AC/DC: Alternating Compressed/DeCompressed Training of Deep Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

The increasing computational requirements of deep neural networks (DNNs) have led to significant interest in obtaining DNN models that are *sparse*, yet *accurate*. Recent work has investigated the even harder case of *sparse training*, where the DNN weights are, for as much as possible, already sparse to reduce computational costs during training. Existing sparse training methods are mainly empirical and often have lower accuracy relative to the dense baseline. In this paper, we present a general approach called Alternating Compressed/DeCompressed (AC/DC) training of DNNs, demonstrate convergence for a variant of the algorithm, and show that AC/DC outperforms existing sparse training methods in accuracy at similar computational budgets; at high sparsity levels, AC/DC even outperforms existing methods that rely on accurate pre-trained dense models. An important property of AC/DC is that it allows *co-training* of dense and sparse models, yielding accurate *sparse–dense model pairs* at the end of the training process. This is useful in practice, where compressed variants may be desirable for deployment in resource-constrained settings without re-doing the entire training flow, and also provides us with insights into the accuracy gap between dense and compressed models.

1 Introduction

The tremendous progress made by deep neural networks in solving diverse tasks has driven significant research and industry interest in deploying efficient versions of these models. To this end, entire families of model compression methods have been developed, such as pruning [29] and quantization [22], which are now accompanied by hardware and software support [54, 8, 11, 43, 23].

Neural network pruning, which is the focus of this paper, is the compression method with arguably the longest history [38]. The basic goal of pruning is to obtain neural networks for which many connections are removed by being set to zero, while maintaining the network's accuracy. A myriad pruning methods have been proposed—please see [29] for an in-depth survey—and it is currently understood that many popular networks can be compressed by more than an order of magnitude without significant accuracy loss.

Many accurate pruning methods require a fully-accurate, dense variant of the model, from which weights are subsequently removed. A shortcoming of this approach is the fact that the memory and computational savings due to compression are only available for the *inference*, post-training phase, and not during training itself. This distinction becomes important especially for large-scale modern models, which can have millions or even billions of parameters, and for which fully-dense training can have high computational and even non-trivial environmental costs [52].

One approach to address this issue is *sparse training*, which essentially aims to remove connections from the neural network as early as possible during training, while still matching, or at least approximating, the accuracy of the fully-dense model. For example, the RigL technique [16] randomly removes a large fraction of connections early in training, and then proceeds to optimize over the sparse support, providing savings due to sparse back-propagation. Periodically, the method re-introduces some of the weights during the training process, based on a combination of heuristics, which requires

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

taking full gradients. These works, as well as many recent sparse training approaches [4, 44, 32], which we cover in detail in the next section, have shown empirically that non-trivial computational savings, usually measured in theoretical FLOPs, can be obtained using sparse training, and that the optimization process can be fairly robust to sparsification of the support.

At the same time, this line of work still leaves intriguing open questions. The first is *theoretical*: to our knowledge, none of the methods optimizing over sparse support, and hence providing training speed-up, have been shown to have convergence guarantees. The second is *practical*, and concerns a deeper understanding of the relationship between the densely-trained model, and the sparsely-trained one. Specifically, (1) current sparse training methods can still leave a non-negligible accuracy gap, relative to dense training, or even post-training sparsification; and (2) most existing work on sparsity requires significant changes to the training flow, and focuses on maximizing global accuracy metrics; thus, we lack understanding when it comes to *co-training* sparse and dense models, as well as with respect to correlations between sparse and dense models *at the level of individual predictions*.

Contributions. In this paper, we take a step towards addressing these questions. We investigate a general approach for sparse training of neural networks, which we call *Alternating Compressed / DeCompressed (AC/DC)* training. AC/DC performs *co-training of sparse and dense models*, and can return both an accurate *sparse* model, and a *dense* model, which can recover the dense baseline accuracy via fine-tuning. We show that a variant of AC/DC ensures convergence for smooth, non-convex objectives, under analytic assumptions. Extensive experimental results show that it provides state-of-the-art accuracy among sparse training techniques at comparable training budgets, and can even outperform *post-training* sparsification approaches.

AC/DC builds on the classic *iterative hard thresholding (IHT)* family of methods for sparse recovery [6]. As the name suggests, AC/DC works by alternating the standard *dense* training phases, with *sparse* phases where optimization is performed exclusively over a fixed *sparse support*, and a subset of the weights and their gradients are fixed at zero, leading to computational savings. (This is in contrast to *error feedback* algorithms, e.g. [9, 40] which require fully-dense gradients, even though the weights may be sparse.) The process uses the same number of epochs and hyper-parameters as regular training, and the frequency and length of the phases can be safely set to standard values, e.g. 5–10 epochs. We ensure that training ends on a *sparse* phase, and return the resulting *sparse* model, as well as the last *dense* model obtained at the end of a *dense* phase. This dense model may be additionally fine-tuned for a short period, leading to a more accurate *dense-finetuned* model, which we usually find to match the accuracy of the *dense baseline*.

We emphasize that algorithms alternating sparse and dense training phases for deep neural networks have been previously investigated [33, 25], but with the different goal on using sparsity as a regularizer to obtain *more accurate dense models*. Relative to these works, our goal is to produce highly-accurate, highly-sparse models, as well as maximizing the fraction of training time for which optimization is performed over a sparse support, leading to computational savings. Also, we are the first to provide convergence guarantees for variants of this approach.

