kaichao You et al. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*, 2019.

Hao nan Yu et al. Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Ruichi Yu et al. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9194–9203, 2018.

Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. Volo: Vision outlooker for visual recognition. *arXiv preprint arXiv:2106.13112*, 2021.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Yuyao Zhang and Nikolaos M Freris. Adaptive filter pruning via sensitivity feedback. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2023.

Chenglong Zhao et al. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2780–2789, 2019.

Hattie Zhou et al. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

A Appendix: Proofs of Theoretical Results

In this section, we provide the proofs of the theoretical results (Theorem 1 and Theorem 2) in the paper. Furthermore, we also provide three new theoretical results (Propositions 1 and 2 and Corollary 1), which extend our results to a more general case of arbitrary network depth D and to the case of finite hidden neurons.

A.1 Proof of Theorems 1 and 2, Corollary 1, Propositions 1 and 2

Theorem 1. Consider a single-hidden layer ReLU activated network represented as $X \xrightarrow{W_1} H \xrightarrow{W_2} Y$, where $X \in \mathbb{R}^d$ is the input, $H = \{H_1(X), H_2(X), ..., H_N(X)\}$ is of infinite width ($N = \infty$), and Y is the network output. W_1 and W_2 represent network parameters (weights, biases). Furthermore, let $X \sim \mathcal{N}(0, \sigma_x^2 I)$ and $W_1 \sim \mathcal{N}(0, \sigma_w^2 I)$, where I is the identity matrix and σ_x, σ_w are scalars. Now, let us consider an iterative pruning method, where in each iteration a fraction $0 \leq p \leq 1$ of the smallest magnitude weights are pruned (layer-wise pruning). Then, after k iterations of pruning, it holds that

$$4E_{AA}(H) \geq \sigma_W^2 + d\sigma_X^2\sigma_W^2 \left((1-p)^k + \sqrt{\tfrac{4}{\pi}}\,\text{erf}^{-1}\left(1-(1-p)^k\right) e^{-\left(\text{erf}^{-1}\left(1-(1-p)^k\right)\right)^2}\right) \tag{4}$$

where erf^{-1}(.) is the inverse error function (Andrews, 1998).

Proof. Let us separate the weights and biases for the first layer into the matrix $W_1 \in \mathbb{R}^{d\times N}$ and the bias vector $b \in \mathbb{R}^N$. We can thus compute the hidden layer activations as,

$$H_j(X) = max\left(0, b_j + \sum_{i=1}^{d} W_1^{ij} X_i\right) \tag{5}$$

We can express $X = [X_1, X_2, .., X_d]$, where $X_i \sim \mathcal{N}(0, \sigma_X^2)$ are i.i.d random variables. Let us denote $Z_j(X) = b_j + \sum_{i=1}^d W_1^{ij} X_i$. We have that $\mathbb{E}_X[Z_j(X)] = \sum_{i=1}^d \mathbb{E}_X[W_1^{ij} X_i] + b_j = b_j \sum_{i=1}^d W_1^{ij} \mathbb{E}_X[X_i] = b_j$. We can similarly compute the variance of $Z_j(X)$, $\mathbb{E}_X[(Z_j(X) - b_j)^2] = \sigma_X^2 \sum_{i=1}^d \left(W_1^{ij}\right)^2$, which follows from the fact that $X_1, X_2, ...X_d$ are independent. Note that as the sum of independent Gaussian distributed variables is Gaussian, we have that $Z_j(X) \sim \mathcal{N}\left(b_j, \sigma_X^2 \sum_{i=1}^d \left(W_1^{ij}\right)^2\right)$.

With this observation, we first show that for the random variable $H_j(X) = max\left(0, b_j + \sum_{i=1}^d W_1^{ij} X_i\right)$, for $b_j \geq 0$, $\mathbb{E}_X[H_j(X)^2] \geq \mathbb{E}_X[Z_j(X)^2]/2 = \left(b_j^2 + \mathbb{E}_X\left[\left(\sum_{i=1}^d W_1^{ij} X_i\right)^2\right]\right)/2$. To show this, we first note that for all $x < 0$, $P(H_j(X) = x) = 0$, $P(H_j(X) = 0) = \int_{-\infty}^0 P(Z_j(x))dx$, and for $x > 0$, $P(H_j(X) = x) = P(Z_j(X) = x)$. Note that for $b_j \geq 0$, $\int_{-\infty}^0 P(Z_j(x))dx \leq \frac{1}{2}$. Let us define $c = 1 - \int_{-\infty}^0 P(Z_j(x))dx$. Thus, it holds that $c \geq \frac{1}{2}$. As $Z_j(X) \sim \mathcal{N}\left(b_j, \sigma_X^2 \sum_{i=1}^d \left(W_1^{ij}\right)^2\right)$, we have that

$$\mathbb{E}_X[H_j(X)^2 | b_j \geq 0] = \int_0^\infty x^2 P(H_j(X) = x)dx = c\int_{0^+}^\infty x^2 \frac{P(H_j(X) = x)}{c}dx \tag{6}$$

$$= c\int_{0^+}^\infty x^2 \frac{P(Z_j(X) = x)}{c}dx = c\int_{-\infty}^\infty x^2 P_{trunc}(Z_j(X) = x)dx, \tag{7}$$

where $P_{trunc}(Z_j(X))$ is a truncated version of the normal distribution $\mathcal{N}\left(b_j, \sigma_X^2 \sum_{i=1}^d \left(W_1^{ij}\right)^2\right)$, where all probability values for all $Z_j(X) \leq 0$ are now zero. For simplicity of notation, we denote $\sigma_j^2 =$

$\sigma_X^2 \sum_{i=1}^d \left(W_1^{ij}\right)^2$. Let $\mu_{trunc} = \int_{-\infty}^{\infty} xP_{trunc}(Z_j(X) = x)dx$. Also, let $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. In what follows we use the expression for the mean and variance of a truncated normal distribution,

$$\begin{aligned}
\int_{-\infty}^{\infty} x^2 P_{trunc}(Z_j(X) = x)dx &= \int_{-\infty}^{\infty} (x - \mu_{trunc})^2 P_{trunc}(Z_j(X) = x)dx + (\mu_{trunc})^2 \\
&= \sigma_j^2 \left(1 - \frac{b_j \phi\left(\frac{-b_j}{\sigma_j}\right)}{c\sigma_j} - \frac{\phi\left(\frac{-b_j}{\sigma_j}\right)^2}{c^2}\right) + \left(b_j + \frac{\sigma_j \phi\left(\frac{-b_j}{\sigma_j}\right)}{c}\right)^2 \\
&= \sigma_j^2 + b_j^2 + \frac{b_j \sigma_j \phi\left(\frac{-b_j}{\sigma_j}\right)}{c} && (8) \\
&\geq \sigma_j^2 + b_j^2 = \mathbb{E}_X[Z_j(X)^2] && (9)
\end{aligned}$$

where the last step follows from the fact that $b_j \geq 0$. Combining this result with equation 7, we have that

$$\mathbb{E}_X[H_j(X)^2 | b_j \geq 0] = c \int_{-\infty}^{\infty} x^2 P_{trunc}(Z_j(X) = x)dx \geq c\mathbb{E}_X[Z_j(X)^2] \geq \frac{1}{2}\mathbb{E}_X[Z_j(X)^2], \tag{10}$$

Next, we have that $\mathbb{E}_X[H_j(X)^2 | b_j < 0] \geq 0$. As b_j is distributed as $\mathcal{N}(0, \sigma_W)$ as well, we have $(P(b_j \leq 0) = P(b_j > 0) = 0.5)$. First, from (10), note that we can write $\mathbb{E}_X[H_j(X)^2] = P(b_j \leq 0)\mathbb{E}_X[H_j(X)^2 | b_j \leq 0] + P(b_j > 0)\mathbb{E}_X[H_j(X)^2 | b_j > 0]$. Then, with $P(b_j < 0) = P(b_j \geq 0) = 0.5$, $Z_j(X) \sim \mathcal{N}\left(b_j, \sigma_X^2 \sum_{i=1}^d \left(W_1^{ij}\right)^2\right)$ (see just before (6)), and given the result in (10), we can write, $\mathbb{E}_X[H_j(X)^2] = P(b_j < 0)\mathbb{E}_X[H_j(X)^2 | b_j < 0] + P(b_j \geq 0)\mathbb{E}_X[H_j(X)^2 | b_j \geq 0] \geq \frac{1}{2} \times \frac{1}{2}\mathbb{E}_X[Z_j(X)^2] + 0 = \frac{1}{4}\left(b_j^2 + \sigma_X^2 \sum_{i=1}^d \left(W_1^{ij}\right)^2\right)$. We then have,

$$\frac{1}{N}\sum_{j=1}^N \mathbb{E}_X[H_j(X)^2] \geq \frac{1}{4}\left(\mathbb{E}_j[b_j^2] + \sigma_X^2 \frac{1}{N}\sum_{j=1}^N \sum_{i=1}^d \left(W_1^{ij}\right)^2\right), \tag{11}$$

and as $N \to \infty$ this yields

$$\frac{1}{N}\sum_{j=1}^N 4\mathbb{E}_X[H_j(X)^2] \geq \sigma_W^2 + d\sigma_X^2 \mathbb{E}_{i,j}\left[\left(W_1^{ij}\right)^2\right]. \tag{12}$$

