PARAMETER EFFICIENT TRAINING OF DEEP CONVOLUTIONAL NEURAL NETWORKS BY DYNAMIC SPARSE REPARAMETERIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern deep neural networks are highly overparameterized, and often of huge sizes. A number of post-training model compression techniques, such as distillation, pruning and quantization, can reduce the size of network parameters by a substantial fraction with little loss in performance. However, training a small network of the post-compression size *de novo* typically fails to reach the same level of accuracy achieved by compression of a large network, leading to a widely-held belief that gross overparameterization is essential to effective learning. In this work, we argue that this is not necessarily true. We describe a dynamic sparse reparameterization technique that closed the performance gap between a model compressed by pruning and a model of the post-compression size trained *de novo*. We applied our method to training deep residual networks and showed that it outperformed existing static reparameterization techniques, yielding the best accuracy for a given parameter budget for training. Compared to other dynamic reparameterization methods that reallocate non-zero parameters during training, our approach broke free from a few key limitations and achieved much better performance at lower computational cost. Our method is not only of practical value for training under stringent memory constraints, but also potentially informative to theoretical understanding of generalization properties of overparameterized deep neural networks.

1 INTRODUCTION

Deep neural network's success is due in no small part to two of its surprising properties that starkly defy conventional wisdom. First, training of modern deep neural networks is a high-dimensional non-convex optimization problem, a kind prohibitively difficult in general from an optimization theoretic viewpoint. Yet first-order gradient-based methods, stochastic gradient descent (SGD) and its variants, prove to be very effective; training seldom gets trapped in bad local minima. Second, overparameterization (i.e. having more model parameters than training data examples) does not undermine generalization; highly overparameterized models in practice seldom overfit, a phenomenon at odds with traditional principles of statistical learning theory.

Simply put, huge models are not hard to train, and trained huge models are not bad ones.

Emerging evidence has attributed these effects to the geometry of high-dimensional loss landscapes of overparameterized deep neural networks (Dauphin et al., 2014; Choromanska et al., 2014; Goodfellow et al., 2014; Im et al., 2016; Wu et al., 2017; Liao & Poggio, 2017; Cooper, 2018; Novak et al., 2018), and to the implicit regularization properties of SGD (Brutzkus et al., 2017; Zhang et al., 2018a; Poggio et al., 2017), though a thorough theoretical understanding is not yet complete. On the other hand, practitioners, blessed in this reality, do not seem to shy away from building gigantic models. Models with parameters in the tens and hundreds of millions are common in practice, with the current record holder having a parameter count in the range of a hundred billion (Shazeer et al., 2017).

However, huge models are computationally expensive, both spatially (in terms of memory requirements) and temporally (in terms of number of operations), at both training time and inference time. Thus, scalable methods for controlling the computational complexity of deep neural networks are key to making them more useful in practice.

To achieve efficient inference, a number of well-established model compression techniques are highly effective in reducing the post-training model size with little degradation in performance. These include distillation (e.g. Bucilua et al. (2006); Hinton et al. (2015)), quantization (e.g. Hubara et al. (2016); McDonnell (2018)), low-rank decomposition (e.g. Denil et al. (2013); Jaderberg et al. (2014)), and pruning (e.g. Han et al. (2015); Zhang et al. (2018b)), to name a few.

Despite their success, post-training compression methods require the full overparameterized model to be trained in the first place. Training remains expensive. In order to make training more efficient, significant efforts have been invested into numerical innovations for training at limited precision (e.g. Courbariaux et al. (2016); Köster et al. (2017)). In contrast, however, little progress has been made in effective training at significantly reduced number of parameters.

One obvious alternative toward this goal is to seek novel network architectures, ones that are more *parameter efficient* [1] than the original. In fact, recent architectural innovations in deep convolutional neural networks (CNNs) not only achieved better generalization performance, but parameter efficiency as well (see Section 5).

Instead of inventing new networks, a more desirable approach is to achieve higher parameter efficiency directly by reparameterizing an existing model architecture. In general, any *differentiable reparameterization* can be used to augment training of a given model. Let an original network (or a layer therein) be denoted by $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. Reparameterize it by $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ through $\boldsymbol{\theta} = g(\boldsymbol{\phi}; \boldsymbol{\psi})$, where g is differentiable w.r.t. $\boldsymbol{\phi}$ but not necessarily w.r.t. $\boldsymbol{\psi}$. Denote the reparameterized network by $f_{\boldsymbol{\psi}}$, considering $\boldsymbol{\psi}$ as *metaparameters* [2]:

$$\boldsymbol{y} = f_{\boldsymbol{\psi}}\left(\boldsymbol{x}; \boldsymbol{\phi}\right) \triangleq f\left(\boldsymbol{x}; g(\boldsymbol{\phi}; \boldsymbol{\psi})\right). \tag{1}$$

Training of $f_{\boldsymbol{\psi}}$ is done by backpropagation further through g, as $\frac{\partial}{\partial \boldsymbol{\phi}} = \frac{\partial g}{\partial \boldsymbol{\phi}} \frac{\partial}{\partial g}$. If it is so chosen that $\dim(\boldsymbol{\Phi}) < \dim(\boldsymbol{\Theta})$ and $f_{\boldsymbol{\psi}} \approx f$ in terms of its generalization performance, then $f_{\boldsymbol{\psi}}$ is a more parameter efficient approximation of f.

Sparse reparameterization is a special case where g is a linear projection; $\boldsymbol{\phi}$ is the non-zero entries (i.e. "weights") and $\boldsymbol{\psi}$ their indices (i.e. "connectivity") in the original parameter $\boldsymbol{\theta}$. Likewise, *parameter sharing* (or more generally *linear constraints of parameters*) is a similar special case of linear reparameterization where $\boldsymbol{\phi}$ is the tied parameters and $\boldsymbol{\psi}$ their indices. In this sense a convolution layer can be considered a reparameterization of a fully connected layer.

Further, if metaparameters $\boldsymbol{\psi}$ are fixed during the course of training, the reparameterization is *static*, whereas if $\boldsymbol{\psi}$ is adjusted adaptively during training, we call it *dynamic* reparameterization.