We perform an extensive empirical investigation of this approach, showing that it provides consistently good results on a wide range of models and tasks (ResNet [28] and MobileNets [30] on the ImageNet [48] / CIFAR [36] datasets, and Transformers [55, 10] on WikiText [42]), under standard values of the training hyper-parameters. Specifically, when executed on the same number of training epochs, our method outperforms all previous *sparse training* methods in terms of the accuracy of the resulting sparse model, often by significant margins. This comes at the cost of slightly higher theoretical computational cost relative to prior sparse training methods, although AC/DC usually reduces training FLOPs to 45–65% of the dense baseline. AC/DC is also close to the accuracy of state-of-the-art post-training pruning methods [37, 51] at medium sparsities (80% and 90%); surprisingly, it *significantly outperforms* them at higher sparsities. In addition, we showcase the flexibility of AC/DC by employing it to obtain *semi-structured* pruned models using the 2:4 sparsity pattern efficiently supported by new NVIDIA hardware [43]. Further, we show that the resulting sparse models can provide real-world speedups for DNN inference on CPUs [12].

An interesting feature of AC/DC is that it allows for accurate dense/sparse co-training of models. Specifically, at medium sparsity levels (80% and 90%), the method allows the co-trained dense model to recover the dense baseline accuracy via a short fine-tuning period. In addition, dense/sparse co-training provides us with a lens into the training dynamics, in particular relative to the sample-level accuracy of the two models, but also in terms of the dynamics of the sparsity masks. Specifically, we observe that co-trained sparse/dense pairs have higher sample-level agreement than sparse/dense pairs obtained via post-training pruning, and that weight masks still change later in training.

Additionally, we probe the accuracy differences between sparse and dense models, by examining their "memorization" capacity [59]. For this, we perform dense/sparse co-training in a setting where a small number of valid training samples have *corrupted labels*, and examine how these samples are classified during dense and sparse phases, respectively. We observe that the sparse model is less able to "memorize" the corrupted labels, and instead often classifies the corrupted samples to their *true* (correct) class. By contrast, during dense phases model can easily "memorize" the corrupted labels. (Please see Figure 2b for an illustration.) This suggests that one reason for the higher accuracy of dense models is their ability to "memorize" hard-to-classify samples.

2 Related Work

There has recently been tremendous research interest into pruning techniques for DNNs; we direct the reader to the recent surveys of [21] and [29] for a comprehensive overview. Roughly, most DNN pruning methods can be split as (1) *post-training* pruning methods, which start from an accurate dense baseline, and remove weights, followed by fine-tuning; and (2) *sparse training* methods, which perform weight removal during the training process itself. (Other categories such as *data-free* pruning methods [39, 53] exist, but they are beyond our scope.) We focus on *sparse training*, although we will also compare against state-of-the-art post-training methods. Another perspective is *computational*: *post-training* methods produce efficient model *inference models*, but usually still compute dense gradients, while *sparse training* methods aim to also perform *sparse backpropagation*, leading to computational gains.

Arguably, the most popular metric for weight removal is *weight magnitude* [24, 26, 61]. Better-performing approaches exist, such as second-order metrics [38, 27, 14, 51], or Bayesian approaches [45], but these tend to have higher computational and implementation cost.

The general goal of *sparse training* methods is to perform both the forward (inference) pass *and the backpropagation* pass over a sparse support, leading to computational gains during the training process as well. One of the first approaches to maintain sparsity throughout training was Deep Rewiring [4], where SGD steps applied to positive weights are augmented with random walks in parameter space, followed by inactivating negative weights. To maintain sparsity throughout training, randomly chosen inactive connections are re-introduced in the "growth" phase. Sparse Evolutionary Training (SET) [44] introduces a non-uniform sparsity distribution across layers, which scales with the number of input and output channels, and trains sparse networks by pruning weights with smallest magnitude and re-introducing some weights randomly. RigL [16], prunes weights at random after a warm-up period, and then periodically performs weight re-introduction using a combination of connectivity- and gradient-based statistics, which require periodically evaluating full gradients. RigL can lead to state-of-the-art accuracy results even compared to post-training methods; however, for this it requires significant additional data passes (e.g. 5x) relative to the dense baseline. Most recently, Top-KAST [32] alleviated the drawback of periodically having to evaluate dense gradients, by updating the sparsity masks using gradients *of reduced sparsity* relative to the weight sparsity. This allows some of the pruned weights to be re-introduced during training, improving accuracy. The latter two methods set the state-of-the-art for sparse training: when executing for the same number of epochs as the dense baseline, they provide computational reductions the order of 2x, while the accuracy of the resulting sparse models is lower than that of leading post-training methods, executed at the same sparsity levels. To our knowledge, all these methods are heuristics, i.e. do not have convergence guarantees.

Another approach towards pruning with an intermediate computational cost is *training sparse networks from scratch*. The masks are updated by continuously pruning and re-introducing weights. For example, [40] uses magnitude pruning after applying SGD on the dense network, whereas [13] update the masks by re-introducing weights with the highest gradient momentum. STR [37] learns a separate pruning threshold for each layer and allows sparsity both during forward and backward passes; however, the desired sparsity can not be explicitly imposed, and the network has low sparsity for a large portion of training. These methods can lead to only limited computational gains, since they either require dense gradients, or the sparsity level cannot be imposed. By comparison, our method provides models of similar or better accuracy at the same sparsity, with computational reductions. We also obtain dense models that match the baseline accuracy, with a fraction of the baseline FLOPS.