Now, originally, the unpruned weights in W_1 follow the given $\mathcal{N}(0, \sigma_W^2 I)$ distribution. However, after k cycles of pruning, the smallest $1 - (1-p)^k$ proportion of weights in W_1 get removed. Thus, the distribution $P(W_1)$ changes such that, for some appropriate β (which depends on k), all $P(-\beta \leq W_1 \leq \beta) = 0$ except for $P(W_1 = 0)$, which will follow $P(W_1 = 0) = 1 - (1-p)^k$. Let us denote this modified distribution of weights via $\mathcal{N}_{p,k}(0, \sigma_W^2)$. Let us denote $P(W) = \mathcal{N}_{p,k}(0, \sigma_W^2)$ Note that as the hidden layer has infinite nodes $(N = \infty)$ we can write

$$\begin{aligned}
\mathbb{E}_{i,j}\left[\left(W_1^{ij}\right)^2\right] &= \mathbb{E}_{W \sim \mathcal{N}_{p,k}(0,\sigma_W^2)}\left[W^2\right] && (13) \\
&= \int_{-\infty}^{\infty} W^2 P(W)dW && (14) \\
&= \int_{-\infty}^{-\beta} W^2 P(W)dW + \int_{-\beta}^{\beta} W^2 P(W)dW + \int_{\beta}^{\infty} W^2 P(W)dW && (15) \\
&= \int_{-\infty}^{-\beta} W^2 P(W)dW + \int_{\beta}^{\infty} W^2 P(W)dW && (16) \\
&= \int_{-\infty}^{\infty} W^2 P'(W)dW, && (17)
\end{aligned}$$

where $P'(W) = P(W)\mathbb{1}_{[-\beta,\beta]}(W)$. In order to find $\int_{-\infty}^{\infty} W^2 P'(W)dW$, for $w \sim \mathcal{N}(0,\sigma_W^2)$, we denote $Q(W) = \mathcal{N}(0,\sigma_W^2)$, and we can write

$$\begin{aligned}\mathbb{E}_{W\sim\mathcal{N}(0,\sigma_W^2)}\left[W^2\right] =& \sigma_W^2 = \int_{-\infty}^{\infty} W^2 Q(W)dW & (18)\\ =& \int_{-\beta}^{\beta} W^2 Q(W)dW + \int_{-\infty}^{\infty} W^2 P'(W)dW & (19)\\ =& \tau\int_{-\beta}^{\beta} W^2 \frac{Q(W)}{\tau}dW + \int_{-\infty}^{\infty} W^2 P'(W)dW. & (20)\\ =& \tau\int_{-\infty}^{\infty} W^2 Q'(W)dW + \int_{-\infty}^{\infty} W^2 P'(W)dW. & (21)\end{aligned}$$

Here, τ is computed such that $\int_{-\beta}^{\beta}\frac{Q(W)}{\tau} = 1$, and $Q'(W) = Q(W)(1-\mathbb{1}_{[-\beta,\beta]}(W))/\tau$. Note that here, $\tau = 1-(1-p)^k$. Furthermore, we can see that $Q'(W)$ represents the truncated normal distribution, which has a variance of $\sigma_W^2\left(1-\frac{2\beta}{\sqrt{2\pi}\sigma_W\tau}e^{-\frac{\beta^2}{2\sigma_W^2}}\right)$. We also note that as $\tau = \int_{-\beta}^{\beta} Q(W)$, and as $Q(W)$ is Gaussian, we can write, $\tau = erf(\frac{\beta}{\sqrt{2}\sigma_W})$, which also implies $\frac{\beta}{\sqrt{2}\sigma_W} = \text{erf}^{-1}(\tau)$. Thus, we have

$$\begin{aligned}\int_{-\infty}^{\infty} W^2 P'(W)dW =& \sigma_W^2 - \tau\int_{-\infty}^{\infty} W^2 Q'(W)dW & (22)\\ =& \sigma_W^2 - \sigma_W^2\left(\tau - \frac{2\beta}{\sqrt{2\pi}\sigma_W}e^{-\frac{\beta^2}{2\sigma_W^2}}\right) & (23)\\ =& \sigma_W^2\left(1-\tau+\frac{2\beta}{\sqrt{2\pi}\sigma_W}e^{-\frac{\beta^2}{2\sigma_W^2}}\right) & (24)\\ =& \sigma_W^2\left((1-p)^k + \frac{2\,\text{erf}^{-1}(1-(1-p)^k)}{\sqrt{\pi}}e^{-\left(\text{erf}^{-1}(1-(1-p)^k)\right)^2}\right). & (25)\end{aligned}$$

Combined with equation 12 and equation 17, we obtain,

$$\begin{aligned}\frac{1}{N}\sum_{j=1}^{N} 4\mathbb{E}_X[H_j(X)^2] \geq& \sigma_W^2 + d\sigma_X^2\mathbb{E}_{i,j}\left[\left(W_1^{ij}\right)^2\right] & (26)\\ =& \sigma_W^2 + d\sigma_X^2\sigma_W^2\left((1-p)^k + \sqrt{\frac{4}{\pi}}\,\text{erf}^{-1}(1-(1-p)^k)e^{-\left(\text{erf}^{-1}(1-(1-p)^k)\right)^2}\right), & (27)\end{aligned}$$

which yields the result, as $E_{AA}(H) = \mathbb{E}_X\left[\frac{1}{N}\sum_{j=1}^{N} H_j(X)^2\right] = \frac{1}{N}\sum_{j=1}^{N}\mathbb{E}_X[H_j(X)^2]$. □

Extension of Theorem 1 to the arbitrary depth case:

Proposition 1. We consider the same setting as in Theorem 1, except for the fact that we consider neural networks of arbitrary depth D. Furthermore, let each hidden layer contain M neurons. We specify the distributions of the weights for each layer as follows: for the first layer we have $W_1 \sim \mathcal{N}(0,\sigma_w^2 I)$, and for all subsequent layers ($l > 1$), we have $W_l \sim \mathcal{N}(0, \frac{1}{\sqrt{M}}\sigma_w^2 I)$. Furthermore, we consider the case where all the first $D-1$ layers do not have biases associated with the weights, but only the D^{th} layer has biases, which has the same distribution as weights, like before. Now, same as in Theorem 1, we consider an iterative pruning method, where in each iteration a fraction $0 \leq p \leq 1$ of the smallest magnitude weights are pruned (layer-wise pruning). Let H_D be the hidden layer output at a depth of D. Then, after k iterations of pruning, in the limiting case of $M \to \infty$ it holds that

$$4E_{AA}(H_D) \geq \sigma_W^2 + \tfrac{1}{2^{D-1}}d\sigma_X^2\sigma_W^{2D}\left((1-p)^k + \sqrt{\tfrac{4}{\pi}}\,\text{erf}^{-1}\left(1-(1-p)^k\right)e^{-\left(\text{erf}^{-1}\left(1-(1-p)^k\right)\right)^2}\right) \quad (28)$$

where $\text{erf}^{-1}(.)$ is the inverse error function (Andrews, 1998).

Proof. To prove this result, we first note that, as we are considering the limiting case of $M \to \infty$, the function at any hidden neuron H_i at depth D can always be represented as follows

$$H_j(X) = max\left(0, b_j + \sum_{i=1}^{d} W_{eff}^{ij} X_i\right), \tag{29}$$

where W_{eff}^{ij} represents the *effective* weight random variables from the first $D-1$ layers of the network which is associated with X_i. Note that the non-linearties associated with ReLU activations in the network will be subsumed inside of W_{eff}^{ij} using other random variables which are probabilistically 0 or 1. We elaborate on this later.

We note that due to $M \to \infty$, we can assume W_{eff} to be normally distributed. Furthermore, due to symmetry of distribution and computation, it is clear that for all i and j, W_{eff}^{ij} will have the same distribution parameters, which we denote by $\mathcal{N}\left(\mu_{eff}, \sigma_{eff}^2\right)$. Furthermore, as there are no biases associated with the first $D-1$ layers of computation, we have that $\mu_{eff} = 0$.

