Cyclical Pruning for Sparse Neural Networks

Suraj Srinivas[1] Andrey Kuzmin[2] Markus Nagel[2] Mart van Baalen[2] Andrii Skliar[2]
Tijmen Blankevoort[2]

[1] Idiap Research Institute & EPFL, Switzerland
[2] Qualcomm AI Research, Netherlands

Abstract

Current methods for pruning neural network weights iteratively apply magnitude-based pruning on the model weights and re-train the resulting model to recover lost accuracy. In this work, we show that such strategies do not allow for the recovery of erroneously pruned weights. To enable weight recovery, we propose a simple strategy called cyclical pruning which requires the pruning schedule to be periodic and allows for weights pruned erroneously in one cycle to recover in subsequent ones. Experimental results on both linear models and large-scale deep neural networks show that cyclical pruning outperforms existing pruning algorithms, especially at high sparsity ratios. Our approach is easy to tune and can be readily incorporated into existing pruning pipelines to boost performance.

1. Introduction

The dominant paradigm for training and inference of deep neural networks uses dense parameter tensors and hardware optimized for dense computations. However, sparse tensor multiplications can be more compute, memory and power efficient, all of which are important considerations for low-power mobile devices. To utilize sparsity, we require methods to either train sparse neural networks from scratch or convert existing dense models to sparse ones. Fortunately, it has been shown that pre-trained dense deep neural networks can be easily sparsified using simple heuristics involving magnitude pruning and re-training [15, 37]. One such commonly-used heuristic is gradual pruning, which iteratively prunes weights and follows it with re-training, each time increasing the number of weights pruned.

On the other hand, recent review papers [3, 13] have shown that is difficult to improve upon these simple heuristics, and as a result, current state-of-the-art approaches [13] still rely on such techniques. In this work, we connect these heuristics to projected gradient descent (PGD), a well-known algorithm for constrained optimization. Similar to gradual pruning, PGD involves alternating between magnitude pruning and re-training. However, as we show in the paper, this analogy breaks down for the simple case of pruning a single weight. To bridge this gap, we propose *cyclical pruning*, a simple strategy that uses a cyclical schedule for pruning rather than a monotonically increasing one. Our experimental results show that cyclical pruning outperforms gradual pruning across datasets on various models, especially at large sparsity ratios. This approach does not introduce any hard-to-tune hyper-parameters and can be readily incorporated into existing pruning pipelines.

Overall, our contributions are:

- We propose *cyclical pruning*, a simple pruning strategy that allows for recovery of previously pruned weights.
- We show that recovery of pruned weights is crucial in the context of pruning linear models, especially when the solution space is non-degenerate.
- We show improvements over gradual pruning on CIFAR-10 and Imagenet datasets across several models, especially at large sparsity ratios.

2. Related Work

Pruning in neural networks involves either structured pruning which removes entire neurons, or unstructured pruning which removes individual weights. Structured pruning [17, 19, 27], typically does not require any specialized hardware support, as opposed to unstructured pruning which requires explicit support for sparse computations [6, 18], and is the main focus on this paper.

Methods for unstructured pruning of neural network weights typically rely on magnitude pruning and re-training [14, 15, 37], and our paper extends these methods to allow for recovery of pruned weights. In these works, each layer can either be pruned to the same level of sparsity by applying magnitude pruning on each layer separately, or

it can be applied once globally. While in this work we use local uniform layerwise sparsity, [1, 24] propose to improve global sparsity by automatically tuning thresholds for magnitude pruning.

Another orthogonal line of work involves replacing magnitude pruning with alternatives that explicitly consider the impact of pruning on the final loss. To this end, second-order [16, 26], and Fisher approximations [34, 35] of the loss function have been employed. However, recent work [25] has shown that these methods do not necessarily improve upon magnitude pruning, especially when combined with fine-tuning.

Distinct from the approaches considered above, probabilistic approaches to pruning involve approximating the original pruning problem via stochastic relaxations [7, 29, 30, 32]. These typically involve stochastic optimization over binary gate variables, in addition to the usual optimization over weights. However, recent work [13] has shown that such techniques often perform on par with, simpler magnitude pruning based approaches, which is the focus of this paper.

Also related is the lottery-ticket hypothesis [11, 12], which states that there exists a pruning mask for every initialization of a deep model that allow for training only the the resulting sparse model from scratch, without the need to alter the pruning mask. While we propose to refine the mask during pruning using our method, we do not check whether these masks also correspond to lottery tickets, as this is outside the scope of this work.

Weight recovery has been an important consideration recently in the context of training sparse neural networks from scratch. While [10, 21] use gradient updates to perform weight recovery, [9] use momentum to do the same. However, both methods place the constraint that intermediate models obtained during the course of optimization are also sparse, which places heavy restrictions on the weight recovery methods. However, no such restrictions apply to our case. [14, 28] use gradient updates computed on a proxy sparse model by using the straight-through estimator (STE) similar to [36] and claim that this can lead to weight recovery. However these methods also use gradual pruning, for which we show that weight recovery is unlikely in practice.