The idea of alternating sparse and dense training phases has been examined before in the context of neural networks, but with the goal of using temporary sparsification as a regularizer. Specifically, *Dense-Sparse-Dense (DSD)* [25] proposes to first *train a dense model to full accuracy*; this model is then sparsified via magnitude; next, optimization is performed over the sparse support, followed

by an additional optimization phase over the full dense support. Thus, this process is used as a regularization mechanism for the dense model, which results in relatively small, but consistent accuracy improvements relative to the original dense model. In [33], the authors propose a similar approach to DSD, but alternate sparse phases during the regular training process. The resulting process is similar to AC/DC, but, importantly, the goal of their procedure is to return a *more accurate dense model*. (Please see their Algorithm 1.) For this, the authors use relatively low sparsity levels, and gradually increase sparsity during optimization; they observe accuracy improvements for the resulting dense models, at the cost of increasing the total number of epochs of training. By contrast, our focus is on obtaining accurate *sparse* models, while reducing computational cost, and executing the dense training recipe. We execute at higher sparsity levels, and on larger-scale datasets and models. In addition, we also show that the method works for other sparsity patterns, e.g. the 2:4 semi-structured pattern [43].

More broadly, the Lottery Ticket Hypothesis (LTH) [19], states that sparse networks can be trained in isolation *from scratch* to the same performance as a post-training pruning baseline, by starting from the "right" weight and sparsity mask initializations, optimizing only over this sparse support. However, initializations usually require the availability of the fully-trained dense model, falling under *post-training* methods. There is still active research on replicating these intriguing findings to large-scale models and datasets [21, 20].

3 Alternating Compressed / DeCompressed (AC/DC) Training

3.1 Background and Assumptions

Obtaining *sparse* solutions to optimization problems is a problem of interest in several areas [7, 6, 17], where the goal is to minimize a function $f : \mathbb{R}^N \to \mathbb{R}$ under sparsity constraints:

$$\min_{\theta \in \mathbb{R}^N} f(\theta) \quad \text{s.t.} \quad \|\theta\|_0 \leq k\,. \tag{1}$$

For the case of ℓ_2 regression, $f(\theta) = \|b - A\theta\|_2^2$, a solution has been provided by Blumensath and Davies [6], known as the *Iterative Hard Thresholding (IHT)* algorithm, and subsequent work [17, 18, 57] provided theoretical guarantees for the linear operators used in compressed sensing. The idea consists of alternating gradient descent (GD) steps and applications of a thresholding operator to ensure the ℓ_0 constraint is satisfied. More precisely, T_k is defined as the "top-k" operator, which keeps the largest k entries of a vector θ in absolute value, and replaces the rest with 0. The IHT update at step $t+1$ has the following form:

$$\theta_{t+1} = T_k(\theta_t - \eta \nabla f(\theta_t)). \tag{2}$$

Most convergence results for IHT assume deterministic gradient descent steps. For DNNs, stochastic methods are preferred, so we describe and analyze a stochastic version of IHT and prove its convergence.

Stochastic IHT. We consider functions $f : \mathbb{R}^N \to \mathbb{R}$, for which we can compute stochastic gradients g_θ, which are unbiased estimators of the true gradient $\nabla f(\theta)$. Define the *stochastic* IHT update as:

$$\theta_{t+1} = T_k(\theta_t - \eta g_{\theta_t}). \tag{3}$$

This formulation covers the practical case where the stochastic gradient g_θ corresponds to a *mini-batch* stochastic gradient. Indeed, as in practice f takes the form $f(\theta) = \frac{1}{m}\sum_{i=1}^m f(\theta; x_i)$, where $S = \{x_1, \ldots, x_m\}$ are data samples, the stochastic gradients obtained via backpropagation take the form $\frac{1}{|B|}\sum_{i \in B} \nabla f(\theta; x_i)$, where B is a sampled mini-batch. We aim to prove strong convergence bounds for stochastic IHT, under common assumptions that arise in the context of training DNNs.

Analytical Assumptions. Formally, our analysis uses the following assumptions on f.

1. Unbiased gradients with variance σ: $\mathbb{E}[g_\theta|\theta] = \nabla f(\theta)$, and $\mathbb{E}[\|g_\theta - \nabla f(\theta)\|^2] \leq \sigma^2$.
2. Existence of a k^*-sparse minimizer θ^*: $\exists \theta^* \in \arg\min_\theta f(\theta)$, s.t. $\|\theta^*\|_0 \leq k^*$.
3. For $\beta > 0$, the β-smoothness condition when restricted to t coordinates ((t, β)-smoothness):

$$f(\theta + \delta) \leq f(\theta) + \nabla f(\theta)^\top \delta + \frac{\beta}{2}\|\delta\|^2, \text{ for all } \theta, \delta \text{ s.t. } \|\delta\|_0 \leq t\,. \tag{4}$$

4. For $\alpha > 0$ and number of indices r, the *r-concentrated Polyak-Łojasiewicz ((r, α)-CPL) condition*:

$$\|T_r(\nabla f(\theta))\| \geq \frac{\alpha}{2}(f(\theta) - f(\theta^*))\,, \text{ for all } \theta. \tag{5}$$