To estimate σ_{eff}^2, we first show the estimation of σ_{eff}^2 at depth $D = 2$. For $D = 2$, note that the effective weight that is tied to the input X_1 ($i = 1$) and the output hidden node $j = 1$, can be written as follows:

$$W_{eff}^{11} = \sum_{k=1}^{M} W_1^{1k} W_2^{k1} \delta_k, \tag{30}$$

where $\delta_1, \delta_2, ...\delta_M$ are random variables which are associated with the ReLU non-linearity at the output of the first hidden layer. Although $\delta_1, \delta_2, ...\delta_M$ are dependent on the output at the hidden nodes of the first hidden layer, we note that w.r.t X_1 and the weights W_1^{1k} and W_2^{k1} themselves, they are independent (individually). Note that this is because δ_i is effectively related to the sign of the i^{th} hidden neuron output at the first layer (before the non-linearity), and thus with only X_1 known and with only W_1^{1k} known, its distribution is unchanged in each case. Thus, $\delta_1, ..\delta_M$ are separately independent of X_1, W_1^{1k} and W_2^{k1}. Furthermore, as $M \to \infty$, we can therefore consider $\delta_1, \delta_2, ...\delta_M$ to be independent random variables distributed as $P(\delta_i = 0) = P(\delta_i = 1) = 0.5$. This follows from the fact that X_i and W_1 both are normally distributed with zero-mean. With this, we can estimate σ_{eff}^2 for $D = 2$ as follows:

$$\sigma_{eff}^2 = \mathbb{E}\left[(\sum_{k=1}^{M} W_1^{1k} W_2^{k1} \delta_k)^2\right] = \mathbb{E}\left[\sum_{k=1}^{M} (W_1^{1k} W_2^{k1} \delta_k)^2\right] \tag{31}$$

$$= \sum_{k=1}^{M} \mathbb{E}\left[(W_1^{1k} W_2^{k1} \delta_k)^2\right] = \sum_{k=1}^{M} \frac{1}{2} \sigma_W^2 \frac{\sigma_W^2}{M} = \frac{1}{2} \sigma_W^4 \tag{32}$$

Similarly, one can easily generalize the above result to the general case of depth D as shown below. For the general case of depth D, we will have that

$$\sigma_{eff}^2 = \left(\frac{1}{2}\right)^{D-1} \sigma_W^{2D} \tag{33}$$

Note that with this, we can indeed directly apply the result in Theorem 1, considering the new values of the energy of the effective weights, to obtain:

$$4E_{AA}(H_D) \geq \sigma_W^2 + \tfrac{1}{2^{D-1}} d\sigma_X^2 \sigma_W^{2D} \left((1-p)^k + \sqrt{\tfrac{4}{\pi}}\, \text{erf}^{-1}\left(1-(1-p)^k\right) e^{-\left(\text{erf}^{-1}\left(1-(1-p)^k\right)\right)^2}\right) \tag{34}$$

This proves our intended result. $\square$

Extension of Theorem 1 to the finite hidden neuron case:

Proposition 2. In the setting of Theorem 1, we consider a finite number of hidden neurons in H, thus we have that $N < \infty$. Furthermore, given initialized network weights and biases of the first layer in W_1, let w_{max} denote the maximum valued weight/bias. Let $E(p,k) = (1-(1-p)^k)$. Then, after k iterations of pruning, it holds that

$$Pr\left(4E_{AA}(H) \geq \sigma_W^2 + d\sigma_X^2\sigma_W^2\left(1 - E(p,k) + \sqrt{\tfrac{4}{\pi}}\,\text{erf}^{-1}\left(E(p,k)\right)e^{-\left(\text{erf}^{-1}E(p,k)\right)^2}\right) - \epsilon\right) > 1 - e^{-\frac{2m\epsilon^2}{w_{max}^4(1+d\sigma_x^2)^2}} \tag{35}$$

where $\text{erf}^{-1}(.)$ is the inverse error function (Andrews, 1998).

Proof. To prove this result, we first note from Theorem 1's proof that $\mathbb{E}_X[H_j(X)^2] \geq \frac{1}{4}\mathbb{E}_X[Z_j(X)^2]$, which yields

$$\mathbb{E}_X[H_j(X)^2] \geq \frac{1}{4}\left(b_j^2 + \sigma_X^2\sum_{i=1}^{d}\left(W_1^{ij}\right)^2\right). \tag{36}$$

Let $\beta_j = \mathbb{E}_X[Z_j(X)^2] = \left(b_j^2 + \sigma_X^2\sum_{i=1}^{d}\left(W_1^{ij}\right)^2\right)$. We note that $0 \leq \beta_j \leq w_{max}^2 + d\sigma_x^2 w_{max}^2$. This allows us to apply Hoeffding's inequality, to yield:

$$Pr\left(\frac{1}{N}\sum_{j=1}^{N}[\beta_j] > \mathbb{E}_j[\beta_j] - \epsilon\right) \geq 1 - e^{-\frac{2m\epsilon^2}{w_{max}^4(1+d\sigma_x^2)^2}} \tag{37}$$

From Theorem 1's proof we have that

$$\mathbb{E}_j[\beta_j] = \sigma_W^2 + d\sigma_X^2\sigma_W^2\left((1-p)^k + \sqrt{\frac{4}{\pi}}\,\text{erf}^{-1}(1-(1-p)^k)e^{-\left(\text{erf}^{-1}(1-(1-p)^k)\right)^2}\right). \tag{38}$$

This, coupled with the fact that $\mathbb{E}_X[H_j(X)^2] \geq \frac{1}{4}\mathbb{E}_X[Z_j(X)^2]$, yields the result. □

Theorem 2. In the setting of Theorem 1, we consider a single epoch of weight update for the network across a training dataset $S = \{(X_1,Y_1),..,(X_n,Y_n)\}$ using the cross-entropy loss, where $Y_i \in \{0,1\}$. Let α denote the learning rate. Let us denote the R.H.S of equation 1 by $C(\sigma_X,\sigma_W,p,k)$. Let the final layer weights before and after one training epoch be W_2 and W_2' respectively. We have,

$$\mathop{\mathbb{E}}_{W_2\sim\mathcal{N}_k(0,\sigma_W^2 I)}\left[E_{WG}\left(W_2,W_2'\right)\right] \geq \alpha^2\gamma C(\sigma_X,\sigma_W,p,k), \tag{39}$$

for some constant γ, where $\mathcal{N}_k(0,\sigma_w^2 I)$ represents the distribution of W_2 after being initialized as the Gaussian $\mathcal{N}(0,\sigma_w^2 I)$ and pruned for k iterations.

Additional Motivation for the S-shape. Theorem 2 implies that when $C(\sigma_X,\sigma_W,p,k)$ decreases during iterative pruning, the learning rate α has to increase proportionally so that the lower bound for the *average gradient norm* (E_{WG}) remains constant for all pruning cycles. We plot the adapted learning rate α (i.e., $\alpha \propto \sqrt{K/\gamma C(\sigma_X,\sigma_W,p,k)}$) with pruning rate $p = 0.2$ and pruning cycles $k = 25$, where $K = \mathbb{E}_{W_2\sim\mathcal{N}_k(0,\sigma_W^2 I)}[E_{WG}(W_2,W_2')]$. We find out that the adapted learning rate α resembles a S-shape trajectory, motivating the proposed SILO.

Proof. For the k^{th} sample, let (a_0^k, a_1^k) denote the network output probabilities for the two classes. Thus, we have $a_0^k + a_1^k = 1$ for all k. Furthermore, for the k^{th} sample, let (y_0^k, y_1^k) represent the one-hot label output, which will depend on the true label Y_k. For the k^{th} sample, let (z_0^k, z_1^k) denote the network output logits, from which the output probabilities are computed using the softmax operator. Let W_2^{ij} represent the weight that connects the i^{th} hidden node H_i to the j^{th} output node. Let $L(X_k, Y_k) = -\sum_{l=1}^{2} y_l^k \ln a_l^k$ be the cross-entropy loss for the k^{th} sample. Lastly, let h_i^k represent the output of the i^{th} node in H, for the k^{th} sample. Using backpropagation, the weight update for W_2^{ij}, from a single example X_k, can be written as,

$$W_2^{ij} = W_2^{ij} - \alpha \frac{\partial L(X_k, Y_k)}{\partial W_2^{ij}} \tag{40}$$

$$= W_2^{ij} - \alpha \sum_{l=1}^{2} \frac{\partial L(X_k, Y_k)}{\partial a_l^k} \frac{\partial a_l^k}{\partial z_j^k} \frac{\partial z_j^k}{\partial W_2^{ij}} \tag{41}$$

$$\tag{42}$$

It can be shown that $\frac{\partial L(X_k,Y_k)}{\partial a_l^k} = -\frac{y_l^k}{a_l^k}$, and $\frac{\partial a_l^k}{\partial z_l^k} = a_l^k(1 - a_l^k)$. For $l \neq j$, we can show that $\frac{\partial a_l^k}{\partial z_j^k} = -a_l^k a_j^k$. Furthermore, we also have that $\frac{\partial z_j^k}{\partial W_2^{ij}} = h_i^k$. Combining these results eventually yields,

$$W_2^{ij} = W_2^{ij} - \alpha h_i^k \left(a_j^k - y_j^k\right). \tag{43}$$

Note that this is the update for a single example. Iterating through the entire dataset, the final value of the updated W_2^{ij} can be written as,

$$W_2^{ij} = W_2^{ij} - \alpha \sum_{k=1}^{n} h_i^k \left(a_j^k - y_j^k\right). \tag{44}$$