3. Methods

In this section, we discuss existing approaches for unstructured pruning which involve magnitude pruning and re-training, and introduce the cyclical pruning algorithm. Given the similarity among these approaches, it is helpful to discuss these as instances of a more general framework for pruning, which we call *time-varying projected gradient descent* (TV-PGD), shown in Algorithm 1. The distinguishing features of this algorithm when compared to classical PGD are the usage of an iteration-dependent (or time-varying) sparsity and learning rate function and updating the pruning mask every Δt iterations. Here, $\texttt{magprune}(\theta, s(t_i))$ refers to magnitude pruning of θ with a sparsity ratio of $s(t_i)$, which refers to *global* pruning in the literature. When this is applied separately layerwise, it is called *local* pruning. Global pruning can result in different sparsity rates for every layer, whereas local pruning ensures every layer is pruned to the same sparsity ratio. In this work, we only consider local pruning, but these methods equally apply to global pruning.

Algorithm 1 Time-Varying Projected Gradient Descent

$\theta \in \mathbb{R}^d$: model weights, $\ell(\theta) \in \mathbb{R}_+$: loss function
$T \in \mathbb{N}$: # iterations, $\Delta t \in \mathbb{N}$: pruning interval
$M \in \{0,1\}^d$: pruning mask, $t_i \in \mathbb{N}_0$: iteration number
$s(t_i) \in [0,1]$: sparsity function, $\eta(t_i) \in \mathbb{R}_+$: learning rate

procedure TV-PGD(θ)
  for $t_i \in [0, T-1]$ iterations **do**
    $\theta \leftarrow \theta - \eta(t_i)\, \nabla_\theta \ell(\theta)$ ▷ (S)GD update
    if $t_i \bmod \Delta t = 0$ **then**
      $M \leftarrow \texttt{magprune}(\theta, s(t_i))$ ▷ Get Mask
    end if
    $\theta \leftarrow \theta \odot M$ ▷ Prune weights in-place
  end for
end procedure

One-Shot Pruning: This simple procedure involves two steps: first magnitude pruning the dense model according to the target sparsity ratio, and then fine-tuning the resulting sparse model [15]. This corresponds to TV-PGD with $\Delta t > T$ and a sparsity function such that $s(0) = s_t$ equal to the final target sparsity. The learning rate $\eta(t_i)$ is monotonically decreasing in accordance with common training practices in deep learning, except at $t_i = 0$, where we have $\eta(0) = 0$.

Gradual Pruning: This involves pruning with a gradually increasing sparsity schedule, with pruning interspersed with fine-tuning. There are two broad variants of this procedure. The first, also called 'iterative pruning' [15] typically performs pruning in few steps (5-10), interspersed with fine-tuning for a large (usually 10+) number of epochs. This corresponds to TV-PGD with linearly increasing $s(t_i) = \frac{s_t\, t_i}{T}$, $\Delta t \sim 10+$ epochs, and a cyclical learning rate schedule $\eta(t_i)$ such that $\eta(t_i \bmod \Delta t)$ is a monotonically decreasing function according to standard training practices, except for $\eta(t_i \bmod \Delta t = 0) = 0$. Note that one-shot pruning emerges as a special case if $\Delta t > T$.

On the other hand, the 'cubic pruning' performs pruning several times (typically 100+), interspersed with a short fine-tuning stage of about few hundred iterations [37]. This corresponds to TV-PGD with a smaller $\Delta t \sim 100$ iterations, and monotonically decreasing $\eta(t_i)$ (as opposed to cyclic) learning rate schedule, and a monotonically increasing cu-

bic sparsity schedule as follows. Here s_{init} is the initial sparsity value and s_t is the target sparsity value.

$$s(t_i) = s_t + (s_{init} - s_t)\left(1 - \frac{t_i}{T}\right)^3 \quad (1)$$

Note that specific implementations of these pruning algorithms may differ slightly from the TV-PGD interpretation, specifically in the usage of in-place pruning which involves directly zero-ing out weights in the weight tensor, but in practice we found no difference in performance between these different variants.

One characteristic of both one-shot and gradual pruning is their lack of a mechanism for *weight recovery*, i.e., the ability of pruned weights to be recovered in future steps, which we define below.

Definition. *(Weight Recovery) is said to have occurred in TV-PGD for some weight j if at any two iterations t_1, t_2 such that $t_2 > t_1$, we have the pruning masks $M_j(t_1) = 0$ and $M_j(t_2) = 1$.*

Intuitively, we expect weight recovery to help in cases where identification of the correct weights to prune are critical, and where one-shot magnitude pruning does not identify these. In such cases, weight recovery can help correct mistakes made by magnitude pruning, and allow pruning of different weights in subsequent steps. However if correct identification of weights to prune does not matter, then we do not expect weight recovery to help. We further elaborate upon this in §4. We observe that in TV-PGD, weight recovery can only occur if a magnitude pruning step immediately follows a gradient update step, resulting in a dense weight tensor.

