Appendices: Parameter Efficient Training of Deep Convolutional Neural Networks by Dynamic Sparse Reparameterization

Hesham Mostafa [1] **Xin Wang** [1 2]

A. A full description of the dynamic parameter reallocation algorithm

Algorithm 1 in the main text informally describes our parameter reallocation scheme. In this appendix, we present a more rigorous description of the algorithm.

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 $\boldsymbol{\psi}_l \in \mathbf{\Psi}_{M_l}\left(\{1, \cdots, N_l\}\right)$ *, 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 $M = \sum_l M_l$.

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. $\boldsymbol{\psi}_l^{(0)} \sim \mathcal{U}\left[\mathbf{\Psi}_{M_l^{(0)}}\left(\{1, \cdots, N_l\}\right)\right], \forall l$, where $M_l^{(0)} = \left\lfloor (1-s) N_l \right\rceil$. 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:

1. 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.
2. We made H adaptive, subject to a simple setpoint control dynamics that ensured roughly N_p weights to be pruned globally per iteration. This is computationally cheaper than pruning exactly N_p smallest weights, which requires sorting all weights in the network.
3. Growth was by uniformly sampling zero weights and tensor-specific, thereby achieving a reallocation of parameters across layers. The heuristic guiding growth is

$$G_l^{(t)} = \left\lfloor \frac{R_l^{(t)}}{\sum_l R_l^{(t)}} \sum_l K_l^{(t)} \right\rfloor, \tag{1}$$

where $K_l^{(t)}$ and $R_l^{(t)} = M_l^{(t)} - K_l^{(t)}$ are the pruned and surviving parameter counts, respectively. 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 †.

The entire procedure can be fully specified by hyperparameters $\left(s, P, N_p, \delta, H^{(0)}\right)$.

B. Details of implementation

We implemented all models and reparameterization mechanisms using `pytorch`. Experiments were run on

[1]Artificial Intelligence Products Group, Intel Corporation, San Diego, CA, USA. [2]Currently with Cerebras Systems, Los Altos, CA, USA, work done while at Intel Corporation. Correspondence to: Xin Wang <xin@cerebras.net>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

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

†Note that an exact match is not guanranteed due to rounding errors in Eq. 1 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.

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

GPUs, and all sparse tensors were represented as dense tensors filtered by a binary mask [‡]. Source code to reproduce all experiments is available in the anonymous repository: `https://github.com/IntelAI/dynamic-reparameterization`.

Training Hyperparameter settings for training are listed in the first block of Table 1. Standard mild data augmentation was used in all experiments for CIFAR10 (random translation, cropping and horizontal flipping) and for Imagenet (random cropping and horizontal flipping). The last linear layer of WRN-28-2 was always kept dense as it has a negligible number of parameters. The number of training epochs for the *thin dense* and *static sparse* baselines are double the number of training epochs shown in Table 1.

Sparse compression baseline We compared our method against iterative pruning methods (Han et al., 2015; Zhu & Gupta, 2017). We start from a full dense model trained with hyperparameters provided in the first block of Table 1 and then gradually prune the network to a target sparsity in T steps. As in (Zhu & Gupta, 2017), the pruning schedule we used was

$$s^{(t)} = s + (1-s)\left(1 - \frac{t}{T}\right)^3, \tag{2}$$

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. Hyperparameter settings for sparse compression are listed in the second block of Table 1.

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

Sparse Evolutionary Training (SET) Because the larger-scale experiments here (WRN-28-2 on CIFAR10 and Resnet-50 on Imagenet) were not attempted by (Mocanu et al., 2018), no specific settings for reparameterization in these cases were available in the original paper. In order to make a fair comparison, we used the same hyperparameters as those used in our dynamic reparameterization scheme (third block in Table 1). At each reparameterization step, the weights in each layer were sorted by magnitude and the smallest fraction was pruned. An equal number of parameters were then randomly allocated in the same layer and initialized to zero. For control, the total number of reallocated weights at each step was chosen to be the same as our

[‡]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.

dynamic reparameterization method, as was the schedule for reparameterization.

Deep Rewiring (DeepR) The fourth block in Table 1 contain hyperparameters for the DeepR experiments. We refer the reader to (Bellec et al., 2017) for details of the deep rewiring algorithm and for explanation of the hyperparameters. We chose the DeepR hyperparameters for the different networks based on a parameter sweep.

C. Comparison to dense reparameterization method *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 1a, those of WRN-28-2 on CIFAR10 in Figure 1b, and those of Resnet-50 on Imagenet in Table 2. For a certain global sparsity s of our method, we compare it against a *HashedNet* with all reparameterized tensors hashed such that each had a fraction $1 - s$ of unique parameters. We found that our method *dynamic sparse* significantly outperformed *HashedNet*.

D. A taxonomy of training methods that yield "sparse" deep CNNs

As an extension to Section 2 of the main text, here we elaborate on existing methods related to ours, how they compare with and contrast to each other, and what features, apart from effectiveness, distinguished our approach from all previous ones. We confine the scope of comparison to training methods that produce smaller versions (i.e. ones with fewer parameters) of a given modern (i.e. post-AlexNet) deep convolutional neural network model. We list representative methods in Table 3. We classify these methods by three key features.