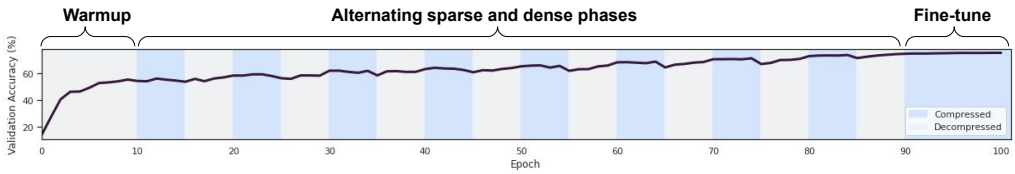


Figure 1: The AC/DC training process. After a short warmup we alternatively prune to maximum sparsity and restore the pruned weights. The plot shows the sparsity and validation accuracy throughout the process for a sample run on ResNet50/ImageNet at 90% sparsity.

The first assumption is standard in stochastic optimization, while the existence of very sparse minimizers is a known property in over-parametrized DNNs [19], and is the very premise of our study. Smoothness is also a standard assumption, e.g. [40]—we only require it along *sparse* directions, which is a strictly weaker assumption. The more interesting requirement for our convergence proof is the (r, α)-CPL condition in Equation (5).

The standard Polyak-Łojasiewicz (PL) condition [34] is common in non-convex optimization, and versions of it are essential in the analysis of DNN training [41, 2]. Its standard form states that small gradient norm, i.e. approximate stationarity, implies closeness to optimum in function value. We require a slightly stronger version, in terms of the norm of the gradient contributed by its largest coordinates in absolute value. This restriction appears necessary for the success of IHT methods, as the sparsity enforced by the truncation step automatically reduces the progress ensured by a gradient step to an amount proportional to the norm of the top-k gradient entries. This strengthening of the PL condition is supported both theoretically, by the mean-field view, which argues that gradients are sub-gaussian [49], and by empirical validations of this behaviour [1, 50].

We are now ready to state our main analytical result.

Theorem 1. *Let $f : \mathbb{R}^N \to \mathbb{R}$ be a function with a k^*-sparse minimizer θ^*. Let $\beta > \alpha > 0$ be parameters, let $k = C \cdot k^* \cdot (\beta/\alpha)^2$ for some appropriately chosen constant C, and suppose that f is $(2k + 3k^*, \beta)$-smooth and (k^*, α)-CPL. For initial parameters θ_0 and precision $\epsilon > 0$, given access to stochastic gradients with variance σ, stochastic IHT (3) converges in $O\left(\frac{\beta}{\alpha} \cdot \ln \frac{f(\theta_0) - f(\theta^*)}{\epsilon}\right)$ iterations to a point θ with $\|\theta\|_0 \leq k$, such that*

$$\mathbb{E}\left[f(\theta) - f(\theta^*)\right] \leq \epsilon + \frac{16\sigma^2}{\alpha}.$$

Roughly, this result suggests that, for a given set of parameters, by increasing the support demanded from our approximate minimizer θ relative to the optimal k^*, the parameter α increases, which leads to lower loss, and potentially to faster running time. We provide a complete proof of this result in the Appendix. Our analysis approach also works in the absence of the CPL condition (Theorem 3), in which case we prove that a version of the algorithm can find sparse nearly-stationary points. As a bonus, we also simplify existing analyses for IHT and extend them to the stochastic case (Theorem 2). Another interpretation of our results is in showing that, under our assumptions, *error feedback* [40] is not necessary for recovering good sparse minimizers; this has practical implications, as it allows us to perform fully-sparse back-propagation in sparse optimization phases. Next, we discuss our practical implementation, and its connection to these theoretical results.

3.2 AC/DC: Applying IHT to Deep Neural Networks

AC/DC starts from a standard DNN training flow, using standard optimizers such as SGD with momentum [47] or Adam [35] for a fixed number of epochs. It preserves the standard training hyper-parameters, e.g., learning rate, momentum, weight decay, and will only periodically modify the *support* for optimization. Please see Algorithm 1 for pseudocode.

We partition the set of training epochs into *compressed* epochs $\mathcal{C}$, and *decompressed* epochs $\mathcal{D}$. We begin with a *dense warm-up* period of Δ_w consecutive epochs, during which regular dense (decompressed) training is performed. We then start alternating *compressed optimization* phases of length Δ_c epochs each, with *decompressed (regular) optimization* phases of length Δ_d epochs each. The process completes on a compressed fine-tuning phase, returning an accurate sparse model. Alternatively, if our goal is to return a dense model matching the baseline accuracy, we take the best dense checkpoint obtained during alternation, and fine-tune it over the entire support. In practice, we noticed that allowing a longer final decompressed phase of length $\Delta_D > \Delta_d$ improves

Algorithm 1 Alternating Compressed/Decompressed (AC/DC) Training

Require: Weights $\theta \in \mathbb{R}^N$, data S, sparsity k, compression phases $\mathcal{C}$, decompression phases $\mathcal{D}$

Train the weights θ for Δ_w epochs ▷ Warm-up phase
while epoch $\leq$ max epochs **do**
  if entered a compression phase **then**
    $\theta \leftarrow T_k(\theta, k)$ ▷ apply compression (top-k) operator on weights
    $m \leftarrow \mathbb{1}[\theta_i \neq 0]$ ▷ create masks
  end if
  if entered a decompression phase **then**
    $m \leftarrow \mathbb{1}_N$ ▷ reset all masks
  end if
  $\theta \leftarrow \theta \odot m$ ▷ apply the masks (ensure sparsity for compression phases)
  $\tilde{\theta} \leftarrow \{\theta_i | m_i \neq 0, 1 \leq i \leq N\}$ ▷ get the support for the gradients
  for x mini-batch in S **do**
    $\theta \leftarrow \theta - \eta \nabla_{\tilde{\theta}} f(\theta; x)$ ▷ optimize the active weights
  end for
  epoch $\leftarrow$ epoch $+1$
end while
return θ

the performance of the dense model, by allowing it to better recover the baseline accuracy after fine-tuning. Please see Figure 1 for an illustration of the schedule.