For one-shot pruning and iterative pruning, we observe that weight recovery is impossible as the magnitude pruning step never occurs immediately after performing a dense SGD update step. For cubic pruning, while weight recovery is technically possible, we found that it is highly improbable in practice. We hypothesize that this happens because weight recovery requires the magnitude of weights after a single update for some pruned weight at j to be larger than that of the smallest unpruned weight, which is improbable owing to usage of relatively small and monotonically decreasing learning rates $\eta(t_i)$ used for fine-tuning. In other words, we require $(\eta(t_i)\nabla_\theta \ell(\theta))_j^2 > \min_{k,\theta_k>0} \theta_k^2$, for pruned weights θ, which is difficult to satisfy when $\eta(t_i)$ is small, and training diverges if $\eta(t_i)$ is set high. We thus require a procedure that can reliably grow back weights that have been pruned previously, and for this purpose, we introduce a simple strategy called *cyclical pruning*.

Cyclical Pruning: We propose to perform pruning with a cyclical pruning schedule rather than a monotonically increasing one. Specifically, we divide the overall pruning schedule into k cycles, and within each cycle the pruning schedule is monotonically increasing. This corresponds to TV-PGD with a periodic $s(t_i)$, where each cycle has a monotonically increasing $s(t_i \bmod T/k)$, and a periodic $\eta(t_i)$ corresponding to a monotonically decreasing $\eta(t_i \bmod T/k)$, and $\Delta t \sim 100$ iterations. We observe that weight recovery always occurs here due to the periodic resetting of the sparsity rates, i.e., $s(t_2) < s(t_1)$ for $t_2 > t_1$ ensures weight recovery. Further, the recovered weights are likely to recover on par with unpruned weights due to the periodic re-setting of learning rates, as this allows for sufficient updates for important weights to recover. In practice, we use the same per-cycle sparsity schedule as in equation 1, and use a different value of s_{init} for the first cycle ($s_{init} = 0$), and subsequent cycles ($s_{init} \sim 0.5 s_t$), although we did not find this to be crucial.

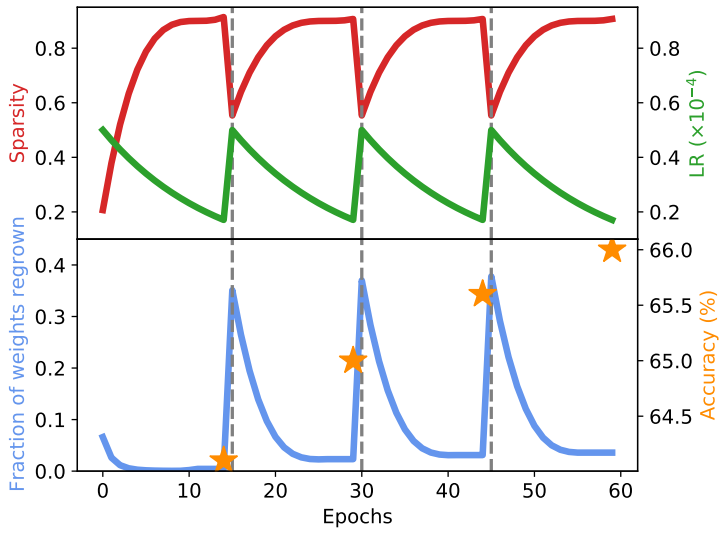

Figure 1. Illustration of cyclical pruning for ResNet18 on Imagenet with $k = 4$ cycles, which specifies both the sparsity and the learning rate schedules. This procedure results in an increase in the number of weights recovered during pruning, and also a corresponding increase in pruned model accuracy across cycles. Note that a weight is considered regrown if it was pruned during any previous training step, but unpruned at the current step.

PGD Pruning: A classical method to perform constrained optimization is projected gradient descent (PGD), which in this case corresponds to TV-PGD with $\Delta t = 1$ and constant functions $\eta(t_i), s(t_i)$. We notice that similar to cyclical pruning, PGD also allows for weight recovery at all steps. However, in practice we find this to be ineffective for pruning deep neural networks, owing to the possible instability caused by pruning at every iteration, and the improbability of recovery due to $(\eta(t_i)\nabla_\theta \ell(\theta))_j^2$ being small. Hence practical considerations such as mini-batching and usage of relatively small, monotonically decreasing learn-

ing rates reduce its effectiveness in practice.

Thus weight recovery is a distinguishing feature of both cyclical pruning and PGD. In the next section, we take a closer look at weight recovery in PGD while pruning linear models, where practical considerations of training deep models do not apply.