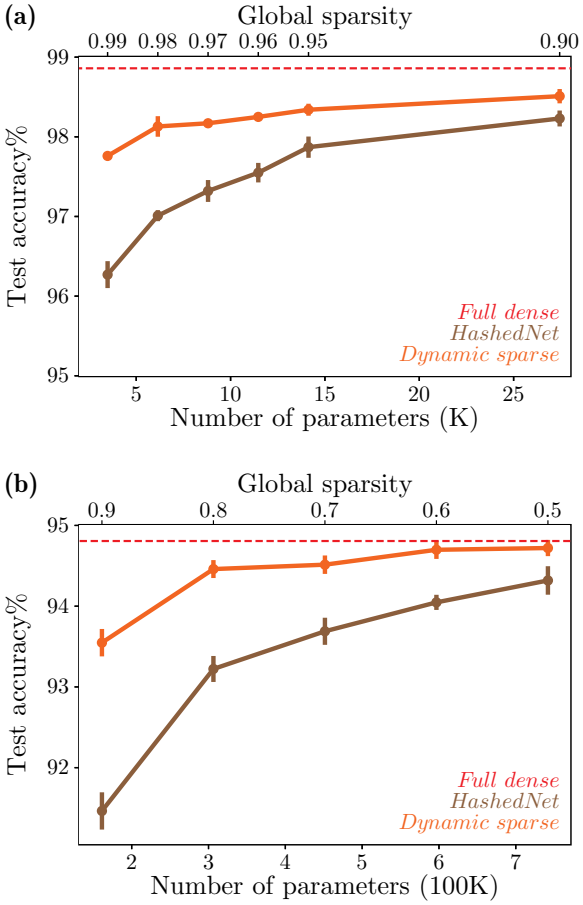

Figure 1: Comparison to *HashedNet*. (a) Test accuracy for LeNet-300-100-10 trained on MNIST. (b) Test accuracy for WRN-28-2 trained on CIFAR10. Conventions same as in Figure 3a.

Strict parameter budget throughout training and inference This feature was discussed in depth in the main text. Most of the methods to date are *compression* techniques, i.e. they start training with a fully parameterized, dense model, and then reduce parameter counts. To the best of our knowledge, only three methods, namely DeepR (Bellec et al., 2017), SET (Mocanu et al., 2018) and ours, *strictly* impose, throughout the entire course of training, a fixed small parameter budget, one that is equal to the size of the final sparse model for inference. We make a distinction between these *direct training* methods (first block) and *compression* methods (second and third blocks of Table 3) [§].

This distinction is meaningful in two ways: (a) practically,

[§]Note that an intermediate case is NeST (Dai et al., 2017; 2018), which starts training with a small network, grows it to a large size, and finally prunes it down again. Thus, a fixed parameter footprint is not strictly imposed throughout training, so we list NeST in the second block of Table 3.

direct training methods are more memory-efficient on appropriate computing substrate by requiring parameter storage of no more than the final compressed model size; (b) theoretically, these methods, if performing on par with or better than *compression* methods (as this work suggests), shed light on an important question: whether gross overparameterization during training is necessary for good generalization performance?

Granularity of sparsity The *granularity* of sparsity refers to the additional structure imposed on the placement of the non-zero entries of a sparsified parameter tensor. The finest-grained case, namely *non-structured*, allows each individual weight in a parameter tensor to be zero or non-zero independently. Early compression techniques, e.g. (Han et al., 2015), and more recent pruning-based compression methods based thereon, e.g. (Zhu & Gupta, 2017), are non-structured (second block of Table 3). So are all direct training methods like ours (first block of Table 3).

Non-structured sparsity can not be fully exploited by mainstream compute devices such as GPUs. To tackle this problem, a class of compression methods, *structured pruning* methods (third block in Table 3), constrain "sparsity" to a much coarser granularity. Typically, pruning is performed at the level of an entire feature map, e.g. ThiNet (Luo et al., 2017), whole layers, or even entire residual blocks (Huang & Wang, 2017). This way, the compressed "sparse" model has essentially smaller and/or fewer *dense* parameter tensors, and computation can thus be accelerated on GPUs the same way as dense neural networks.

These *structured compression* methods, however, did not make a useful baseline in this work, for the following reasons. First, because they produce dense models, their relevance to our method (non-structured, non-compression) is far more remote than non-structured compression techniques yielding sparse models, for a meaningful comparison. Second, typical structured pruning methods substantially underperformed non-structured ones (see Table 2 in the main text for two examples, ThiNet and SSS), and emerging evidence has called into question the fundamental value of structured pruning: (Mittal et al., 2018) found that the channel pruning criteria used in a number of state-of-the-art structured pruning methods performed no better than random channel elimination, and (Liu et al., 2018) found that fine-tuning in a number of state-of-the-art pruning methods fared no better than direct training of a randomly initialized pruned model which, in the case of channel/layer pruning, is simply a less wide and/or less deep dense model (see Table 2 in the main text for comparison of ThiNet and SSS against *thin dense*).