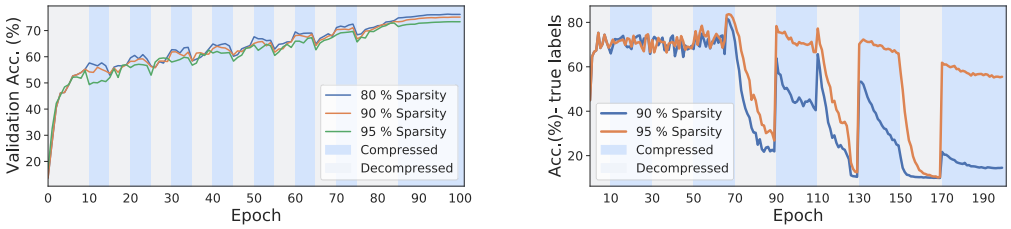
In our experiments, we focus on the case where the compression operation is unstructured or semi-structured pruning. In this case, at the beginning of each sparse optimization phase, we apply the top-k operator across all of the network weights to obtain a mask M over the weights θ. The top-k operator is applied globally across all of the network weights, and will represent the sparse support over which optimization will be performed for the rest of the current sparse phase. At the end of the sparse phase, the mask M is reset to all-1s, so that the subsequent dense phase will optimize over the full dense support. Furthermore, once all weights are re-introduced, it is beneficial to reset to 0 the gradient momentum term of the optimizer; this is particularly useful for the weights that were previously pruned, which would otherwise have stale versions of gradients.

Discussion. Moving from the theoretical algorithm to a robust practical implementation required some adjustments, so we now interpret AC/DC from the perspective of our analysis. First, each *decompressed phase* can be directly mapped to a *deterministic/stochastic IHT* step, where, instead of a single gradient step in between consecutive truncations of the support, we perform several stochastic steps. These additional steps improve the accuracy of the method in practice, and we can bound their influence in theory as well, although they do not necessarily provide better bounds. This leaves open the interpretation of the *compressed phases*: for this, notice that the core of the proof for Theorem 1 is in showing that a single IHT step significantly decreases the expected value of the objective; using a similar argument, we can prove that additional optimization steps over the sparse support can only improve convergence. We are able to show convergence for a variant of IHT closely following AC/DC, but the bounds do not improve over Theorem 1.

4 Experimental Validation

Goals and Setup. We tested AC/DC on image classification tasks (CIFAR-100 [36] and ImageNet [48]) and on language modelling tasks [42] using the Transformer-XL model [10]. The goal is to examine the *validation accuracy* of the resulting sparse and dense models, versus the induced sparsity, as well as the number of FLOPs used for training and inference, relative to other sparse training methods. Additionally, we compare against state-of-the-art post-training pruning methods [51, 37]. We also examine prediction differences between the sparse and dense models.

We use PyTorch [46] for our implementation, and Weights & Biases [5] for experimental tracking. Experiments were run on NVIDIA RTX 2080 GPUs for image classification tasks, and NVIDIA RTX 3090 GPUs for language modelling. Each ImageNet run took approximately 2 days for ResNet50 and one day for MobileNet, while each Transformer-XL experiment took approximately 2 days. All reported image classification experiments were performed in triplicate by varying the random seed; we report mean and standard deviation. Due to computational limitations, the language modelling experiments were conducted in a single run.



(a) Sparsity pattern and validation accuracy vs. number of epochs (ResNet50/ImageNet).

(b) Percentage of samples with corrupted training labels classified to their *true* class (ResNet20/CIFAR10).

Figure 2: Accuracy vs. sparsity during training, for the ResNet50/ImageNet experiment (left) and accuracy on the corrupted samples for ResNet20/CIFAR10, w.r.t. the *true* class (right).

ImageNet Experiments. On the ImageNet dataset [48], we test AC/DC on ResNet50 [28] and MobileNetV1 [30]. In all reported results, the models were trained for a fixed number of 100 epochs, using SGD with momentum. We use a cosine learning rate scheduler and training hyper-parameters following [37], but without label smoothing. Our dense baseline has 76.84% validation accuracy. We report the average and the standard deviation across 3 different seeds. Unless otherwise specified, weights are pruned globally, based on their magnitude and in a single step. Similar to previous work, we did not prune biases, nor the Batch Normalization parameters.

For all results, the AC/DC compression schedule starts with a "warm-up" phase of dense training for 10 epochs, after which we alternate between compression and de-compression every 5 epochs, until the last dense and sparse phase. It is beneficial to allow these last two "fine-tuning" phases to run longer: the last decompression phase runs for 10 epochs, whereas the final 15 epochs are the compression fine-tuning phase. We reset SGD momentum at the beginning of every decompression phase. In total, we have an equal number of epochs of dense and sparse training; see Figure (2a) for an illustration. We use exactly the same setup for both ResNet50 and MobileNetV1 models, which resulted in high-quality sparse models. To recover a dense model with baseline accuracy using AC/DC, we finetune the best dense checkpoint obtained during training; practically, this replaces the last *sparse* fine-tuning phase with a phase where the *dense* model is fine-tuned instead.