4. Is Weight Recovery Necessary?

In this section we study the importance of weight recovery in the simple case of sparse linear regression with a single pruned weight. Formally, let $\hat{\mathbf{y}} = \mathbf{w}^\top \mathbf{X}$, where $\mathbf{y} \in \mathbb{R}^n, \mathbf{X} \in \mathbb{R}^{d \times n}, \mathbf{w} \in \mathbb{R}^d$. Also assume that outputs are generated from an underlying sparse vector, i.e, $\mathbf{y} = \alpha^\top \mathbf{X}$, where $\|\alpha\|_0 = d - 1$, and $\alpha_c = 0$ for some index $c \in \{1, ..., d\}$. Also assume that the problem is over-parameterized ($d > n$). Here, we wish to solve the following.

$$\mathbf{w}^* = \arg \min_{\mathbf{w}, \|\mathbf{w}\|_0 \leq d-1} \|\mathbf{y} - \mathbf{w}^\top \mathbf{X}\|^2 \tag{2}$$

In general, sparse linear regression is NP-hard [31], however if $\mathbf{X}$ satisfies the Restricted Isometry Property (RIP) [5], then efficient polynomial time solutions are known to exist (i.e., PGD) [4]. Note that specifics of the RIP condition are not relevant to our discussion here, and we refer interested readers to [20]. In practice, it is easy to construct approximately RIP matrices by sampling matrix entries from a scaled unit normal distribution [2]. As a result, we henceforth assume that $\mathbf{X}$ satisfies RIP, and begin by making the following observation.

Observation 1. *For problem 2, gradual pruning (with $\eta(0) = 0$) is equivalent to one-shot pruning, and cyclical pruning (with $\Delta t = 1$) is equivalent to projected gradient descent.*

This is true because for pruning a single weight, $s(t_i)$ for any *strictly* monotonically increasing schedule reduces to a step function, and the cyclical schedule $s(t_i)$ reduces a constant function $s(t_i) = s_t$ for $\Delta t = 1$. Further, the usage of mini-batches is unnecessary here, and we use full-batch gradient descent instead. We thus only study one-shot pruning and PGD as proxies for studying gradual and cyclical pruning respectively. For PGD, strong recovery guarantees hold for the sparse linear regression problem under some regularity conditions [4, 20]. Note that discussion of these conditions is out of scope for this paper.

Unfortunately, such guarantees do not hold for one-shot pruning. This is easy to see by applying one-shot pruning on randomly initialized weights. Here the first step involves magnitude pruning which effectively prunes a random weight, and the probability of pruning the correct index c at initialization is only $\frac{1}{d}$. The second step involves re-training, which cannot change the pruned weight. Thus with overwhelming probability $(\frac{d-1}{d})$, random initialization followed by one-shot pruning fails to recover α. This simple counter-example illustrates why such recovery guarantees cannot hold for one-shot pruning.

However this analysis does not reflect standard practice in pruning where one-shot pruning is typically after dense training, and not on randomly initialized weights. For the linear case, dense training corresponds to solving an unconstrained version of equation 2, which is solved via regularized least-squares method.

Assuming some $\lambda > 0$ for regularized least-squares, let $\mathbb{A} = (\mathbf{X}^\top \mathbf{X} + \lambda I)^{-1} \mathbf{X}^\top \mathbf{X}$, then it is easy to see that the least-squares solution is $\mathbf{w} = \mathbb{A}\alpha$. We can use this to construct a problem (i.e, pick α) such that for some index c with $\alpha_c = 0$, we have $c \neq \arg\min_i \mathbf{w}_i^2$. This ensures that magnitude pruning performed on the least squares solution $\mathbf{w}$ does not select the correct index c. One such choice of α is as follows: $\alpha_i = \begin{cases} \mathbb{A}[c, i], & i \neq c \\ 0, & i = c \end{cases}$.

This ensures that $\mathbf{w} = \mathbb{A}\alpha$ has a large magnitude on the c^{th} co-ordinate, causing magnitude pruning to select an incorrect index, which we call the **adversarial** choice of α.

Running simulations on this problem with $d = 5, n = 4, c = 3$, and sampling 10^4 different RIP matrices $\mathbf{X}$, we find that magnitude pruning succeeds in picking the correct index c only $\sim 3\%$ of the time. We also empirically observe that if we set $d >> n$, then this probability tends to zero. We summarize the results of the simulations in the following statement.

Observation 2. *We find empirically that it is possible to choose solutions α for problem 2 such that dense training followed by one-shot pruning fails to recover α with high probability.*

This shows that even in the realistic setting of one-shot pruning applied after dense training, there exists problems such that one-shot pruning fails to select the correct index. Note that PGD is immune to this in principle as the recovery guarantees are independent of initialization.

4.1. When does PGD fail?

Having considered instances where one-shot pruning fails, we now ask the converse question: when does PGD fail? The regularity conditions of PGD recovery [20] indicate that this can happen for severely over-parameterized problems, i.e., $n << d$. In this case, the linear system $\mathbf{y} = \mathbf{w}^\top \mathbf{X}$ not only maintains infinitely many dense solutions for $\mathbf{w}$, but also has multiple sparse solutions. To see this, we rewrite the linear system as $(\mathbf{w} - \alpha)^\top \mathbf{X} = 0$, which implies that $\mathbf{w} - \alpha$ lies in the left-nullspace of $\mathbf{X}$. As an example, assume that the dimensionality of the left null-space is $d_{null} = 2$ for problem dimension $d = 5, n = 3$ and the

Table 1. Simulation results for 100 runs of sparse linear regression performed after regularized least squares on a problem with $d = 5$. We observe that when the problem is severely over-parameterized, both PGD and one-shot pruning perform similarly, while PGD outperforms one-shot pruning in other scenarios. This property also holds for larger image datasets and deep neural networks (see §5.1).