Now, the question is: given a full model f, is it possible to find a more efficient reparameterization $f_{\boldsymbol{\psi}}$ that, trained *de novo*, can generalize comparably well? The success of various model compression techniques suggests that such reparameterizations might exist for most well-known models, and a recent study (Frankle & Carbin, 2018) made successful *post hoc* identifications of sparse reparameterized small networks with precisely such properties. Nevertheless, all attempts of training small networks *de novo* have yielded results significantly underperforming those compressed from the original models (Zhu & Gupta, 2017). This has led to a commonly held belief that gross overparameterization is necessary for effective training, and it is further hypothesized that this necessity is due to the haphazard occurrences of good reparameterizations and their sensitivity to initialization (Frankle & Carbin, 2018). To date, a method capable of training a sparse network *de novo* to match the performance of a dense model compressed to the same size remains out of reach.

In this study, we present a dynamic sparse reparameterization technique that successfully overcame this challenge. Our major discoveries and contributions are listed as follows.

1. Ours is the first systematic method able to train sparse models directly without an increased parameter footprint during the entire course of training, and still achieve performance on par with post-training compression of dense models, the best result at a given sparsity.

[1] We formally introduce the notion of *parameter efficiency* as follows: given a dataset and a task, if model family A achieves a specific level of generalization performance with fewer parameters than model family B, we say A is more *parameter efficient* than B at that performance level.

[2] We use the term *metaparameter* to refer to the parameters $\boldsymbol{\psi}$ of the reparameterization function g. They differ from parameters $\boldsymbol{\phi}$ in that they are not simultaneously optimized with model parameters, and they differ from hyperparameters in that they define meaningful structures of the model which are required for inference.

2. We described the first dynamic reparameterization method for training convolutional networks.
3. We showed that our dynamic sparse reparameterization significantly outperformed static ones.
4. Our method not only outperformed existing dynamic sparse reparameterization techniques, but also incurred much lower computational costs.

2 RELATED WORK

Training of differentiably reparameterized networks has been proposed in numerous studies before.

Dense reparameterization Several dense reparameterization techniques sought to reduce the size of fully connected layers. These include low-rank decomposition (Denil et al., 2013), fastfood transform (Yang et al., 2014), ACDC transform (Moczulski et al., 2015), HashedNet (Chen et al., 2015), low displacement rank (Sindhwani et al., 2015) and block-circulant matrix parameterization (Treister et al., 2018).

Note that similar reparameterizations were also used to introduce certain algebraic properties to the parameters for purposes other than reducing model sizes, e.g. to make training more stable as in unitary evolution RNNs (Arjovsky et al., 2015) and in weight normalization (Salimans & Kingma, 2016), to inject inductive biases (Thomas et al., 2018), and to alter (Dinh et al., 2017) or to measure (Li et al., 2018) properties of the loss landscape. All dense reparameterization methods to date are static.

Sparse reparameterization Successful training of sparse reparameterized networks invariably employed iterative pruning and retraining (e.g. Han et al. (2015); Narang et al. (2017); Zhu & Gupta (2017)). Training typically started with a large model and sparsity was scheduled to increase during the course of learning. Training a small, sparse model *de novo* always fared much worse than training a large one to begin with (Zhu & Gupta, 2017).

Frankle & Carbin (2018) successfully identified small and sparse subnetworks post-training which, when trained in isolation, reached a similar accuracy as the enclosing big network. They further showed that these subnetworks were sensitive to initialization, and hypothesized that the role of overparameterization is to provide a large number of candidate subnetworks, thereby increasing the likelihood that one of these subnetworks will have the necessary structure and initialization needed for effective learning.

Most closely related to our work are dynamic sparse reparameterization techniques that emerged only recently. Like ours, these methods adaptively alter, by certain heuristic rules, reparameterization during training. Sparse evolutionary training (Mocanu et al., 2018) used magnitude-based pruning and random growth at the end of each training epoch. NeST (Dai et al., 2017; 2018) iteratively grew and pruned parameters and neurons during training; parameter growth was guided by gradient and pruning by magnitude. Deep rewiring (Bellec et al., 2017) combined sparse reparameterization with stochastic parameter updates for training. These methods were mostly concerned with sparsifying fully connected layers and applied to relatively small and shallow networks. As will be discussed in Section 5, our method, more scalable and computationally efficient than these previous approaches, fully closed the generalization gap for the first time between training a compact sparse network and compression of a large deep CNN.

3 METHODS

In this section, we describe our dynamic sparse reparameterization method. Some notations first. Let all reparameterized weight tensors in the original network be denoted by $\{\mathbf{W}_l\}$, where $l = 1, \cdots, L$ indexes layers. Let N_l be the number of parameters in $\mathbf{W}_l$, and $N = \sum_l N_l$ the total parameter count.

Sparse reparameterize $\mathbf{W}_l = g\left(\boldsymbol{\phi}_l; \boldsymbol{\psi}_l\right)$, where function g places components of parameter $\boldsymbol{\phi}_l$ into positions in $\mathbf{W}_l$ indexed by $\boldsymbol{\psi}_l \in \boldsymbol{\Psi}_{M_l}\left(\{1, \cdots, N_l\}\right)$ [3], s.t. $W_{l,\psi_{l,i}} = \phi_{l,i}, \forall i$ indexing components. Let $M_l < N_l$ be the dimensionality of $\boldsymbol{\phi}_l$ and $\boldsymbol{\psi}_l$, i.e. the number of non-zero weights in $\mathbf{W}_l$. Define $s_l = 1 - \frac{M_l}{N_l}$ as the *sparsity* of $\mathbf{W}_l$. Global sparsity is then defined as $s = 1 - \frac{M}{N}$ where $M = \sum_l M_l$.

[3] By $\boldsymbol{\Psi}_p(Q) \triangleq \left\{\sigma(\Psi) : \Psi \in 2^Q, \left|\Psi\right| = p, \sigma \in \mathrm{S}_p\right\}$ we denote the set of all cardinality p ordered subsets of finite set Q.