ResNet50 Results. Tables 1 & 2 contain validation accuracy results across medium and high global sparsity levels, as well as inference and training FLOPs. Overall, AC/DC achieves higher validation accuracy than any of the state-of-the-art sparse training methods, when using the same number of epochs. At the same time, due to dense training phases, AC/DC tends to have higher FLOP requirements relative to RigL or Top-KAST at the same sparsity.

Table 1: Results for medium sparsity on ResNet50.

Method	Sparsity (%)	Top-1 Acc. (%)	GFLOPs Inference	EFLOPs Train
Dense	0	76.84	8.2	3.15
AC/DC	79.83	76.3 ± 0.1	0.29×	0.65×
$\text{RigL}_{1\times}$(ERK)	80	75.1 ± 0.05	0.42×	0.42×
Top-KAST	80	75.03	0.23×	0.32×
STR	79.55	76.19	0.19×	-
WoodFisher	80	**76.76**	0.25×	-
AC/DC	89.8	75.03 ± 0.1	0.19×	0.58×
$\text{RigL}_{1\times}$ (ERK)	90	73.0 ± 0.04	0.25×	0.24×
Top-KAST	90	74.76	0.13×	-
STR	90.23	74.31	0.08×	-
WoodFisher	90	**75.21**	0.14×	-

Table 2: Results for high sparsity on ResNet50.

Method	Sparsity (%)	Top-1 Acc. (%)	GFLOPs Inference	EFLOPs Train
Dense	0	76.84	8.2	3.15
AC/DC	94.8	$\mathbf{73.14 \pm 0.2}$	0.12×	0.5×
$\text{RigL}_{1\times}$ (ERK)	95	69.7 ± 0.17	0.12×	0.42×
Top-KAST	95	70.42	0.08×	0.1×
STR	94.8	70.97	0.04×	-
WoodFisher	95	72.12	-	-
AC/DC	97.8	$\mathbf{68.44 \pm 0.09}$	0.06×	0.46×
Top-KAST	98	67.06	0.05×	0.08×
STR	97.78	62.84	0.02×	-
WoodFisher	98	65.55	-	-

At medium sparsities (80% and 90%), AC/DC sparse models are slightly less accurate than the state-of-the-art post-training methods (STR, WoodFisher), by small margins. The situation is reversed at higher sparsities, where AC/DC produces significantly more accurate models: the gap to the second-best methods (WoodFisher / Top-KAST) is of more than 1% at 95% and 98% sparsity.

Of the existing sparse training methods, Top-KAST is closest in terms of validation accuracy to our sparse model, at 90% sparsity. However, Top-KAST does not prune the first and last layers, whereas the results in the tables do not restrict the sparsity pattern. For fairness, we executed AC/DC using the same layer-wise sparsity distribution as Top-KAST, for both uniform and global magnitude pruning. For 90% global pruning, results for AC/DC improved significantly; the best sparse model reached 75.63% validation accuracy (0.6% increase over Table 1), while the best dense model had

76.85% after fine-tuning. For uniform sparsity, our results did not change significantly: 75.04% validation accuracy for the sparse model and 76.43% - for the fine-tuned dense model. We also note that Top-KAST has better results at 98% when increasing the number of training epochs 2 times, and considerably fewer training FLOPs (e.g. 15% of the dense FLOPs). For fairness, we compared against all methods on a fixed number of 100 training epochs and we additionally trained AC/DC at high sparsity without pruning the first and last layers. Our results improved significantly: 74.16% accuracy for 95% sparsity, and 71.27% for 98% sparsity, both surpassing Top-KAST with prolonged training. We provide a more detailed comparison in the Supplementary Material, which also contains results on the CIFAR-100 dataset.

An advantage of AC/DC is that it provides *both* sparse and dense models at cost *below* that of a single dense training run. For medium sparsity, the accuracy of the dense-finetuned model is very close to the dense baseline. Concretely, at 90% sparsity, with 58% of the total (theoretical) baseline training FLOPs, we obtain a *sparse* model which is close to state of the art; in addition, by fine-tuning the best dense model, we obtain a dense model with 76.56% (average) validation accuracy. The whole process takes 73% of the baseline training FLOPs.

Interestingly, the sparsity distribution over layers does not change significantly during training; yet, the dynamics of the masks has an important impact on the performance of AC/DC, as it can enable faster recovery from sub-optimal pruning decisions. Specifically, we observed that masks keep getting updated over time, although the change between consecutive sparse masks decreases, and small percentage of the weights remain fixed at 0 even during dense training. Please see the Supplementary material for additional results and analysis.

Table 3: Sparsity results on MobileNetV1.