Using the above we can write the total difference of squares between all weights, $\mathbb{E}[\|W_2 - W_2'\|^2]$, as

$$\mathbb{E}[\|W_2 - W_2'\|^2] = \alpha^2 \frac{1}{N} \sum_{i=1}^{N} \sum_{l=1}^{2} \left(\sum_{k=1}^{n} h_i^k \left(a_l^k - y_l^k\right) \right)^2 \tag{45}$$

Note that here $N = \infty$, as mentioned in Theorem 1. The above expression can be split into two terms as follows

$$\mathbb{E}[\|W_2 - W_2'\|^2] = \alpha^2 \frac{1}{N} \sum_{i=1}^{N} \sum_{l=1}^{2} \left(\sum_{k=1}^{n} h_i^k \left(a_l^k - y_l^k\right) \right)^2 \tag{46}$$

$$= \alpha^2 \sum_{i=1}^{N} \sum_{l=1}^{2} \sum_{k=1}^{n} (h_i^k)^2 \left(a_l^k - y_l^k\right)^2 + \tag{47}$$

$$\alpha^2 \frac{1}{N} \sum_{i=1}^{N} \sum_{l=1}^{2} \sum_{k_1=1}^{n} \sum_{k_2=1}^{n} h_i^{k_1} h_i^{k_2} \left(a_l^{k_1} - y_l^{k_1}\right) \left(a_l^{k_2} - y_l^{k_2}\right) \tag{48}$$

We analyze each term separately as follows, starting with the first expression. In what follows, we incorporate the expectation over $\mathbb{E}_{W_2 \sim \mathcal{N}_k(0, \sigma_W^2 I)}$ into the two expressions. For simplicity of notation, the expectation $\mathbb{E}_{W_2 \sim \mathcal{N}_k(0, \sigma_W^2 I)}$ will be written simply as $\mathbb{E}_{W_2}$.

$$\mathbb{E}_{W_2} \left[\frac{1}{N} \sum_{i=1}^{N} \sum_{l=1}^{2} \sum_{k=1}^{n} (h_i^k)^2 \left(a_l^k - y_l^k\right)^2 \right] = 2n \times \mathbb{E}_{W_2, i, l, k} \left[(h_i^k)^2 \left(a_l^k - y_l^k\right)^2 \right], \tag{49}$$

Now, let us define two random variables R_1 and R_2, such that $R_1 = (h_i^k)^2$ and $R_2 = \left(a_l^k - y_l^k\right)^2$ where $0 \leq i \leq N$, $0 \leq k \leq n$ and $1 \leq l \leq 2$ are random variables all drawn uniformly within their corresponding range. We note that $\mathbb{E}_{W_2,i,l,k} \left[(h_i^k)^2 \left(a_l^k - y_l^k\right)^2 \right] = \mathbb{E}_{W_2,i,l,k} \left[R_1 R_2\right]$. Note that the random variable h_i^k is independent of a_l^k, as h_i^k does not yield any information about W_2, and as a_l^k is a function of H^k and W_2, this implies $P(a_l^k|h_i^k) = P(a_l^k)$. This is mainly because W_2 is a random variable as well. As such, knowledge of h_i^k will not yield any additional information about α_l^k, because (a) the weights that map the entire H^k to the output α_l^k are an independent Gaussian distribution as described in (2) and (b) h_i^k is just one of infinite other hidden layer outputs each of which affect α_l^k as well given a fixed W_2. The independence of h_i^k and y_l^k follows in the same manner. Thus, as $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ for independent X and Y, we have

$$\begin{aligned}\mathop{\mathbb{E}}_{W_2}\left[\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{2}\sum_{k=1}^{n}(h_i^k)^2\left(a_l^k-y_l^k\right)^2\right] &= 2n \times \mathop{\mathbb{E}}_{W_2,i,l,k}\left[(h_i^k)^2\left(a_l^k-y_l^k\right)^2\right] && (50)\\ &= 2n \mathop{\mathbb{E}}_{W_2,i,k}\left[(h_i^k)^2\right]\mathop{\mathbb{E}}_{W_2,l,k}\left[\left(a_l^k-y_l^k\right)^2\right] && (51)\\ &\geq nC(\sigma_X,\sigma_W,p,k)\gamma', && (52)\end{aligned}$$

where the last step follows from Theorem 1, and $\gamma' = \mathbb{E}_{W_2,l,k}\left[\left(a_l^k - y_l^k\right)^2\right]/2$ is a constant that only depends on the first layer weights W_1, as the expectation is over all W_2. Similarly, we can expand the second term of equation 48 as follows.

$$\begin{aligned}&\mathop{\mathbb{E}}_{W_2}\left[\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{2}\sum_{k_1=1}^{n}\sum_{k_2=1}^{n}h_i^{k_1}h_i^{k_2}\left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)\right] && (53)\\ &= 2n^2 \times \mathop{\mathbb{E}}_{W_2,i,l,k_1,k_2}\left[h_i^{k_1}h_i^{k_2}\left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)\right]. && (54)\end{aligned}$$

As before, we define two random variables in this context, $R_1 = h_i^{k_1}h_i^{k_2}$ and $R_2 = \left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)$, where $0 \leq i \leq N$, $0 \leq k_1, k_2 \leq n$ and $1 \leq l \leq 2$ are random variables all drawn uniformly within their corresponding range. We similarly note that $\mathbb{E}_{W_2,i,l,k_1,k_2}\left[h_i^{k_1}h_i^{k_2}\left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)\right] = \mathbb{E}_{W_2,i,l,k_1,k_2}[R_1R_2]$. Like before, $h_i^{k_1}$ individually is independent of $\left(a_l^{k_1}-y_l^{k_1}\right)$ and $h_i^{k_2}$ is independent of $\left(a_l^{k_2}-y_l^{k_2}\right)$, implying that $h_i^{k_1}h_i^{k_2}$ is independent of $\left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)$. Thus, we can write $\mathbb{E}_{W_2,i,l,k_1,k_2}[R_1R_2] = \mathbb{E}_{W_2,i,k_1,k_2}[R_1]\,\mathbb{E}_{W_2,l,k_1,k_2}[R_2]$.

Furthermore, we have that $\left(a_l^{k_1}-y_l^{k_1}\right)$ is itself independent of $\left(a_l^{k_2}-y_l^{k_2}\right)$, as $P(k_2|k_1) = P(k_2)$ as they are independently chosen. Thus, it similarly follows that $\mathbb{E}_{W_2,l,k_1,k_2}[R_2] = \mathbb{E}_{W_2,l,k_1}\left[\left(a_l^{k_1}-y_l^{k_1}\right)\right]\mathbb{E}_{W_2l,k_2}\left[\left(a_l^{k_2}-y_l^{k_2}\right)\right]$. Lastly note $\mathbb{E}_{W_2,l,k_1}\left[\left(a_l^{k_1}-y_l^{k_1}\right)\right] = \mathbb{E}_{W_2,k_1}\left[\left(a_0^{k_1}+a_1^{k_1}-y_0^{k_1}-y_1^{k_1}\right)\right] = \mathbb{E}_{W_2,k_1}[0] = 0$. This results in,

$$\begin{aligned}&\mathop{\mathbb{E}}_{W_2}\left[\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{2}\sum_{k_1=1}^{n}\sum_{k_1=1}^{n}h_i^{k_1}h_i^{k_2}\left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)\right] && (55)\\ &= 2n^2 \times \mathop{\mathbb{E}}_{W_2,i,l,k_1,k_2}\left[h_i^{k_1}h_i^{k_2}\left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)\right] && (56)\\ &= 2n^2 \times \mathop{\mathbb{E}}_{W_2,i,k_1,k_2}\left[h_i^{k_1}h_i^{k_2}\right]\mathop{\mathbb{E}}_{W_2,l,k_1,k_2}\left[\left(a_l^{k_1}-y_l^{k_1}\right)\left(a_l^{k_2}-y_l^{k_2}\right)\right] && (57)\\ &= 2n^2 \times \mathop{\mathbb{E}}_{W_2,i,k_1,k_2}\left[h_i^{k_1}h_i^{k_2}\right]\mathop{\mathbb{E}}_{W_2,l,k_1}\left[\left(a_l^{k_1}-y_l^{k_1}\right)\right]\mathop{\mathbb{E}}_{W_2,l,k_2}\left[\left(a_l^{k_2}-y_l^{k_2}\right)\right] && (58)\\ &= 0, && (59)\end{aligned}$$

where the last step follows as $\mathbb{E}_{W_2,l,k_1}\left[\left(a_l^{k_1}-y_l^{k_1}\right)\right] = 0$. Thus, replacing the terms in equation 48, we have

$$\mathop{\mathbb{E}}_{W_2}\left[\mathbb{E}[\|W_2 - W_2'\|^2]\right] \geq n\alpha^2 C(\sigma_X,\sigma_W,p,k)\gamma' \qquad (60)$$