Algorithm 1: Reallocate free parameters within and across weight tensors

Input: $\left\{\left(\boldsymbol{\phi}_l^{(t)}, \boldsymbol{\psi}_l^{(t)}\right)\right\}$, $M^{(t)}$, $H^{(t)}$ ▷ From step t

Output: $\left\{\left(\boldsymbol{\phi}_l^{(t+1)}, \boldsymbol{\psi}_l^{(t+1)}\right)\right\}$, $M^{(t+1)}$, $H^{(t+1)}$ ▷ To step $t+1$

Need: K, δ ▷ Target number of parameters to be pruned and its fractional tolerance

for $l \in \{1, \cdots, L\}$ **do** ▷ For each reparameterized weight tensor

  $\Pi_l^{(t)} \leftarrow \left\{i : \left|\phi_{l,i}^{(t)}\right| < H^{(t)}\right\}$ ▷ Indices of subthreshold components of $\boldsymbol{\phi}_l^{(t)}$ to be pruned

  $\left(K_l^{(t)}, R_l^{(t)}\right) \leftarrow \left(\left|\Pi_l^{(t)}\right|, M_l^{(t)} - \left|\Pi_l^{(t)}\right|\right)$ ▷ Numbers of pruned and surviving weights

if $\sum_l K_l^{(t)} < (1-\delta)K$ **then** ▷ Too few parameters pruned

  $H^{(t+1)} \leftarrow 2H^{(t)}$ ▷ Increase pruning threshold

else if $\sum_l K_l^{(t)} > (1+\delta)K$ **then** ▷ Too many parameters pruned

  $H^{(t+1)} \leftarrow \frac{1}{2}H^{(t)}$ ▷ Decrease pruning threshold

else ▷ A proper number of parameters pruned

  $H^{(t+1)} \leftarrow H^{(t)}$ ▷ Maintain pruning threshold

for $l \in \{1, \cdots, L\}$ **do** ▷ For each reparameterized weight tensor

  $G_l^{(t)} \leftarrow \left\lfloor \frac{R_l^{(t)}}{\sum_l R_l^{(t)}} \sum_l K_l^{(t)} \right\rfloor$ ▷ Redistribute parameters for growth

  $\tilde{\boldsymbol{\psi}}_l^{(t)} \sim \mathcal{U}\left[\boldsymbol{\Psi}_{G_l^{(t)}}\left(\{1, \cdots, N_l\} \setminus \left\{\psi_{l,i}^{(t)}\right\}\right)\right]$ ▷ Sample zero positions to grow new weights

  $M_l^{(t+1)} \leftarrow M_l^{(t)} - K_l^{(t)} + G_l^{(t)}$ ▷ New parameter count

  $\left(\boldsymbol{\phi}_l^{(t+1)}, \boldsymbol{\psi}_l^{(t+1)}\right) \leftarrow \left(\left[\boldsymbol{\phi}_{l,i\notin\Pi_l^{(t)}}^{(t)}, \mathbf{0}\right], \left[\boldsymbol{\psi}_{l,i\notin\Pi_l^{(t)}}^{(t)}, \tilde{\boldsymbol{\psi}}_l^{(t)}\right]\right)$ ▷ New reparameterization

During the whole course of training, we kept global sparsity constant, specified by hyperparameter $s \in (0, 1)$. Reparameterization was initialized by uniformly sampling positions in each weight tensor at the global sparsity s, i.e. $\boldsymbol{\psi}_l^{(0)} \sim \mathcal{U}\left[\boldsymbol{\Psi}_{M_l^{(0)}}\left(\{1, \cdots, N_l\}\right)\right], \forall l$, where $M_l^{(0)} = \lfloor (1-s)N_l \rceil$. Associated parameters $\boldsymbol{\phi}_l^{(0)}$ were randomly initialized.

Dynamic reparameterization was done periodically by repeating the following steps during training:

1. Train the model (currently reparameterized by $\left\{\left(\boldsymbol{\phi}_l^{(t)}, \boldsymbol{\psi}_l^{(t)}\right)\right\}$) for P batch iterations;
2. Reallocate free parameters within and across weight tensors following Algorithm 1 to arrive at new reparameterization $\left\{\left(\boldsymbol{\phi}_l^{(t+1)}, \boldsymbol{\psi}_l^{(t+1)}\right)\right\}$.

The adaptive reallocation is in essence a two-step procedure: a global pruning followed by a tensor-wise growth. Specifically our algorithm has the following key features:

- Pruning was based on magnitude of weights, by comparing all parameters to a global threshold H, making the algorithm much more scalable than methods relying on layer-specific pruning.
- We made H adaptive, subject to a simple setpoint control dynamics that ensured roughly K weights to be pruned globally per iteration. This proved to be much cheaper computationally than pruning exactly K smallest weights, which requires sorting all weights in the network.
- Growth was by uniformly sampling zero weights and tensor-specific, thereby achieving a reallocation of parameters across layers. The heuristic guiding growth is

$$G_l^{(t)} = \left\lfloor \frac{R_l^{(t)}}{\sum_l R_l^{(t)}} \sum_l K_l^{(t)} \right\rfloor. \tag{2}$$

  This rule allocated more free parameters to weight tensors with more surviving entries, while keeping the global sparsity the same by balancing numbers of parameters pruned and grown [4].

[4] Note that an exact match is not guanranteed due to rounding errors in Eq. 2 and the possibility that $M_l^{(t)} - K_l^{(t)} + G_l^{(t)} > N_l$, i.e. free parameters in a weight tensor exceeding its dense size after reallocation. We added an extra step to redistribute parameters randomly to other tensors in these cases, thereby assuring an exact global sparsity.

Table 1: Datasets and models used in experiments

Dataset	MNIST	CIFAR10	Imagenet
Model	LeNet-300-100 (LeCun et al., 1998)	WRN-28-2 (Zagoruyko & Komodakis, 2016)	Resnet-50 (He et al., 2015)
Architecture	F300 F100 F10	C16/3×3 [C16/3×3,C16/3×3]×4 [C64/3×3,C64/3×3]×4 [C128/3×3,C128/3×3]×4 GlobalAvgPool, F10	C64/7×7-2, MaxPool/3×3-2 [C64/1×1, C64/3×3, C256/1×1]×3 [C128/1×1, C128/3×3, C512/1×1]×4 [C256/1×1, C256/3×3, C1024/1×1]×6 [C512/1×1, C512/3×3, C2048/1×1]×3 GlobalAvgPool, F1000
# Parameters	267K	1.5M	25.6M

For brevity architecture specifications omit batch normalization and activations. Fully connected (F) and convolutional (C) layers are specified with output size and kernel size, Max pooling (MaxPool) with kernel size and none with global average pooling (GlobalAvgPool). Brackets enclose residual blocks postfixed with repetition numbers; downsampling convolution in the first block of a scale group is implied.