Method	Sparsity (%)	Top-1 Acc. (%)	GFLOPs Inference	EFLOPs Train
Dense	0	71.78	1.1	0.44
AC/DC	74.61	$\mathbf{70.3 \pm 0.07}$	0.35×	0.65×
$\text{RigL}_{1\times}$ (ERK)	75	68.39	0.52×	0.53×
STR	75.28	68.35	0.18×	-
WoodFisher	75.28	70.09	0.28×	-
AC/DC	89.52	$\mathbf{66.08 \pm 0.09}$	0.19×	0.57×
$\text{RigL}_{1\times}$ (ERK)	90	63.58	0.27×	0.29×
STR	89.01	62.1	0.07×	-
WoodFisher	89	63.87	0.15×	-

Table 4: Sparsity results on Transformer-XL.

Method	Sparsity (%)	Perplexity Sparse	Perplexity Dense	Perplexity Finetuned Dense
Dense	0	-	18.95	-
AC/DC	80	20.93	20.36	19.46
AC/DC	80, 50 embed.	20.99	20.28	19.5
Top-KAST	80, **0 bwd**	**19.8**	-	-
Top-KAST	80, 60 bwd	21.3	-	-
AC/DC	90	**22.65**	21.07	20.08
AC/DC	90, 50 embed.	**22.81**	21.02	20.07
Top-KAST	90, 80 bwd	25.1	-	-

MobileNet Results. We perform the same experiment, using exactly the same setup, on the MobileNetV1 architecture [30], which is compact and thus harder to compress. On a training budget of 100 epochs, our method finds sparse models with higher Top-1 validation accuracy than existing sparse- and post-training methods, on both 75% and 90% sparsity levels (Table 3). Importantly, AC/DC uses exactly the same hyper-parameters used for training the dense baseline [37]. Similar to ResNet50, at 75% sparsity, the *dense-finetuned model* recovers the baseline performance, while for 90% it is less than 1% below the baseline.

The only method which obtains higher accuracy for the same sparsity is the version of RigL [16] which executes for 5x more training epochs than the dense baseline. However, this version also uses more computation than the dense model. We limit ourselves to a fixed number of 100 epochs, the same used to train the dense baseline, which would allow for savings in training time. Moreover, RigL does not prune the first layer and the depth-wise convolutions, whereas for the results reported we do not impose any sparsity restrictions. Overall, we found that keeping these layers dense improved our results on 90% sparsity by almost 0.5%. Then, our results are quite close to $\text{RigL}_{2\times}$, with half the training epochs, and less training FLOPs. We provide a more detailed comparison in the Supplementary material.

Semi-structured Sparsity. We also experiment with the recent 2:4 sparsity pattern (2 weights out of each block of 4 are zero) proposed by NVIDIA, which ensures inference speedups on the Ampere architecture. Recently, [43] showed that accuracy can be preserved under this pattern, by deriving masks from the pretrained dense model, and re-doing the entire training flow. Also, [60] proposed more general N:M structures, together with a method for training such sparse models from scratch. We applied AC/DC to the 2:4 pattern, performing training from scratch and obtained sparse models with $76.64\% \pm 0.05$ validation accuracy, i.e. slightly below the baseline. Furthermore, the dense-finetuned model fully recovers the baseline performance (76.85% accuracy).

Language Modeling. Next, we apply AC/DC to compressing NLP models. We use Transformer-XL [10], on the WikiText-103 dataset [42], with the standard model configuration with 18 layers and

285M parameters, trained using the Lamb optimizer [56] and standard hyper-parameters, which we describe in the Supplementary Material.

The same Transformer-XL model trained on WikiText-103 was used in Top-KAST [32], which allows a direct comparison. Similar to Top-KAST, we did not prune the embedding layers, as this greatly affects the quality, without reducing computational cost. (For completeness, we do provide results when embeddings are pruned to 50% sparsity.) Our sparse training configuration consists in starting with a dense warm-up phase of 5 epochs, followed by alternating between compression and decompression phases every 3 epochs; we follow with a longer decompression phase between epochs 33-39, and end with a compression phase between epochs 40-48.

The results are shown in Table 4. Relative to Top-KAST, our approach provides significantly improved test perplexity at 90% sparsity, as well as better results at 80% sparsity with sparse back-propagation. (Top-KAST provides superior perplexity at 0% backward sparsity, but this provides little to no training speedup.) The results confirm that AC/DC is scalable and extensible. We note that our hyper-parameter tuning for this experiment was minimal.

Output Analysis. Finally, we probe the accuracy difference between the sparse and dense-finetuned models. We first examineed *sample-level agreement* between sparse and dense-finetuned pairs produced by AC/DC, relative to model pairs produced by gradual magnitude pruning (GMP). Co-trained model pairs consistently agree on more samples relative to GMP: for example, on the 80%-pruned ResNet50 model, the AC/DC model pair agrees on the Top-1 classification of 90% of validation samples, whereas the GMP models agree on 86% of the samples. A similar trend holds for the *cross-entropy* between model outputs.

Second, we analyze differences in "memorization" capacity [59] between dense and sparse models. For this, we apply AC/DC to ResNet20 trained on a variant of CIFAR-10 where a subset of 1000 samples have randomly corrupted class labels, and examine the accuracy on these samples during training. We consider 90% and 95% sparsity AC/DC runs. Figure 2b shows the results, when the accuracy for each sample is measured with respect to the *true, un-corrupted* label. During early training and during *sparse phases*, the network tends to classify corrupted samples to their *true class*, "ignoring" label corruption. However, as training progresses, due to dense training phases and lower learning rate, networks tend to "memorize" these samples, assigning them to their corrupted class. This phenomenon is even more prevalent at 95% sparsity, where the network is less capable of memorization. We discuss this finding in more detail in the Supplementary material.