However, note here that the above expression results from *all* the weights in each iteration. In truth, as a $(1-p)^k$ proportion of the weights remain in W_2, only a $(1-p)^k$ fraction of the weights in W_2 will be updated. This indicates that the pruning corrected value of $\mathbb{E}[\|W_2 - W_2'\|^2]$, denoted by $\mathbb{E}_{corr}[\|W_2 - W_2'\|^2]$ must be,

$$\mathbb{E}_{corr}[\|W_2 - W_2'\|^2] = \mathop{\mathbb{E}}_{i,j}[\|W_2^{ij} - (W_2^{ij})'\|^2 \delta_{ij}], \tag{61}$$

where δ_{ij} is a variable which will be 0 if $W_2^{ij} = 0$ (pruned) or will be 1, depending on the weight magnitude itself. However, note that as the magnitude of the weight does not affect the magnitude of change for W_2^{ij} in equation 43, δ_{ij} in equation 61 is independent of $W_2^{ij} - (W_2^{ij})'$, and thus we can write $\mathbb{E}_{corr}[\|W_2 - W_2'\|^2] = \mathbb{E}_{i,j}[\|W_2^{ij} - (W_2^{ij})'\|^2]\mathbb{E}[\delta_{ij}] = (1-p)^k \mathbb{E}[\|W_2 - W_2'\|^2]$. This also indicates that the pruning corrected average gradient norm $\mathbb{E}[\|W_2 - W_2'\|^2]$, denoted by $E_{WG}(W_2, W_2')$ will be

$$\mathop{\mathbb{E}}_{W_2}[E_{WG}(W_2, W_2')] = \frac{\mathbb{E}_{W_2}\left[\mathbb{E}_{corr}[\|W_2 - W_2'\|^2]\right]}{(1-p)^k} = \mathop{\mathbb{E}}_{W_2}\left[\mathbb{E}[\|W_2 - W_2'\|^2]\right] \tag{62}$$

$$\geq n\alpha^2 C(\sigma_X, \sigma_W, p, k)\gamma' = \alpha^2 C(\sigma_X, \sigma_W, p, k)\gamma, \tag{63}$$

where we substitute $\gamma = n\gamma'$, which yields our result. □

S-Shape learning rate for arbitrary depth networks. Even for the general case of arbitrary depth D, as shown in Proposition 1, we find that the resulting trajectory still resembles an S-shape, as the expression which controls the average activation norm has a similar form (see equation 28). We also note that Theorem 2's result directly applies to the general case discussed in Proposition 1 as well. We present the extension of Theorem 2 to the more general case of arbitrary depth in the following Corollary.

Corollary 1. *In Proposition 1's setting, we consider a single epoch of weight update for the network across a training dataset $S = \{(X_1, Y_1), .., (X_n, Y_n)\}$ using the cross-entropy loss, where $Y_i \in \{0, 1\}$. Let α denote the learning rate. Let us denote the R.H.S of equation 28 by $C_D(\sigma_X, \sigma_W, p, k)$. Let the final layer weights before and after one training epoch be W_D and W_D' respectively. We have,*

$$\mathbb{E}_{W_D \sim \mathcal{N}_k(0, \sigma_W^2 I)}[E_{WG}(W_D, W_D')] \geq \alpha^2 \gamma C_D(\sigma_X, \sigma_W, p, k), \tag{64}$$

where $\mathcal{N}_k(0, \sigma_w^2 I)$ represents a normal distribution $\mathcal{N}(0, \sigma_w^2 I)$ initialization of W_D, followed by k iterations of pruning, and γ is a constant.

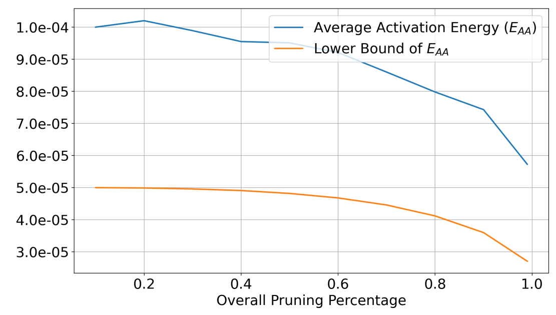
Proof. The proof directly follows from the fact that in the proof of Theorem 2, we only make use of the backpropagatory signals, and therefore the result holds independent of the number of layers before the final layer. Furthermore, as the result in Theorem 2 only depends on the average activation norm of the final hidden layer, we thus have $C_D(\sigma_X, \sigma_W, p, k)$ (from equation 28) in the R.H.S instead of $C(\sigma_X, \sigma_W, p, k)$ which applies only to the $D = 1$ case (single hidden layer). This completes the proof. □

B Appendix: Empirical Testing of Theoretical Results

In this section, we verify the theoretical results from Theorems 1 and 2, via experiments on 2-layered fully connected neural networks.

Experiments for Theorem 1: For Theorem 1, we compute the average activation norm for a neural network with 1000 hidden nodes, 100 input dimensionality, on a toy classification problem where the ground truth function is known. Given random weights $W_0 \in \mathbb{R}^{100\times 2}$ and the input X, the ground truth classification function is set to $\mathbb{1}_{\mathbb{R}^+}\left(W_0^T X\right)$. We set $\sigma_x = 0.1$ and $\sigma_w = 0.01$. Subsequently, we prune the weights of both layers, and choose the amount of overall pruning percentage $(1-(1-p)^k)$ from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$.

Results are shown in Fig. 3 (left figure). We find that in each case, the L.H.S of equation (1) is always greater than the R.H.S. Furthermore, as we increase the pruning percentage, we find that both the L.H.S



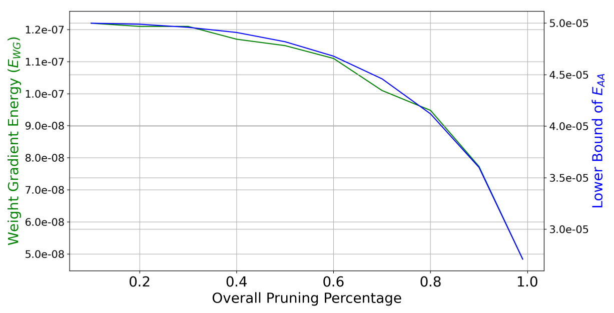


Figure 3: Figures showing the result of the experiments conducted for testing the results in Theorems 1 and 2. In both settings, we analyze two-layer neural networks. In the left figure, we showcase the average activation norm alongside its lower bound from Theorem 1. In the right figure, we showcase the mean value of the average gradient norm across multiple weight initializations alongside the lower bound of Theorem 1.

and R.H.S of (1) get smaller and nearer to each other. Furthermore, in each case, we find that the trend of the R.H.S of (1) quite accurately reflects the trend of the empirically observed average activation norm.

Experiments for Theorem 2: For Theorem 2, we test how the average gradient norm changes as the $C(\sigma_X, \sigma_W, p, k)$ term in the R.H.S of (2) changes. We follow the same overall setup as in the previous experiment. As the L.H.S of (2) represents the average gradient norm, further averaged across many initializations of W_2, we record the average gradient norm for one iteration for 500 initializations of W_2 and return the mean value of the average gradient norm. Furthermore, we fix the learning rate to 0.5 for all updates. Following the same setup as in the experiments of Theorem 1, we choose the overall pruning percentage from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$, and observe the changes in the mean value of the average gradient norm. The 2-layer networks are updated after one iteration of gradient descent using a cross-entropy loss, and the difference in weight values is subsequently used to compute the average gradient norm measures.

The results are shown in Fig. 3 (right figure). We find that the changes in $C(\sigma_X, \sigma_W, p, k)$ very precisely reflect the changes in E_{WG}. As $C(\sigma_X, \sigma_W, p, k)$ decreases in response to the increase of pruning percentage p, E_{WG} decreases proportionally. As we have shown in the main paper that $1/\sqrt{C(\sigma_X, \sigma_W, p, k)}$ follows an S-shaped pattern in response to k (Remark 2), the observation that E_{WG} relates proportionately to $C(\sigma_X, \sigma_W, p, k)$ directly validates the usefulness of SILO in this setting of 2-layer neural networks.

C Appendix: Supplementary Experimental Results

In the Appendix, we show some additional experimental results. Specifically,

1. In Section C.1, we show the experimental results on the distribution of weight gradients and hidden representations using AlexNet, ResNet-20 & VGG-19 via structured and unstructured pruning methods.
2. In Section C.2 we demonstrate the effect of SILO on weight update.
3. In Section C.3, we explain why SILO helps to improve generalization performance.
4. In Section C.4, we show the experimental results for more values of λ for experiments in Tables 2 - 7.