The entire procedure can be fully specified by hyperparameters $\left(s, P, K, \delta, H^{(0)}\right)$. For specific values used in experiments see implementational details in Appendix A.

4 Experimental results

We evaluated our method in three sets of experiments (Table 1). We chose more modern convolutional networks, Resnet (He et al., 2015) and Wide Resnet (Zagoruyko & Komodakis, 2016), with parameter efficiency superior to earlier models, e.g. AlexNet (Krizhevsky et al., 2012) and VGG (Simonyan & Zisserman, 2014), thanks to the adoption of skip connections and use of global average pooling over fully connected layers. Such a setup makes a much stronger case for our method because compression is much harder for these recent models. Pre-activation batch normalization were used in all cases.

Dynamic sparse reparameterization was applied to all weight tensors of fully connected and convolutional layers (with the exception of downsampling convolutions and the first convolutional layer taking the input image), while all biases and parameters of normalization layers were kept dense [5].

At a specific sparsity s, we compared our method (*dynamic sparse*) against four baselines:

- *Full dense*: original large and dense model, with N parameters;
- *Thin dense*: original model with less wide layers, such that it had $(1-s)N$ parameters;
- *Static sparse*: original model reparameterized to sparsity s, then trained with connectivity fixed;
- *Compressed sparse*: state-of-the-art compression of the original model by iterative pruning and retraining the original model to target sparsity s (see Appendix A for details of implementation).

LeNet-300-100 on MNIST Like previous work on post-training compression, e.g. Han et al. (2015), and dynamic reparameterization, e.g. Bellec et al. (2017), we first experimented with a simple LeNet-300-100 trained on MNIST. We found that *static sparse* grossly underperformed *compressed sparse*, a gap effectively closed by our *dynamic sparse* method (Figure 1a) [6], with the following nuances. *Dynamic sparse* slightly outperformed *compressed sparse* at very high sparsity (i.e. low parameter count), a trend reversed, albeit weakly, at lower sparsities.

A breakdown of post-training sparsities across layers of the network (Figure 1b) reveals reliable patterns emerged from the dynamics of our adaptive reparameterization algorithm. This is consistent with previous observations (Bellec et al., 2017) that, given a fixed global sparsity, it was wiser to allocate more free parameters to the last layer, than to impose constant sparsity on all layers.

WRN-28-2 on CIFAR10 Next, we experimented with a Wide Resnet model WRN-28-2 (Zagoruyko & Komodakis, 2016) trained to classify CIFAR10 images (see Appendix A for details of imple-

[5] As described in Section 3, global sparsity s was defined as the overall sparsity of all reparameterized parameter tensors, not the entire model, which had a tiny fraction of dense parameters in our examples.

[6] At 99% sparsity, ours yielded 97.8% accuracy, cf. 97.4% by sparse compression and 96.3% by DeepR.

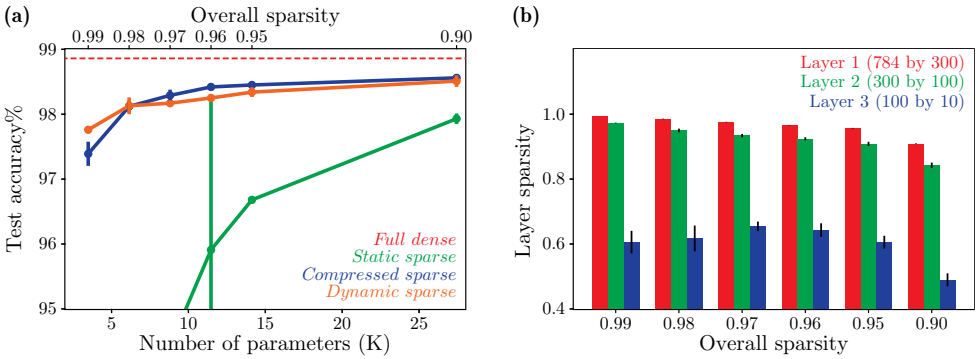

Figure 1: LeNet-300-100 on MNIST. (a) Test accuracy plotted against number of trainable parameters for different methods. Circular symbols mark the median of 5 runs, error bars standard deviation. Parameter counts include all trainable parameters, i.e. reparameterized sparse paremeter tensors plus all other parameter tensors that were kept dense, such as those of batch normalization layers. For small numbers of trainable parameters, the static sparse model fails to learn (not shown, out of ordinate range). (b) Layer-wise breakdown of final sparsities emerged from our dynamic sparse parameterization algorithm (Algorithm 1) at different levels of overall sparsity. Mean and standard deviation from 5 runs.