# samples (n)	Choice of α	Prob. of one-shot recovery	Prob. of PGD recovery
2 ($n << d$)	random	**0.33**	**0.32**
	adversarial	**0.12**	**0.11**
4 ($n < d$)	random	0.22	**0.36**
	adversarial	0.02	**0.23**
10 ($n > d$)	random	0.35	**0.64**
	adversarial	0.04	**0.44**

task of pruning a single weight. Let $v_1, v_2 \in \mathbb{R}^d$ be the basis vectors of the left null-space. Then, it is possible to find $\gamma_1, \gamma_2 \in \mathbb{R}$ such that the system of equations $\gamma_1 \times v_{1,i} + \gamma_2 \times v_{2,i} = \mathbf{w}_i - \alpha_i$ and $\gamma_1 \times v_{1,j} + \gamma_2 \times v_{2,j} = \mathbf{w}_j - \alpha_j$ is satisfied for any two arbitrary indices $i, j \in \{1, .., d\}$, assuming that the determinant of $\begin{bmatrix} v_{1,i} & v_{2,i} \\ v_{1,j} & v_{2,j} \end{bmatrix}$ is nonzero. In particular, we can set $i = c$, such that $\alpha_c = \mathbf{w}_c = \mathbf{w}_j = 0$ and the system still maintains a solution. This implies that we have a valid solution $\|\mathbf{w}\|_0 = 3$ which is sparser than the ground truth solution $\|\alpha\|_0 = 4$. A simple generalization states that a (n, d) over-parameterized ($d > n$) sparse linear regression problem has at least a $d - n$-dimensional left null-space, which contains at most $\binom{d}{d-n}$ number of $n-$sparse solution vectors.

Thus when the problem is severely over-parameterized, we can always find n-sparse solutions $\mathbf{w}$ irrespective of the sparsity of the ground truth solution α. If $\|\alpha\|_0 > n$, then sparse recovery is not possible and we are able to prune more weights (n) than the solution α requires. This impossibility of sparse recovery renders PGD ineffective, making it no more effective than simple one-shot pruning.

To verify this, we run simulations on problems with different number of samples n and different choices for α, chosen either randomly or adversarially. The results of this simulation are given in Table 1. Note that in all cases we perform either one-shot pruning or PGD on the regularized least squares solution. We observe that one-shot pruning recovers the optimal solution at the same rate as PGD for the severely over-parameterized case, but PGD outperforms one-shot pruning in other cases. We attribute the less than perfect solution recovery of PGD to the non-adherance to the regularity conditions for both PGD and gaurantees for gaussian matrices to be RIP [1].

[1]Specifically, we do not ensure the RIP with an order $> 3(d-1)$ is maintained, as this would not allow for $(d-1)$-sparsity that applies to

Summarizing this section, we first find that one-shot pruning can fail for linear models, while PGD is more likely to converge to the correct solution, both in theory and practice. We next find that when the problem is severely over-parameterized, we can prune more weights than the solution requires and thus sparse recovery is not possible, which renders PGD no more effective than one-shot pruning. We stress here that the analysis done here is limited to the case of linear models, as notions such as RIP do not apply to non-linear regression using deep neural networks. However as we shall see in the next section, this behaviour of one-shot pruning and PGD on linear models carries over to their respective proxies, i.e., gradual and cyclical pruning applied to deep neural networks trained on large image datasets. This is because the underlying principle is same in both cases, i.e., when the decision of which weight to prune is not important, then weight recovery does not help.

5. Experiments

In this section, we show detailed experimental results and ablation studies examining various aspects of cyclical pruning. Our experiments are done using the Pytorch framework [33], and are organized as follows. In § 5.1, we compare cyclic pruning with state-of-the-art pruning algorithms. Here, we show results on CIFAR10 [23] and Imagenet [8] datasets across various models. In § 5.2, we perform controlled ablation experiments to study the impact of the sparsity and learning rate schedules, and the behaviour of the algorithm across different cycles.

5.1. Comparison with Gradual Pruning

Here we shall compare cyclical pruning with two baselines: one-shot pruning and gradual pruning. These are in accordance with the best practices suggested by [3]. In particular, we consider overall 8 architecture-dataset pairs with modern architectures, we report values along the trade-off curve for all methods, we compare different methods using an identical model, library and optimizer setup, and we report standard deviations whenever possible. We do not report explicit compression ratio and speedup as we use local layerwise sparsity, and hence these numbers are identical across different methods for the same sparsity ratio.