Practical Speedups. One remaining question regards the potential of sparsity to provide real-world speedups. While this is an active research area, e.g. [15], we partially address this concern in the Supplementary Material, by showing inference speedups for our models on a CPU inference platform supporting unstructured sparsity [12]: for example, our 90% sparse ResNet50 model provides 1.75x speedup for real-time inference (batch-size 1) on a resource-constrained processor with 4 cores, and 2.75x speedup on 16 cores at batch size 64, versus the dense model.

5 Conclusion, Limitations, and Future Work

We introduced AC/DC—a method for co-training sparse and dense models, with theoretical guarantees. Experimental results show that AC/DC improves upon the accuracy of previous sparse training methods, and obtains state-of-the-art results at high sparsities. Importantly, we recover near-baseline performance for dense models and do not require extensive hyper-parameter tuning. We also show that AC/DC has potential for real-world speed-ups in inference and training, with the appropriate software and hardware support. The method has the advantage of returning both an accurate standard model, and a compressed one, which may be useful in deployment scenarios. Our model output analysis confirms the intuition that sparse training phases act as a regularizer, preventing the (dense) model from memorizing corrupted samples. At the same time, they prevent the memorization of *hard samples*, which can affect accuracy.

The main limitations of AC/DC are its reliance on dense training phases, which limits the achievable training speedup, and the need for tuning the length and frequency of sparse/dense phases. We believe the latter issue can be addressed with more experimentation, as simple patterns tend to work well out-of-the-box; however, both the theoretical results and the output analysis suggest that dense phases may be *necessary* for good accuracy. We plan to further investigate this in future work, together with applying AC/DC to other compression methods, such as quantization, as well as leveraging sparse training on hardware that could efficiently support it, such as Graphcore IPUs [23].

References

[1] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, 2018.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

[3] Kyriakos Axiotis and Maxim Sviridenko. Sparse convex optimization via adaptively regularized hard thresholding. In *International Conference on Machine Learning*, pages 452–462. PMLR, 2020.

[4] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

[5] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[6] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14(5-6):629–654, 2008.

[7] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

[8] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 578–594, 2018.

[9] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[10] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[11] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, et al. Tensorflow lite micro: Embedded machine learning on tinyml systems. *arXiv preprint arXiv:2010.08678*, 2020.

[12] DeepSparse. Neuralmagic deepsparse inference engine, 2021.

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

[14] Xin Dong, Shangyu Chen, and Sinno Jialin Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon, 2017.

[15] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14629–14638, 2020.

[16] 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.

[17] Simon Foucart. Hard thresholding pursuit: an algorithm for compressive sensing. *SIAM Journal on Numerical Analysis*, 49(6):2543–2563, 2011.

[18] Simon Foucart. Sparse recovery algorithms: sufficient conditions in terms of restricted isometry constants. In *Approximation Theory XIII: San Antonio 2010*, pages 65–77. Springer, 2012.

[19] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

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

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

[22] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[23] Graphcore. Graphcore poplar sdk 2.0, 2021.

[24] Masafumi Hagiwara. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6(2):207 – 218, 1994. Backpropagation, Part IV.

[25] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[26] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 1135–1143, 2015.

[27] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[29] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[30] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[31] Prateek Jain, Ambuj Tewari, and Purushottam Kar. On iterative hard thresholding methods for high-dimensional M-estimation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 1*, pages 685–693, 2014.

[32] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020.

[33] Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods. *arXiv preprint arXiv:1607.05423*, 2016.

[34] Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer, 2016.

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

[36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[37] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pages 5544–5555. PMLR, 2020.

[38] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605, 1990.

[39] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[40] Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2019.

[41] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv preprint arXiv:2003.00307*, 2020.

[42] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[43] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[44] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[45] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

[46] 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 *Advances in Neural Information Processing Systems 32*. 2019.

[47] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[48] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[49] Alexander Shevchenko and Marco Mondelli. Landscape connectivity and dropout stability of sgd solutions for over-parameterized neural networks. In *International Conference on Machine Learning*, pages 8773–8784. PMLR, 2020.

[50] Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.

[51] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33, 2020.

[52] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13693–13696, 2020.

[53] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33, 2020.

[54] Han Vanholder. Efficient inference with TensorRT. NVIDIA GTC On-Demand. Slides available at https://on-demand-gtc.gputechconf.com/gtcnew/sessionview.php?sessionName=23425-efficient+inference+with+tensorrt, 2017.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

[56] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2019.

[57] Xiaotong Yuan, Ping Li, and Tong Zhang. Gradient hard thresholding pursuit for sparsity-constrained optimization. In *International Conference on Machine Learning*, pages 127–135. PMLR, 2014.

[58] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

[59] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[60] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: M fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021.

[61] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Sections 3, 4
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 3.1
   (b) Did you include complete proofs of all theoretical results? [Yes] See Supplemental Material
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will include the code with the Supplemental Material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We included standard deviations for experiments in Tables 1, 2, 3, and for CIFAR100 experiments
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We included a discussion at the beginning of Section 4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited the creators of all datasets and architectures used
   (b) Did you mention the license of the assets? [N/A] We used publicly available models and datasets
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We will include the code in the Supplemental material, which is based on an open-source repository for model compression.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We only used publicly available data
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We used standard machine learning benchmarking datasets.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not use crowdsourcing or conducted research with human subjects
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]