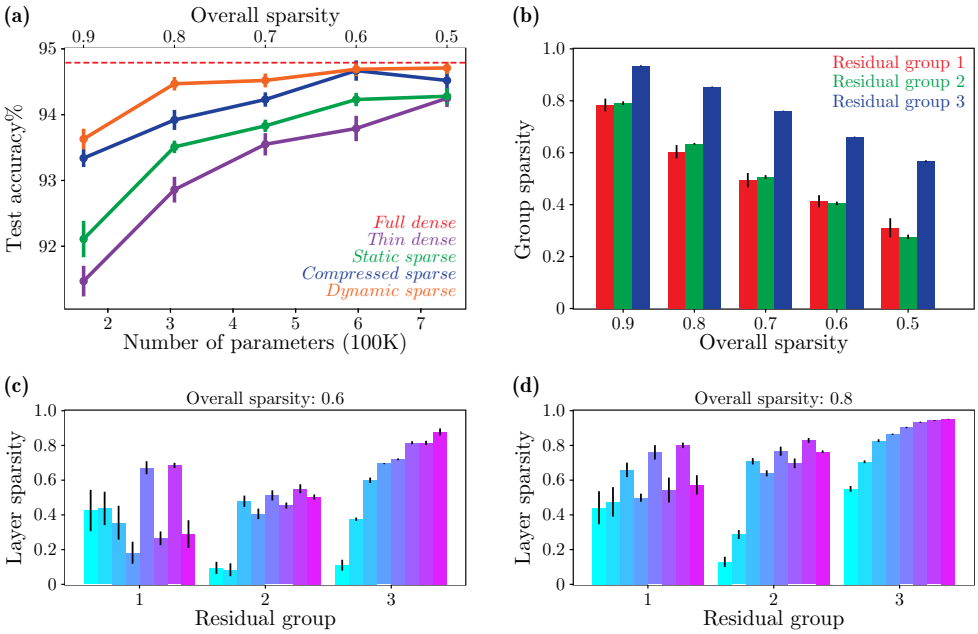

Figure 2: WRN-28-2 on CIFAR10. (a) Test accuracy plotted against number of trainable parameters for different methods. Conventions same as in Figure 1a. (b) Breakdown of final sparsities emerged from our dynamic sparse parameterization algorithm (Algorithm 1) at different levels of global sparsity. (c) A further breakdown of final sparsities of individual residual blocks in the WRN groups at three scales, at the overall sparsity of 0.6. (d) Same as (c) at overall sparsity of 0.8. All results in (b, c & d) are mean and standard deviation from 5 runs.

mentation). As shown in Figure 2a, *static sparse* and *thin dense* significantly underperformed the state-of-the-art *compressed sparse* model, whereas our method *dynamic sparse* significantly outperformed it. Again, consistent sparsity patterns emerged from the adaptive reparameterization dynamics of Algorithm 1 (Figure 2b, c & d). We observed two rough trends: (a) larger parameter tensors tended to be sparser than smaller ones, and (b) deeper layers tended to be sparser than shallower ones.

Resnet-50 on Imagenet Finally, we experimented with the Resnet-50 bottleneck architecture (He et al., 2015) trained on Imagenet (see Appendix A for details of implementation). We tested two global sparsity levels, 0.8 and 0.9 (Table 2). Again, our method (*dynamic sparse*) outperformed state-of-the-art sparse compression (*compressed sparse*), which in turn outperformed *static sparse* and *thin*

Table 2: Test accuracy% (top-1, top-5) of Resnet-50 trained on Imagenet

Final overall sparsity (# Parameters)	0.8 (7.3M)		0.9 (5.1M)	
Thin dense	71.6	90.3	69.4	89.2
Static sparse	70.4	89.8	66.4	87.4
Compressed sparse	73.2	91.5	70.3	90.0
Dynamic sparse (ours)	**73.3**	**92.4**	**71.6**	**90.5**

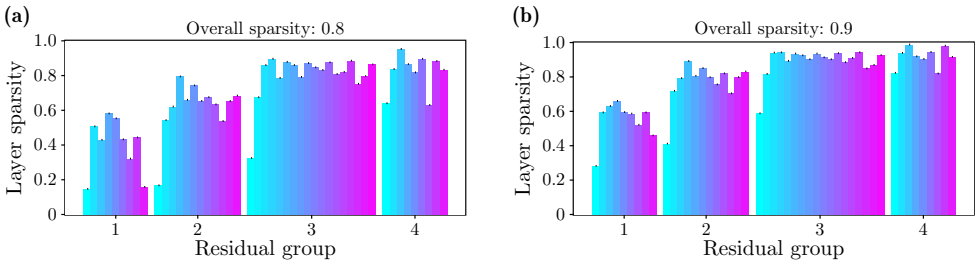

Figure 3: Block-wise breakdown of final sparsities of Resnet-50 trained on Imagenet. (a) At overall sparsity 0.8. Similar to Figure 2c. (b) Same as (a) at overall sparsity of 0.9.

dense baselines. Furthermore, consistent with the aforementioned experiments with LeNet-100-10 and WRN-28-2, reliable sparsity patterns across layers emerged from dynamic parameter reallocation during training, displaying the same empirical trends described above (Figure 3).

5 Discussion

In this work, we described and validated the best known solution so far to the following problem: given a small, fixed budget of parameters for a deep CNN throughout training time, how to train it to yield the best generalization performance. Our method used a dynamic reparameterization technique that adaptively reallocated free parameters across the network based on a simple heuristic. We demonstrated that this method not only fared much better than static reparameterization techniques, but also significantly outperformed even the state-of-the-art sparse compression methods. Note that compression is a much more forgiving situation where the parameter budget is not imposed during the entire course, but only at the end, of training.

Thus, our method yielded *the most parameter efficient sparse reparameterization* of a given CNN. The past few years have seen a steady improvement in parameter efficiency of state-of-the-art CNN models thanks to innovations in network architectures. Specifically, the introduction of residual modules and the use of global pooling in place of fully connected layers are largely responsible, as convolutional layers are more parameter efficient than fully connected layers in general. For example, AlexNet (Krizhevsky et al., 2012) had a majority of its parameters in fully connected layers, and this fraction has been decreasing in VGG (Simonyan & Zisserman, 2014), Inception (Szegedy et al., 2014), and more recently in Resnets (He et al., 2015) it was reduced to close to none. Meanwhile, compact architectures with far fewer parameters also emerged at reasonable costs of performance degradation, e.g. SqueezeNet (Iandola et al., 2016) and MobileNet (Howard et al., 2017), for niche use cases where computing resources are limited.

Here we took a different, but complementary, approach. Instead of searching the space of network architectures for new models with superior parameter efficiency, we sought to reparameterize a given model to make it more parameter efficient. Such techniques have a history as long as modern convolutional networks. Most early methods used dense reparameterization (Denil et al., 2013; Yang et al., 2014; Moczulski et al., 2015; Sindhwani et al., 2015), effective in reducing redundancy in fully connected layers of big sizes, such as in AlexNet. These became less relevant as models became increasingly free of large linear layers but heavy in convolutional ones [7].

[7] Note that a dense reparameterization trick, HashedNet (Chen et al., 2015), is closely related to sparse methods and can be readily applied to CNNs. We compared our method with HashedNet in Appendix B.