We first discuss results on the CIFAR10 dataset across two models. We present results at sparsity ratios from 90% to 99%, for Resnet56, and 70% to 95% for Mobilenet owing to the compact nature of this model. For rigorous comparisons, we perform pruning from the same baseline model in all cases, and allow each method the same amount of computation. Specifically, we train for 100 epochs for one-shot pruning and gradual pruning, and use 20 epochs with 5 cycles for cyclical pruning. We use SGD with momentum as

Observation 1

Table 2. Accuracy (%) after pruning various models on the CIFAR-10 dataset, with gradual pruning and one-shot pruning run for 100 epochs, and cyclical pruning for 5 cycles of 20 epochs each. We observe that cyclical pruning offers an advantage primarily at larger sparsity values, while being competitive at smaller values, in accordance with the theory in §4.

Model	Baseline	Methods			Pruning ratio
		One-Shot Pruning [15]	Gradual Pruning [37]	Cyclical Pruning (Ours)	
ResNet-20	92.24	90.10 ± 0.2	**90.78** ± 0.4	**90.90** ± 0.1	90%
		86.52 ± 0.9	**89.14** ± 0.2	**89.29** ± 0.2	95%
		78.55 ± 0.2	83.44 ± 0.1	**85.79** ± 0.1	98%
		33.43 ± 0.0	50.77 ± 2.7	**66.04** ± 2.4	99%
ResNet-56	93.28	**92.35** ± 0.1	**92.44** ± 0.0	**92.41** ± 0.1	90%
		90.85 ± 0.0	**91.69** ± 0.1	**91.90** ± 0.2	95%
		79.22 ± 0.0	89.57 ± 0.1	**90.54** ± 0.0	98%
		58.03 ± 0.3	68.20 ± 0.1	**70.99** ± 0.3	99%
VGG-14	93.57	92.98 ± 0.1	**93.25** ± 0.1	93.03 ± 0.1	90%
		92.17 ± 0.1	**92.36** ± 0.2	**92.47** ± 0.0	95%
		89.29 ± 0.1	90.64 ± 0.3	**91.71** ± 0.1	98%
		85.75 ± 0.0	88.59 ± 0.0	**89.59** ± 0.7	99%
Mobilenet	89.75	**90.22** ± 0.2	**90.25** ± 0.0	89.83 ± 0.1	70%
		88.44 ± 0.0	**89.49** ± 0.2	**89.37** ± 0.0	80%
		84.99 ± 0.3	85.51 ± 0.7	**86.99** ± 0.3	90%
		75.05 ± 1.0	73.42 ± 0.0	**79.07** ± 0.6	95%

our optimizer, and use the same learning rate schedules in all cases within a single cycle, i.e., we start fine-tuning with a learning rate of $1e-2$ and drop it to $1e-3$ after completing 75% of the allocated epochs, and use a batch size of 256. For the cyclical sparsity, the allocated epochs corresponds to the number of epochs for a single cycle, in this case being 20. Our experimental results in Table 2 shows that cyclical pruning outperforms gradual pruning, especially at high sparsity ratios. This aligns perfectly with the observations made for pruning of linear models, where PGD showed no benefits for the case of severe over-parameterization, which in this case corresponds to pruning with smaller sparsity ratios.

We also show experimental results on Imagenet in Table 3, where we show results on four pre-trained models with varying sparsity levels. In this case, for one-shot pruning and gradual pruning we allow 60 epochs of training, and use 20 epochs with 3 cycles for cyclical pruning. We use an exponential learning rate schedule. The learning rate is always decreased at the halfway mark, i.e., by a factor of 10 every 10 epochs for the cyclical pruning and every 30 epochs for gradual pruning. We use starting learning rate $1e-4$ and Adam optimizer [22], with a batch size of 64 for all the experiments in Table 3. The experiments show that cyclical pruning outperforms gradual pruning for higher compression ratios for all the models. This further supports the observations made earlier for pruning of linear models.

In addition to this, we also make informal comparisons with other reported results in literature in Table 4. Note that comparisons with reported results are not recommended practice [3], and this only provides an approximate indication of the relative performance of different methods. Furthermore, other methods in literature are trained sparse networks from scratch, whereas we prune pre-trained models. Although it is possible to apply our method to train from scratch as well, we do not do this here due to lack of resources to tune hyper-parameters for full Imagenet training. Also here we only make comparisons with methods which use local sparsity, similar to us. For instance, [10, 13] also provide pruning results with global sparsity, but we do not compare against those.[2]. The highest claimed performance in literature that we are aware of is by [10] who run 500 epochs of training on Imagenet, which is several times larger than our computational budget. For cyclical pruning of Resnet 50 in Table 4 we use SGD with momentum with a learning rate of $1e-2$ and momentum 0.9, instead of Adam which was used to obtain results of Table 3.

5.2. Ablation Experiments

Here we perform controlled experiments to understand the behaviour of cyclical pruning across successive cycles. First, we consider the effect of the pruning schedule within a single cycle by comparing the evolution of the model upon

[2]For results of [13] see: https://bit.ly/39KSC6Z

Table 3. Accuracy (%) after pruning various models on the Imagenet dataset, with gradual pruning and one-shot pruning run for 60 epochs, and cyclical pruning for 3 cycles of 20 epochs each. We observe that cyclical pruning offers an advantage primarily at larger sparsity values, while being competitive at smaller values, in accordance with the theory in §4.