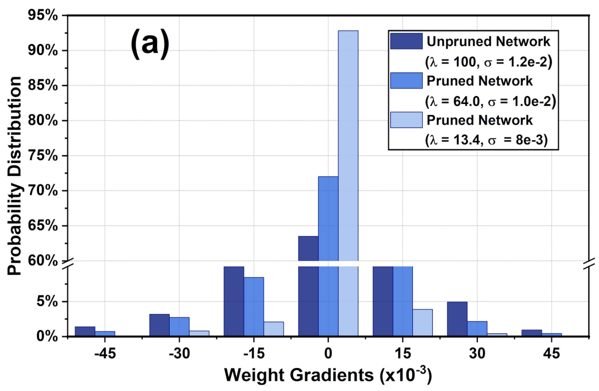
Network	Train Steps	Batch	Learning Rate Schedule	BatchNorm
AlexNet	781K Iters	64	warmup to 1e-2 over 150K, 10x drop at 300K, 400K	No
	Pruning Metric: Unstructured Pruning - Layer Weight			
ResNet-20	63K Iters	128	warmup to 3e-2 over 20K, 10x drop at 20K, 25K	Yes
	Pruning Metric: Unstructured Pruning - Global Gradient			
VGG-19	63K Iters	128	warmup to 1e-1 over 10K, 10x drop at 32K, 48K	Yes
	Pruning Metric: Structured Pruning - L1 Norm			

Table 15: Architectures and training details used in the Appendix.

C.1 More Experimental Results on the Distribution of Weight Gradients and Hidden Representations

We now present more results on the distribution of weight gradients and hidden representations using popular networks and pruning methods in Figs. 4 - 7. The configuration for each network is given in Table 15.

We observe that the results in Figs. 4 - 7 largely mirror those in Fig. 1(a). Specifically, in Fig. 4, we show the distribution of weight gradients and hidden representations when pruning a fully connected ReLU-based network using global magnitude with batch normalization applied to each hidden layer. We observe that the weight gradients of the unpruned network have a standard deviation of 1.2e-2 while the weight gradients of the pruned network (λ = 13.4) have a smaller standard deviation of 8e-3. Similarly, in Fig. 7(a), the standard deviation of the unpruned VGG-19's weight gradients also reduces from 1.2e-2 to 9e-3 when the VGG-19 is iteratively pruned to λ = 13.4. Lastly, we note that the corresponding distributions of hidden representations are also shown in Fig. 4 (b) - 7 (b), which largely mirror those in Fig. 1(b).



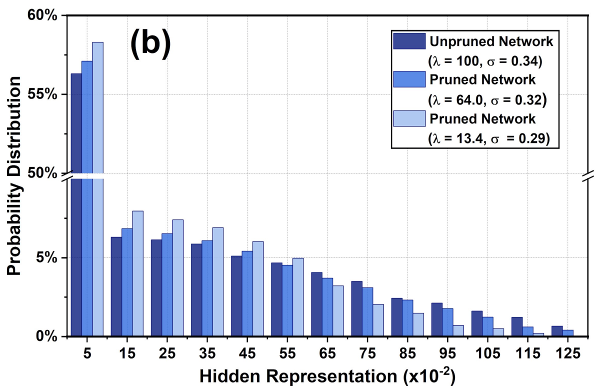
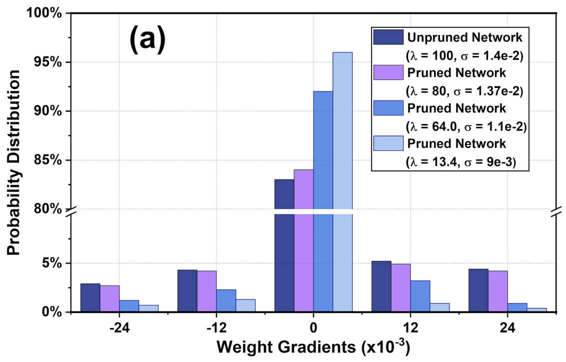


Figure 4: (a) The distribution of weight gradients when iteratively pruning a fully connected ReLU-based network using global magnitude, where λ is the percent of weights remaining and σ is the standard deviation of the distribution. (b) The corresponding distribution of hidden representations (i.e., post-activation outputs of all hidden layers). The main difference with Fig. 1 is that the batch normalization is used here.



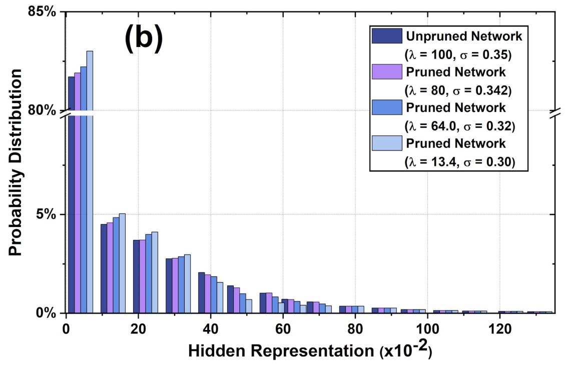
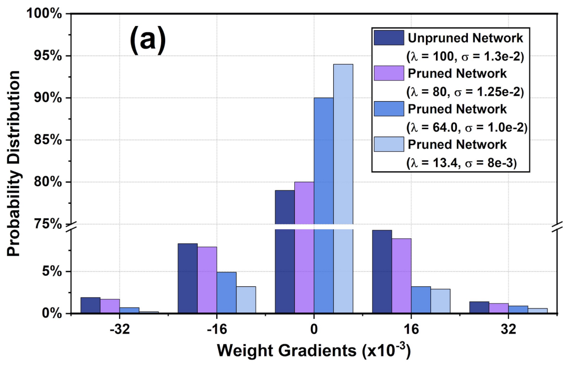


Figure 5: (a) The distribution of weight gradients when iteratively pruning the AlexNet network using the layer magnitude pruning method, where λ is the percent of weights remaining and σ is the standard deviation of the distribution. (b) The corresponding distribution of hidden representations (i.e., post-activation outputs of all hidden layers). Please note that there is a line breaker in the vertical axis.



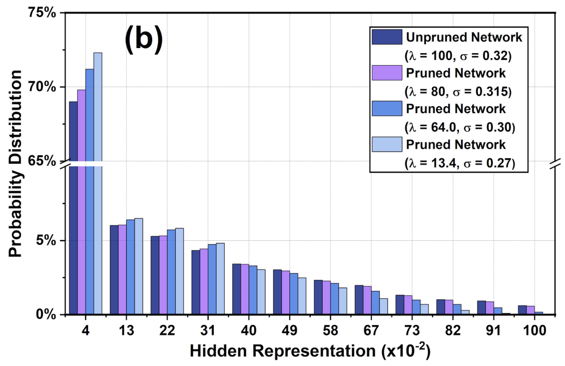
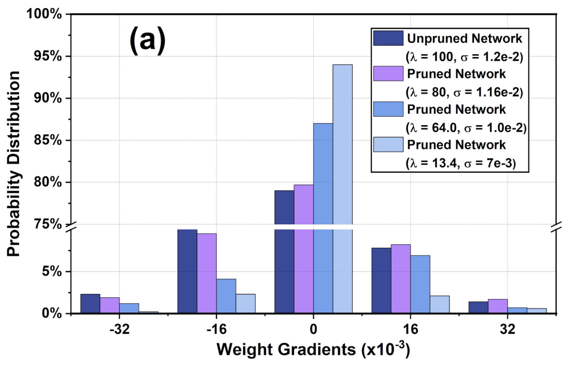


Figure 6: (a) The distribution of weight gradients when iteratively pruning the ResNet-20 network using the global gradient pruning method, where λ is the percent of weights remaining and σ is the standard deviation of the distribution. (b) The corresponding distribution of hidden representations (i.e., post-activation outputs of all hidden layers). Please note that there is a line breaker in the vertical axis.



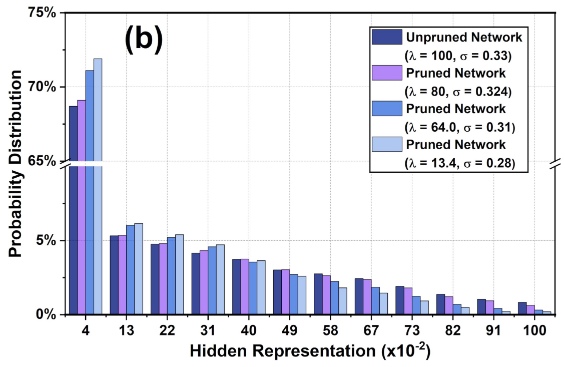


Figure 7: (a) The distribution of weight gradients when iteratively pruning the VGG-19 network using the structured filter pruning method (Li et al., 2017), where λ is the percent of weights remaining and σ is the standard deviation of the distribution. (b) The corresponding distribution of hidden representations (i.e., post-activation outputs of all hidden layers). Please note that there is a line breaker in the vertical axis.