Sparse reparameterization, on the other hand, has been shown useful in compression of state-of-the-art deep CNNs (Han et al., 2015; Narang et al., 2017). However, a prominent empirical observation from these compression studies is that training a small sparse model *de novo* could never reach the same accuracy achieved by one of the same size compressed down from a large model (Zhu & Gupta, 2017). Why does it seem impossible to train a compact network but easy to compress a large one?

The "lottery ticket hypothesis" (Frankle & Carbin, 2018) offered a plausible explanation. The authors identified, by post-training pruning, trainable sparse models that, albeit sensitive to initialization, generalized comparably well (called "winning tickets"). They further posited that, the necessity of starting training with a large model is because only a big network with combinatorially a huge number of small subnetworks has a high probability of "winning the initialization lottery". Similarly, here we argue that *post hoc* identification of trainable compact (not necessarily sparse) reparameterization is actually rather trivial also because *the optimization trajectory is typically a very low-dimensional object*. Take for example the training of Resnet-152 (parameter count 60.2M) as described by He et al. (2015): parameter updates for 600K iterations reliably send a random initialization to a state-of-the-art solution. One could simply take a trivial *post hoc* reparameterization as a linear projection onto the span of the 600K parameter updates which, even if they are all linearly independent from each other, amount to less than 1% of the total parameter count. The low dimensionality of optimization trajectory is consistent with emerging evidence suggesting that optima of overparameterized deep neural networks are usually high-dimensional manifolds with low co-dimensions (Cooper, 2018), and that such optima have superior generalization properties (Wu et al., 2017) and favored by SGD (Zhang et al., 2018a; Poggio et al., 2017).

What prohibits any *post hoc* reparameterized compact model from being effectively trained *de novo* is however its sensitivity to initialization–though optimization trajectories are each low-dimensional, their linear spans do not necessarily overlap when starting from different initial points. An obvious way "to win the lottery" is to purchase all available tickets (i.e. having a huge model to cover all possible linear spans of optimization trajectories), but it is also conceivably feasible "to cheat" by adaptively changing the number on a single ticket as the lottery result is being announced (i.e. dynamic reparameterization that continually re-orients a low-dimensional parameter manifold tangentially along an optimization trajectory). We believe our method did exactly this.

Though we are the first to apply dynamic reparameterization to deep CNNs and our method the first to outperform state-of-the-art sparse compression techniques, it should be noted that ours is not the first dynamic sparse reparameterization approach. We were inspired by, and successfully exceeded the limitations of, three previous methods, DeepR (Bellec et al., 2017), SET (Mocanu et al., 2018) and NeST (Dai et al., 2017; 2018). Specifically, in addition to producing better sparse models, our approach was more scalable and much cheaper computationally. First, compared to DeepR, our method reallocated free parameters much less frequently (once per hundreds of training iterations), and did not require random walk in the parameter space, only random sampling of dimensions. Second, by using an adaptive threshold for pruning, we avoided expensive sorting operations as used by SET. Third, our reallocation of parameters across layers was automatic, without the need of manually configuring sparsity for each layer; the heuristic we used discovered similar optimal allocation patterns as manual choices [8]. Finally, our parameter reallocation, like DeepR, was by random sampling, which is much cheaper in memory consumption than gradient-based growth heuristic such as NeST, which requires gradients of zero entries of a sparse tensor to be tracked.

We believe this study might have profound implications. If training compact sparse models directly proved to be competitive as our work suggested, it would warrant a renewed look on practical choices of best hardware architectures for training, between computing devices optimized for dense linear algebra versus those suited for sparse operations.

REFERENCES

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary Evolution Recurrent Neural Networks. nov 2015. URL `http://arxiv.org/abs/1511.06464`.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep Rewiring: Training very sparse deep networks. nov 2017. URL `http://arxiv.org/abs/1711.05136`.

[8] Dynamic reallocation across layers indeed contributed to the success of our method (see Appendix C).

Alon Brutzkus, Amir Globerson, Eran Malach, and Shai Shalev-Shwartz. SGD Learns Overparameterized Networks that Provably Generalize on Linearly Separable Data. oct 2017. URL http://arxiv.org/abs/1710.10174.

Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model Compression. Technical report, 2006. URL https://www.cs.cornell.edu/{~}caruana/compression.kdd06.pdf.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing Neural Networks with the Hashing Trick. apr 2015. URL http://arxiv.org/abs/1504.04788.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The Loss Surfaces of Multilayer Networks. nov 2014. URL http://arxiv.org/abs/1412.0233.

Y Cooper. The loss landscape of overparameterized neural networks. apr 2018. URL http://arxiv.org/abs/1804.10200.

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. feb 2016. URL http://arxiv.org/abs/1602.02830.

Xiaoliang Dai, Hongxu Yin, and Niraj K. Jha. NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm. pp. 1–15, 2017. URL http://arxiv.org/abs/1711.02017.

Xiaoliang Dai, Hongxu Yin, and Niraj K. Jha. Grow and Prune Compact, Fast, and Accurate LSTMs. may 2018. URL http://arxiv.org/abs/1805.11797.

Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv*, pp. 1–14, 2014. URL http://arxiv.org/abs/1406.2572.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting Parameters in Deep Learning. jun 2013. URL http://arxiv.org/abs/1306.0543.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp Minima Can Generalize For Deep Nets. 2017. ISSN 1938-7228. URL http://arxiv.org/abs/1703.04933.

Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Small, Trainable Neural Networks. mar 2018. URL http://arxiv.org/abs/1803.03635.

Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. Qualitatively characterizing neural network optimization problems. dec 2014. URL http://arxiv.org/abs/1412.6544.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. jun 2015. URL http://arxiv.org/abs/1506.02626.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00124. URL http://arxiv.org/pdf/1512.03385v1.pdf.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. pp. 1–9, 2015. ISSN 0022-2488. doi: 10.1063/1.4931082. URL http://arxiv.org/abs/1503.02531.

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. apr 2017. URL http://arxiv.org/abs/1704.04861.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. sep 2016. URL http://arxiv.org/abs/1609.07061.

Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and. feb 2016. URL http://arxiv.org/abs/1602.07360.