		Methods			
Model	Baseline	One-Shot Pruning [15]	Gradual Pruning [37]	Cyclical Pruning (Ours)	Pruning ratio
ResNet18	69.7	**69.9**	**69.9**	69.6	60%
		69.2	69.2	**69.4**	70%
		68.2	67.8	**68.3**	80%
		63.5	63.6	**64.9**	90%
ResNet50	76.16	75.9	**76.1**	75.8	60%
		75.9	75.8	75.7	70%
		75.4	74.9	**75.3**	80%
		72.8	71.9	**73.3**	90%
		67.1	64.7	**68.7**	95%
EfficientNet	74.8	73.9	74.0	**74.1**	40%
		73.2	73.2	**73.4**	50%
		71.2	71.8	**72.4**	60%
		68.0	68.2	**69.9**	70%
		65.1	65.2	**67.5**	75%
MobilenetV2	71.7	70.8	**70.9**	69.8	40%
		67.6	69.8	**70.1**	50%
		66.7	67.6	**68.4**	60%
		61.3	62.7	**64.4**	70%

Table 4. Informal comparison of cyclical pruning with published results on Resnet50 trained on Imagenet. We only compare with methods that use local sparsity. For cyclical pruning, we start from a dense pre-trained ResNet-50 trained for 90 epochs, and use a cycle length of 20 epochs for pruning. Thus the total number of epochs corresponds to 130 epochs = 90 + 2 cycles $\times$ 20 epochs, and similarly for 110 & 150 epochs. Note that 110 epochs of cyclical pruning corresponds to one cycle, thus being identical to gradual pruning. Other methods in literature train from scratch.

Method	Pruning ratio	Dense Baseline	Pruned	Difference
SNFS (100 epochs) [9]	72.4%	75.95	74.59	-1.36
DPF (90 epochs) [28]	73.5%	75.95	75.48	-0.47
Cyclical Pruning (110 epochs, Ours)	73.5%	76.16	75.46	-0.7
Cyclical Pruning (130 epochs, Ours)	73.5%	76.16	75.84	**-0.32**
Cyclical Pruning (150 epochs, Ours)	73.5%	76.16	76.00	**-0.15**
SNFS (100 epochs) [9]	82.0%	75.95	72.65	-3.30
RigL (100 epochs) [10]	80.0%	76.80	74.60	-2.2
RigL (500 epochs) [10]	80.0%	76.80	76.60	**-0.2**
DPF (90 epochs) [28]	82.6%	75.95	74.55	-1.44
Gradual Pruning (100 epochs) [13]	80.0%	76.69	75.58	-1.11
Cyclical Pruning (110 epochs, Ours)	82.6%	76.16	74.65	-1.51
Cyclical Pruning (130 epochs, Ours)	82.6%	76.16	75.29	**-0.87**
Cyclical Pruning (150 epochs, Ours)	82.6%	76.16	75.40	**-0.75**

using a cubic schedule, with that of linear and step schedules. Here, step schedule corresponds to one-shot pruning, which can be thought of as using heaviside step function for the sparsity schedule. In all cases, each cycle consists of 20 epochs, and within each cycle, 16 epochs are used for alternating pruning and re-training, and the final 4 epochs con-

Table 5. Comparison of different per-cycle pruning schedules used with cyclical pruning, on Resnet20 / CIFAR10 @ 99% sparsity. 'Mask distance' refers to a Jaccard distance computed between a given mask and the mask obtained after the first cycle. We observe that while cubic schedule performs the best, we observe that in all cases the accuracy increases and the pruning mask changes every cycle. This provides evidence for weight recovery in cyclical pruning. We also observe that simply training longer using cyclical learning rates is insufficient.

	Cycle #	1	2	3	4	5
Cubic schedule	Accuracy (%)	62.74 $\pm$ 0.2	65.89 $\pm$ 0.6	66.89 $\pm$ 0.7	67.23 $\pm$ 0.6	67.56 $\pm$ 0.7
	Mask distance	0	0.38	0.49	0.55	0.58
Linear schedule	Accuracy (%)	47.63 $\pm$ 1.2	52.035 $\pm$ 0.9	56.33 $\pm$ 0.9	57.65 $\pm$ 1.2	58.27 $\pm$ 1.2
	Mask distance	0	0.48	0.58	0.62	0.64
Step schedule	Accuracy (%)	39.88 $\pm$ 0.5	47.47 $\pm$ 0.4	50.94 $\pm$ 0.4	53.64 $\pm$ 0.2	55.36 $\pm$ 0.6
	Mask distance	0	0.46	0.59	0.65	0.68
Finetune with cyclical learning rates	Accuracy (%)	63.01 $\pm$ 0.0	63.73 $\pm$ 0.1	64.11 $\pm$ 0.2	63.8 $\pm$ 0.2	64.06 $\pm$ 0.2
	Mask distance	0	0	0	0	0

sist of purely fine-tuning of the sparse model. For step pruning, we prune at the halfway mark, i.e., at 10 epochs. The results shown in Table 5 indicate that regardless of the sparsity schedule, cyclical pruning always shows accuracy improvements over successive cycles, with cubic pruning performing the best overall. Further, we also compute the Jaccard distance of pruning masks obtained at the end of cycles to understand the extend of mask evolution. Table 5 also shows that the mask changes drastically across cycles in all cases, thus confirming our hypothesis that cyclical pruning allows for correction of erroneously pruned weights.