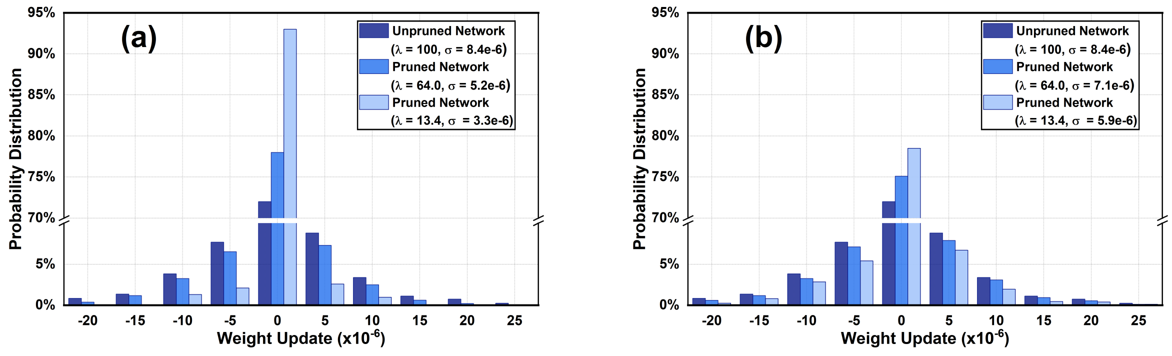


Figure 8: Distribution of weight update (weight gradient×learning rate) without (a) and with SILO (b) when iteratively pruning a fully connected ReLU network (without batch normalization) using global magnitude.

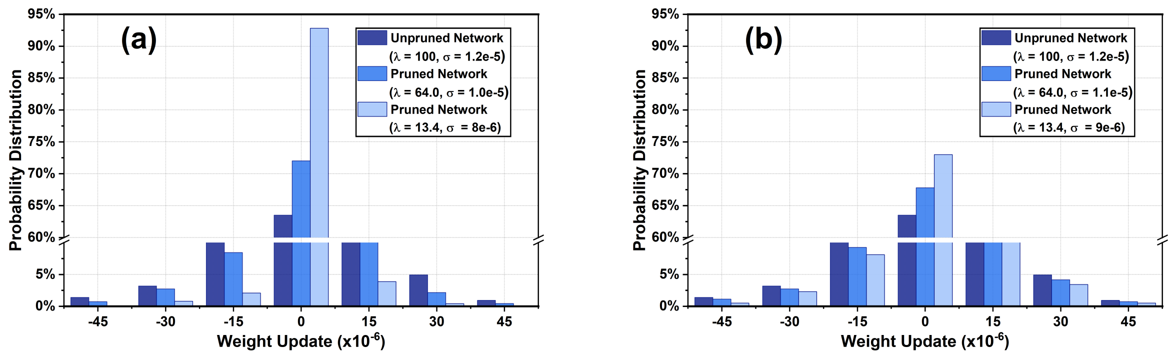


Figure 9: Distribution of weight update (weight gradient×learning rate) without (a) and with SILO (b) when iteratively pruning a fully connected ReLU network (with batch normalization) using global magnitude.

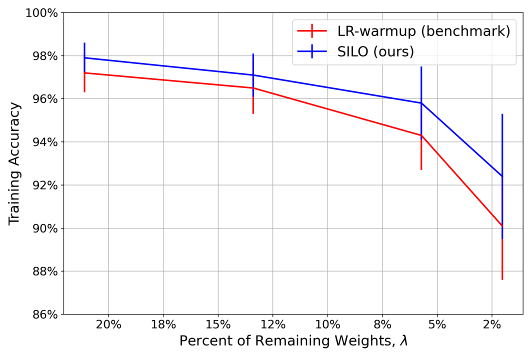
C.2 Effect of SILO on Weight Update

To examine the effect of SILO on weight update (weight gradient $\times$ learning rate), we repeat the experimental setup in Section 3.1 and compare the distribution of weight update with and without using SILO in Fig. 8. It can be seen that, with SILO, the distribution of weight update of pruned networks becomes less centralized, i.e., compare pruned network ($\lambda = 13.4$) in Fig. 8 (a) to pruned network ($\lambda = 13.4$) in Fig. 8 (b). This suggests that SILO works as expected and mitigates the issue of decreasing weight gradients. Similar performance trends can also be observed when using batch normalization (see Fig. 9).

C.3 Why does SILO improve generalization performance?

Our initial motivation behind SILO is based on the observations in Section 3.1, where we find that average weight gradients undergo a significant reduction as the network gets pruned. Our intuitive hypothesis is that in addition to hampering the ability of the pruned network to fit the data well, it also prevents the network from escaping local optimum points due to the smaller weight updates. We thus hypothesize that by fixing average weight gradients, SILO improves training accuracy, which is indeed observed to be the case (see Fig. 10). The Figure shows that the pruned network is better trained with SILO, resulting in a higher training accuracy for ResNet-20 and VGG-19 networks on CIFAR-10. Thus, a significant reason for the generalization improvements observed with SILO can be traced to better training performance itself.

We next reason why SILO improves training accuracy. First, we note that pruned networks intrinsically lose a significant amount of their base complexity, affecting their ability to fit labels successfully. Second, the weight gradients for the pruned network undergo are significantly smaller than the original unpruned network (Section 3.1), which further reduces the value of the weight gradient norm itself. Recent studies



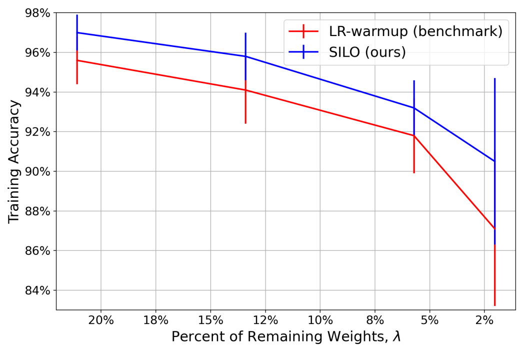


Figure 10: Training Accuracy of ResNet-20 (left) and VGG-19 (right) when they are iteratively pruned using global magnitude using CIFAR-10.

(see (Gat et al., 2022)) have shown that weight gradient norm effectively encodes a surrogate complexity of the network. This was found in Theorem 3.5 in (Gat et al., 2022), which shows that the generalization gap directly depends on the gradient norm, and it increases as the gradient norm increases. Thus, due to smaller gradient norms of the pruned network, the effective complexity of the pruned network reduces even further.

Thus, as the network is pruned, we should expect the effective complexity of the network to decrease quite significantly due to the two reasons mentioned in the previous paragraph. This is expected to negatively impact the training accuracy of the network (Fig. 10), which should negatively impact the test accuracy, as underfitting will yield a lower test accuracy. As SILO increases the learning rate appropriately during pruning, it thus counters the expected reduction of the average gradient norm, thus effectively increasing the surrogate complexity of the pruned network, avoiding underfitting, and improving training accuracy.

C.4 Experimental Results for More Values of λ

We note that, in Tables 2 - 7, we only show the experimental results for some key values of λ. In this subsection, we show the results for more values of λ in Tables 16 - 21. The LR schedules (i.e., LR-warmup) from Table 16 - 20 are from (Frankle & Carbin, 2019), (Frankle et al., 2020), (Zhao et al., 2019), (Chin et al., 2020) and (Renda et al., 2019), respectively. The LR schedule (i.e., cosine decay) in Table 21 is from (Dosovitskiy et al., 2020). The implementation details are provided in the top row of each table and the descriptions of each benchmark LR schedule are summarized in Table 1. It should be noted that, for the IMP method examined in this work, we rewind the unpruned weights to their values during training (e.g., epoch 6), in order to obtain a more stable subnetwork (Frankle et al., 2019).

Due to the width of these tables, we rotate them and present the results in the landscape style. We observe that the performance of SILO for other values of λ still outperforms the selected LR schedule benchmarks. For example, in Table 18, SILO achieves an improvement of 4.0% at $\lambda = 5.72$ compared to LR warmup.

Params: 227K	**Train Steps**: 63K Iters		**Batch**: 128	**Batch Norm**: Yes		**Optimizer**: SGD		**Rate**: 0.2
Percent of Weights Remaining, λ	100	64	40.9	32.8	26.2	13.4	8.59	5.72
constant LR (1e-2)	90.4±0.4	89.7±0.5	88.9±0.7	88.1±0.9	87.5±0.8	86.0±0.9	82.8±0.9	79.1±0.8
LR decay (3e-2, 63K)	91.2±0.4	91.0±0.3	90.3±0.5	89.8±0.4	89.0±0.7	87.7±0.6	83.9±0.6	79.8±0.7
cyclical LR (0, 3e-2, 8K)	90.8±0.3	90.4±0.5	90.1±0.4	89.7±0.6	88.2±0.7	87.6±0.8	84.1±0.6	80.3±0.7
LR-warmup (3e-2, 20K, 20K, 25K, Nil)	**91.7±0.2**	91.5±0.3	90.8±0.5	90.3±0.4	89.8±0.6	88.2±0.6	85.9±0.9	81.2±1.1
SILO (3e-2, 4e-2, 20K, 20K, 25K, Nil)	91.7±0.2	**91.9±0.4**	**91.2±0.6**	**90.8±0.5**	**90.3±0.4**	**89.7±0.5**	**87.5±0.8**	**82.7±1.2**

Table 16: Performance comparison (averaged top-1 test accuracy ± std over 5 runs) of pruning ResNet-20 on CIFAR-10 dataset using the global magnitude pruning method (Blalock et al., 2020). LR-warmup is the standard implementation used in (Frankle & Carbin, 2019; Frankle et al., 2020).