Daniel Jiwoong Im, Michael Tao, and Kristin Branson. An empirical analysis of the optimization of deep network loss surfaces. dec 2016. URL http://arxiv.org/abs/1612.04010.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. may 2014. URL http://arxiv.org/abs/1405.3866.

Urs Köster, Tristan J. Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William H. Constable, Ouz H. Elibol, Scott Gray, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J. Pai, and Naveen Rao. Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks. nov 2017. URL http://arxiv.org/abs/1711.02213.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. Technical report, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998. ISSN 00189219. doi: 10.1109/5.726791.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the Intrinsic Dimension of Objective Landscapes. apr 2018. URL http://arxiv.org/abs/1804.08838.

Qianli Liao and Tomaso Poggio. Theory of Deep Learning II: Landscape of the Empirical Risk in Deep Learning. *arXiv*, mar 2017. URL http://arxiv.org/abs/1703.09833.

Mark D. McDonnell. Training wide residual networks for deployment using a single bit for each weight. feb 2018. URL http://arxiv.org/abs/1802.08530.

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):2383, dec 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04316-3. URL http://www.nature.com/articles/s41467-018-04316-3.

Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. ACDC: A Structured Efficient Linear Layer. nov 2015. URL http://arxiv.org/abs/1511.05946.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring Sparsity in Recurrent Neural Networks. apr 2017. URL http://arxiv.org/abs/1704.05119.

Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and Generalization in Neural Networks: an Empirical Study. feb 2018. URL http://arxiv.org/abs/1802.08760.

Tomaso Poggio, Kenji Kawaguchi, Qianli Liao, Brando Miranda, Lorenzo Rosasco, Xavier Boix, Jack Hidary, and Hrushikesh Mhaskar. Theory of Deep Learning III: explaining the non-overfitting puzzle. 2017. URL http://arxiv.org/abs/1801.00173.

Tim Salimans and Diederik P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. feb 2016. URL http://arxiv.org/abs/1602.07868.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. jan 2017. URL https://arxiv.org/abs/1701.06538.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. sep 2014. URL http://arxiv.org/abs/1409.1556.

Vikas Sindhwani, Tara N. Sainath, and Sanjiv Kumar. Structured Transforms for Small-Footprint Deep Learning. oct 2015. URL http://arxiv.org/abs/1510.01722.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. sep 2014. URL http://arxiv.org/abs/1409.4842.

Anna T Thomas, Albert Gu, Tri Dao, Atri Rudra, and R Christopher. Learning invariance with compact transforms. pp. 1–7, 2018.

Eran Treister, Lars Ruthotto, Michal Sharoni, Sapir Zafrani, and Eldad Haber. Low-Cost Parameterizations of Deep Convolution Neural Networks. may 2018. URL http://arxiv.org/abs/1805.07821.

Lei Wu, Zhanxing Zhu, and Weinan E. Towards Understanding Generalization of Deep Learning: Perspective of Loss Landscapes. jun 2017. URL http://arxiv.org/abs/1706.10239.

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep Fried Convnets. dec 2014. URL http://arxiv.org/abs/1412.7149.

Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. may 2016. URL http://arxiv.org/abs/1605.07146.

Chiyuan Zhang, Qianli Liao, Alexander Rakhlin, Brando Miranda, Noah Golowich, and Tomaso Poggio. Theory of Deep Learning IIb: Optimization Properties of SGD. jan 2018a. URL http://arxiv.org/abs/1801.02254.

Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers. apr 2018b. URL http://arxiv.org/abs/1804.03294.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. 2017. URL http://arxiv.org/abs/1710.01878.

APPENDIX A DETAILS OF IMPLEMENTATION

We implemented all models and reparameterization mechanisms using `pytorch`. Experiments were run on GPUs, and all sparse tensors were represented as dense tensors filtered by a binary mask [9]. Source code to reproduce all experiments is available on GitHub:

`Link suppressed for the sake of anonymity during review process.`

Training Hyperparameter settings for training are listed in the first block of Table 3. Standard mild data augmentation were used in all experiments for CIFAR10 (random translation, cropping and horizontal flipping) and for Imagenet (random cropping and horizontal flipping).

Dynamic reparameterization Hyperparameter settings for dynamic sparse reparameterization (Algorithm 1) are listed in the second block of Table 3.

[9] This is a mere implementational choice for ease of experimentation given available hardware and software, which did not save memory because of sparsity. With computing substrate optimized for sparse linear algebra, our method is duly expected to realize the promised memory efficiency.

Table 3: Hyperparameters for training, dynamic sparse reparameterization and sparse compression

Experiment	**LeNet-300-100 on MNIST**	**WRN-28-2 on CIFAR10**	**Resnet-50 on Imagenet**
Hyperparameters for training			
Number of training epochs	100	200	100
Mini-batch size	100	100	256
Learning rate schedule (epoch range: learning rate)	1 - 25: 0.100 26 - 50: 0.020 51 - 75: 0.040 76 - 100: 0.008	1 - 60: 0.100 61 - 120: 0.020 121 - 160: 0.040 161 - 200: 0.008	1 - 30: 0.1000 31 - 60: 0.0100 61 - 90: 0.0010 91 - 100: 0.0001
Momentum (Nesterov)	0.9	0.9	0.9
L^1 regularization multiplier	0.0001	0.0	0.0
L^2 regularization multiplier	0.0	0.0005	0.0001
Hyperparameters for dynamic sparse reparameterization			
Number of parameters to prune (K)	600	20,000	200,000
Fractional tolerance of K (δ)	0.1	0.1	0.1
Initial pruning threshold ($H^{(0)}$)	0.001	0.001	0.001
Reparameterization period (P) schedule (epoch range: P)	1 - 25: 100 26 - 50: 200 51 - 75: 400 76 - 100: 800	1 - 25: 100 26 - 80: 200 81 - 140: 400 141 - 200: 800	1 - 25: 1000 26 - 50: 2000 51 - 75: 4000 76 - 100: 8000
Hyperparameters for sparse compression (iterative pruning and re-training)			
Number of pruning iterations (T)	10	20	20
Number of training epochs between pruning iterations	2	2	2
Number of training epochs post-pruning	20	10	10
Number of epochs during pruning	40	50	50
Learning rate schedule during pruning (epoch range: learning rate)	1 - 20: 0.0200 21 - 30: 0.0040 31 - 40: 0.0008	1 - 25: 0.0200 25 - 35: 0.0040 36 - 50: 0.0008	1 - 25: 0.0100 26 - 35: 0.0010 36 - 50: 0.0001