Second, we decouple the effect of cyclical pruning with cyclical learning rates and show that the improvement across cycles is precisely due to the pruning schedule and not due the learning rate schedule. To test this, we run a control experiment referred to as 'Finetune with cyclical learning rates' in Table 5, where the first cycle is identical to cyclical pruning, and in the subsequent cycles only fine-tuning is performed for the obtained sparse model without additional pruning. For this fine-tuning, we maintain the learning rate schedule used for cyclical sparsity. We observe that increasing the number of fine-tuning epochs does lead to improved accuracy across cycles as expected, but not as much as that obtained for cyclical sparsity. As expected, fine-tuning also cannot allow any changes in the pruning mask which leads to zero Jaccard distances.

Overall, the experiments in Table 5 here show that, (1) Accuracy improves across rounds in cyclical pruning regardless of the sparsity schedule, and weight recovery indeed takes place as indicated by the Jaccard distances. (2) Cubic pruning outperforms linear and one-shot (step) pruning. (3) Improved performance of cyclical pruning is **not** explained by longer training, as shown by the comparison with cyclical learning rates.

6. Discussion

In this work, we introduce *cyclical pruning*, a simple strategy that allows for recovery of erroneously pruned weights, leading to an improved sparsity-accuracy trade-off across various datasets and models. The cyclical paradigm can be used in conjunction with any per-cycle sparsity schedule. In addition, the cycle-wise accuracy improvements also show that cyclical sparsity can also be used to improve performance of existing sparse models obtained via any other method. Our theory and experiments reveal that cyclical pruning offers an advantage primarily for pruning with large sparsity ratios, when the solution space is not degenerate and the choice of weights to prune is critical. This method introduces only two additional hyperparameters, the number of cycles k and the initial sparsity s_{init} for subsequent cycles. We found that it is generally beneficial to keep the number of cycles, and the number of epochs per cycle to be as large as possible within the computational budget. These indicate that the method is also easy to tune.

Future work involves understanding why cubic pruning works well, and whether it is possible to use cyclical pruning in a more efficient manner that decouples its dependence on cubic pruning. Our unified view of pruning methods via TV-PGD also leads to a natural open problem: how do we optimally set $(s(t_i), \eta(t_i), \Delta t)$ in TV-PGD to guarantee convergence for neural network pruning?

References

[1] Kambiz Azarian, Yash Bhalgat, Jinwon Lee, and Tijmen Blankevoort. Learned threshold pruning. *arXiv preprint arXiv:2003.00075*, 2020. 2

[2] Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 28(3):253–263, 2008. 4

[3] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020. 1, 5, 6

[4] T Blumensath and ME Davies. Iterative hard thresholding for compressed sensing. 2008. 4

[5] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006. 4

[6] Jack Choquette and Wish Gandhi. Nvidia a100 gpu: Performance & innovation for gpu computing. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–43. IEEE Computer Society, 2020. 1

[7] Bin Dai, Chen Zhu, and David Wipf. Compressing neural networks using the variational information bottleneck. *arXiv preprint arXiv:1802.10399*, 2018. 2

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 5

[9] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019. 2, 7

[10] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020. 2, 6, 7

[11] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 2

[12] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 8, 2019. 2

[13] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019. 1, 2, 6, 7

[14] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in Neural Information Processing Systems*, 2016. 1, 2

[15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28:1135–1143, 2015. 1, 2, 6, 7

[16] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993. 2

[17] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017. 1

[18] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. 1

[19] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 1

[20] Prateek Jain and Purushottam Kar. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4):142–363, 2017. 4

[21] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20744–20754. Curran Associates, Inc., 2020. 2

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[23] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 5

[24] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the International Conference on Machine Learning*, July 2020. 2

[25] César Laurent, Camille Ballas, Thomas George, Pascal Vincent, and Nicolas Ballas. Revisiting loss modelling for unstructured pruning, 2021. 2

[26] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 2

[27] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations (ICLR)*, 2017. 1

[28] Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020. 2, 7

[29] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3288–3298. Curran Associates, Inc., 2017. 2

[30] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *International Conference on Learning Representations*, 2018. 2

[31] Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995. 4

[32] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6775–6784. Curran Associates, Inc., 2017. 2

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H Wallach, H Larochelle, A Beygelzimer, F d\textquotesingle Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037. Curran Associates, Inc., 2019. 5

[34] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. *Advances in Neural Information Processing Systems*, 2020. 2

[35] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018. 2

[36] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. In *Advances in Neural Information Processing Systems*, 2019. 2

[37] Michael H. Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. 2018. 1, 2, 6, 7