Params: 139M	**Train Steps**: 63K Iters		**Batch**: 128	**Batch Norm**: Yes		**Optimizer**: SGD		**Rate**: 0.2
Percent of Weights Remaining, λ	100	64	40.9	32.8	26.2	13.4	8.59	5.72
constant LR (8e-3)	91.3±0.3	90.5±0.5	89.5±0.6	88.8±0.6	87.4±0.7	85.8±0.6	82.2±1.4	73.7±1.3
LR decay (1e-2, 63K)	92.0±0.5	90.9±0.5	90.2±0.5	89.4±0.4	88.6±0.5	87.4±0.6	83.3±0.8	75.4±0.9
cyclical LR (0, 3e-2, 15K)	92.3±0.6	91.2±0.6	90.4±0.4	89.8±0.5	89.1±0.6	88.6±0.8	83.7±1.0	75.7±1.2
LR-warmup (1e-1, 10K, 32K, 48K, Nil)	92.2±0.3	91.3±0.2	90.6±0.4	90.2±0.5	89.8±0.8	89.2±0.8	84.5±0.9	76.5±1.1
SILO (4e-2, 6e-2, 10K, 32K, 48K, Nil)	**92.6±0.4**	**91.8±0.6**	**90.9±0.5**	**90.6±0.6**	**90.3±0.6**	**89.8±0.9**	**86.1±0.8**	**78.5±1.0**

Table 17: Performance comparison (averaged top-1 test accuracy ± std over 5 runs) of pruning VGG-19 on CIFAR-10 dataset using the global gradient pruning method (Blalock et al., 2020). LR-warmup is the standard implementation used in (Frankle & Carbin, 2019; Frankle et al., 2020; Liu et al., 2019b).

Params: 1M	**Train Steps**: 117K Iters		**Batch**: 128	**Batch Norm**: Yes		**Optimizer**: SGD	**Pruning Rate**: 0.2	
Percent of Weights Remaining, λ	100	64	40.9	32.8	26.2	13.4	8.59	5.72
constant LR (1e-2)	73.7±0.4	72.8±0.4	71.4±0.5	70.3±0.8	68.1±0.7	63.5±0.6	60.8±1.1	59.1±1.2
LR decay (4e-2, 117K)	74.3±0.3	73.5±0.9	72.2±0.4	71.2±0.8	69.0±0.6	64.7±0.7	62.6±1.2	60.3±1.4
cyclical LR (0, 4e-2, 24K)	74.4±0.4	73.2±0.5	72.0±0.3	70.9±0.6	69.4±0.6	65.1±0.9	63.0±1.1	60.8±1.3
LR-warmup (12e-2, 58K, 58K, 92K, Nil)	74.6±0.5	73.4±0.6	72.3±0.4	71.5±0.7	69.6±0.8	65.8±0.9	63.9±1.0	61.2±0.9
SILO (7e-2, 5e-2, 58K, 58K, 92K, Nil)	**75.0±0.5**	**74.1±0.3**	**72.9±0.6**	**72.4±0.7**	**70.8±0.8**	**67.6±1.2**	**65.7±1.1**	**63.7±1.0**

Table 18: Performance comparison (averaged top-1 test accuracy ± standard deviation over 5 runs) of pruning DenseNet-40 on CIFAR-100 dataset using LAMP. LR-warmup is adapted from the standard implementation used in (Zhao et al., 2019; Huang et al., 2017).

Params: 2.36M	**Train Steps**: 78K Iters		**Batch**: 128	**Batch Norm**: Yes		**Optimizer**: SGD	**Pruning Rate**: 0.2	
Percent of Weights Remaining, λ	100	64	40.9	32.8	26.2	13.4	8.59	5.72
constant LR (1e-2)	72.7±0.2	71.5±0.4	70.4±0.5	69.8±1.1	68.2±0.9	64.5±0.6	63.8±1.1	62.1±1.2
LR decay (15e-1, 78K)	73.3±0.3	72.1±0.3	71.8±0.4	70.9±1.0	69.4±0.6	65.9±0.7	65.1±0.8	64.0±1.1
cyclical LR (0, 5e-2, 14K)	73.5±0.4	72.3±0.5	72.0±0.3	71.5±0.7	69.6±0.6	66.7±0.9	65.3±1.0	64.3±1.2
LR-warmup (1e-1, 23K, 23K, 46K, 62K)	73.7±0.4	72.5±0.4	72.3±0.5	72.1±0.8	70.5±0.9	67.3±0.8	66.2±1.1	64.8±1.5
SILO (5e-2, 5e-2, 23K, 23K, 46K, 92K)	**74.0±0.5**	**73.0±0.3**	**72.9±0.8**	**72.5±0.6**	**71.0±0.7**	**68.8±0.8**	**67.6±1.2**	**66.8±1.4**

Table 19: Performance comparison (averaged top-1 test accuracy ± standard deviation over 5 runs) of pruning MobileNetV2 on CIFAR-100 dataset using LAP. LR-warmup is adapted from the standard implementation used in (Chin et al., 2020).

Params: 25.5M	**Train Steps**: 70K Iters		**Batch**: 128		**Batch Norm**: Yes		**Optimizer**: SGD	**Rate**: 0.2
Percent of Weights Remaining, λ	100	64	40.9	32.8	26.2	13.4	8.59	5.72
constant LR (1e-2)	75.3±0.2	75.2±0.3	74.6±0.7	74.2±0.8	73.9±0.7	72.2±0.9	70.5±0.6	69.2±0.9
LR decay (3e-2, 70K	76.5±0.2	76.1±0.5	75.8±0.6	75.6±0.5	75.1±0.5	73.9±0.7	72.7±0.8	70.5±0.6
cyclical LR (0, 5e-2, 20K)	76.8±0.3	76.9±0.5	77.0±0.6	76.5±0.5	75.5±0.6	74.5±0.6	73.4±0.8	71.2±0.7
LR-warmup (1e-1, 4K, 23K, 46K, 62K)	77.0±0.1	77.2±0.2	76.9±0.5	76.6±0.2	75.8 ±0.3	75.2±0.4	73.8±0.5	71.5±0.4
SILO (5e-2, 5e-2, 4K, 23K, 46K, 62K)	**77.2±0.2**	**77.4±0.3**	**77.0±0.4**	**76.8±0.4**	**76.1±0.7**	**75.8±0.6**	**75.2±0.8**	**73.8±0.6**

Table 20: Performance comparison (averaged top-1 test accuracy ± standard deviation over 5 runs) of pruning ResNet-50 on ImageNet dataset using the IMP pruning method (Le & Yang, 2015). LR-warmup is adapted from the standard implementation used in (Frankle et al., 2020; Renda et al., 2019).

Params: 86M	**Train Steps**: 2K Iters		**Batch**: 1024		**Batch Norm**: Yes		**Optimizer**: Adam	**Rate**: 0.2
Percent of Weights Remaining, λ	100	64	40.9	32.8	26.2	13.4	8.59	5.72
constant LR (5e-5)	97.4±0.2	97.6 ±0.3	97.5 ±0.4	96.4 ±0.5	96.0±0.7	86.3±0.6	83.0±0.9	80.1±0.8
cosine decay (1e-4, 2K	98.0±0.3	98.2 ± 0.3	97.9 ± 0.4	97.2 ± 0.2	96.5±0.6	87.7±0.5	84.1±1.0	81.6±1.1
cyclical LR (0, 1e-4, 2K)	97.8±0.4	98.0 ± 0.2	97.8 ± 0.3	97.0 ± 0.2	96.5±0.6	87.2±0.9	83.4±0.6	81.0±1.1
LR-warmup (4e-4, 0.3K, 0.5K, 0.9K, 1.3K)	98.0±0.3	98.4 ±0.3	97.8 ±0.5	97.3 ±0.6	96.8±0.7	88.1±0.9	84.4±0.8	82.1±0.9
SILO (1e-4, 3e-4, 0.3K, 0.5K, 0.9K, 1.3K)	**98.0±0.3**	**98.5±0.4**	**98.0±0.3**	**97.7±0.5**	**97.4±0.6**	**89.2±0.8**	**85.5±0.9**	**83.4±0.8**

Table 21: Performance comparison (averaged top-1 test accuracy ± standard deviation over 5 runs) of pruning pruning Vision Transformer (ViT-B-16) on CIFAR-10 using IMP (Frankle & Carbin, 2019). Cosine decay is the standard implementation used in (Dosovitskiy et al., 2020).