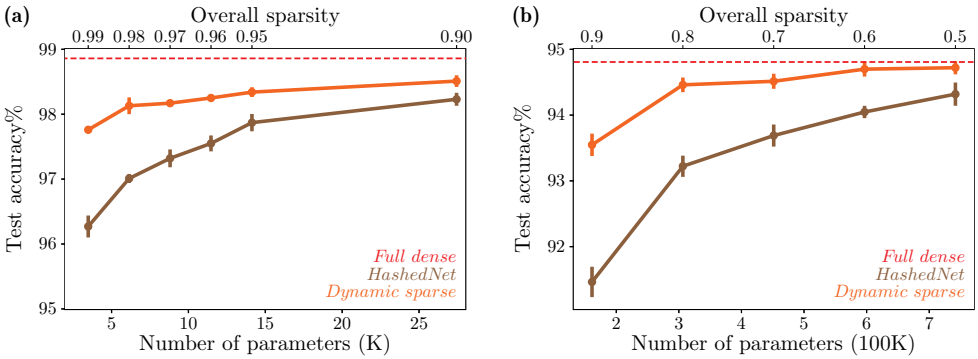

Figure 4: Comparison to *HashedNet*. (a) Test accuracy for LeNet-300-100-10 trained on MNIST. (b) Test accuracy for WRN-28-2 trained on CIFAR10. Conventions same as in Figure 1a.

Table 4: Test accuracy% (top-1, top-5) of Resnet-50 on Imagenet for *dynamic sparse* vs. *HashedNet*

Final overall sparsity (# Parameters)	0.8 (7.3M)		0.9 (5.1M)	
HashedNet	70.0	89.6	66.9	87.4
Dynamic sparse (ours)	**73.3**	**92.4**	**71.6**	**90.5**

Sparse compression baseline We compared our method against state-of-the-art model compression methods using iterative pruning during training (Han et al., 2015; Zhu & Gupta, 2017), starting from a full dense model trained with hyperparameters provided in the first block of Table 3 and then gradually prune the network to a target sparsity in T steps. The pruning schedule we used was

$$s^{(t)} = 1 - (1 - s)^{\frac{t}{T}}, \tag{3}$$

where $t = 0, 1, \cdots, T$ indexes pruning steps, and s the target sparsity reached at the end of training. Thus, this baseline (labeled as *compressed sparse* in the paper) was effectively trained for more iterations (original training phase plus compression phase) than our *dynamic sparse* method, making it a so far strongest possible baseline to benchmark our method.

Hyperparameter settings for sparse compression are listed in the third block of Table 3.

Exceptions For reasons described below, we made a few minor exceptions in sparsification for certain model layers in our experiments.

- The sparsity of the last linear layer in LeNet-300-100 was allowed to be sparsified to at most 90% sparsity. In cases where global sparsity was over 90%, we redistribute free parameters from earlier layers to maintain the total parameter count. This is because, if the last layer was allowed to be overly sparse, performance sustained huge degradation unsuited for meaningful comparison.
- The last linear layer of WRN-28-2 was always kept dense. It has a negligible fraction of parameter count.

Appendix B Comparison to HashedNet

We also compared our dynamic sparse reparameterization method to a number of static dense reparameterization techniques, e.g. Denil et al. (2013); Yang et al. (2014); Moczulski et al. (2015); Sindhwani et al. (2015); Chen et al. (2015); Treister et al. (2018). Instead of sparsification, these methods impose structure on large parameter tensors by parameter sharing. Most of these methods have not been used for convolutional layers except for recent ones (Chen et al., 2015; Treister et al., 2018). We found that *HashedNet* (Chen et al., 2015) had the best performance over other static dense reparameterization methods, and also benchmarked our method against it. Instead of reparameterizing a parameter tensor with N entries to a sparse one with $M < N$ non-zero components, *HashedNet*'s reparameterization is to put M free parameters into N positions in the parameter through a random mapping from $\{1, \cdots, N\}$ to $\{1, \cdots, M\}$ computed by cheap hashing, resulting in a dense parameter tensor with shared components.

Results of LeNet-300-100-10 on MNIST are presented in Figure 4a, those of WRN-28-2 on CIFAR10 in Figure 4b, and those of Resnet-50 on Imagenet in Table 4. For a certain global sparsity s of our

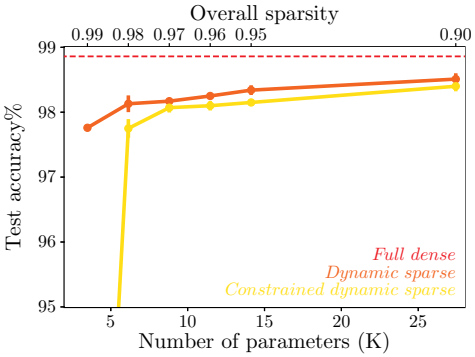

Figure 5: Test accuracy for LeNet-300-100-10 on MNIST of *dynamic sparse* (i.e. Algorithm 1) compared against *constrained dynamic sparse* for which parameter reallocation occurred only within, but not across, layers.

method, we compare it against a *HashedNet* with all reparameterized tensor hashed such that each had a fraction $1 - s$ of effective parameter count. We found that our method *dynamic sparse* significantly outperformed *HashedNet*.

APPENDIX C BENEFITS OF PARAMETER REALLOCATION ACROSS LAYERS

In order to assess whether and how much our parameter reallocation heuristic (Equation 2) contributed to the effectiveness of our method, we did a set of control experiments in which all free parameter reallocation were constrained to within parameter tensors, i.e. parameter reallocation across layers were disabled, and sparsity stayed constant (uniformly initialized) for each layer.

The results are presented in Figure 5 for LeNet-300-100-10 on MNIST. We found that removing inter-layer parameter allocation yielded worse performance, particularly at high sparsity levels. This suggested that parameter reallocation across the entire model was beneficial to our dynamic reparameterization.