In addition, we performed extra experiments in which we constrained our method to operate on networks with structured sparsity and obtained significantly worse results, see Appendix E.

Predefined versus automatically discovered sparsity levels across layers The last key feature (rightmost column of Table 3) for our classification of methods is whether the sparsity levels of different layers of the network is automatically discovered during training or predefined by manual configuration. The value of automatic sparsification, e.g. ours, is twofold. First, it is conceptually more general because parameter reallocation heuristics can be applied to diverse model architectures, whereas layer-specific configuration has to be cognizant of network architecture, and at times also of the task to learn. Second, it is practically more scalable because it obviates the need for manual configuration of layer-wise sparsity, keeping the overhead of hyperparameter tuning constant rather than scaling with model depth/size. In addition to efficiency, we also show in Appendix F extra experiments on how automatic parameter reallocation across layers contributed to its effectiveness.

In conclusion, our method is unique in that it:

1. strictly maintains a fixed parameter footprint throughout the entire course of training.
2. automatically discovers layer-wise sparsity levels during training.

E. Structured versus non-structured sparsity

We investigated how our method performs if it were constrained to training sparse models at a coarser granularity. Consider a weight tensor of a convolution layer, of size $C_{\text{out}} \times C_{\text{in}} \times 3 \times 3$, where C_{out} and C_{in} are the number of output and input channels, respectively. Our method performed dynamic sparse reparameterization by pruning and reallocating individual weights of the 4-dimensional parameter tensor–the finest granularity. To adapt our procedure to coarse-grain sparsity on groups of parameters, we modified our algorithm (Algorithm 1 in the main text) in the following ways:

1. the pruning step now removed entire groups of weights by comparing their L^1-norms with the adaptive threshold.
2. the adaptive threshold was updated based on the difference between the target number and the actual number of groups to prune/grow at each step.
3. the growth step reallocated groups of weights within and across parameter tensors using the heuristic in Line 11 of Algorithm 1.

We show results at kernel-level granularity (i.e. groups are 3×3 kernels) in Figure 2 and Table 4, for WRN-28-2 on CIFAR10 and Resnet-50 on Imagenet, respectively. We observe that enforcing kernel-level sparsity leads to significantly worse accuracy compared to unstructured sparsity. For WRN-28-2, kernel-level parameter re-allocation still

outperforms the *thin dense* baseline, though the performance advantage disappears as the level of sparsity decreases. Note that the *thin dense* baseline was always trained for double the number of epochs used to train the models with dynamic parameter re-allocation.

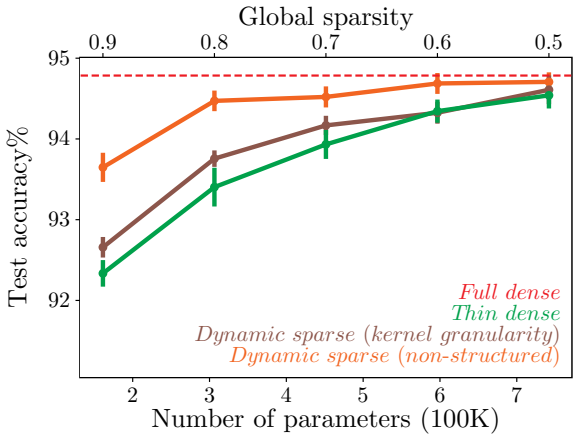

Figure 2: Test accuracy for WRN-28-2 trained on CIFAR10 for two variants of *dynamic sparse*, i.e. kernel-level granularity of sparsity and non-structured (same as *dynamic sparse* in the main text), as well as the *thin dense* baseline. Conventions same as in Figure 3a.

When we further coarsened the granularity of sparsity to channel level (i.e. groups are $C_{\text{in}} \times 3 \times 3$ slices that generate output feature maps), our method failed to produce performant models.

F. Multi-layer perceptrons and training at extreme sparsity levels

We carried out experiments on small multi-layer perceptrons to assess whether our dynamic parameter reallocation method can effectively distribute parameters in small networks at extreme sparsity levels. we experimented with a simple LeNet-300-100 trained on MNIST. Hyperparameters for the experiments are reported in appendix B. The results are shown in Fig. 3a. Our method is the only method, other than pruning from a large dense model, that is capable of effectively training the network at the highest sparsity setting by automatically moving parameters between layers to realize layer sparsities that can be effectively trained. The per-layer sparsities discovered by our method are shown in Fig. 3b. Our method automatically leads to a top layer with much lower sparsity than the two hidden layers. Similar sparsity patterns were found through hand-tuning to improve the performance of DeepR (Bellec et al., 2017). All layers were initialized at the same sparsity level (equal to the global sparsity level). While hand-tuning the per-layer sparsities should allow SET and DeepR to learn at the highest sparsity setting, our method automatically discovers the per-layer sparsities and allows us to dispense

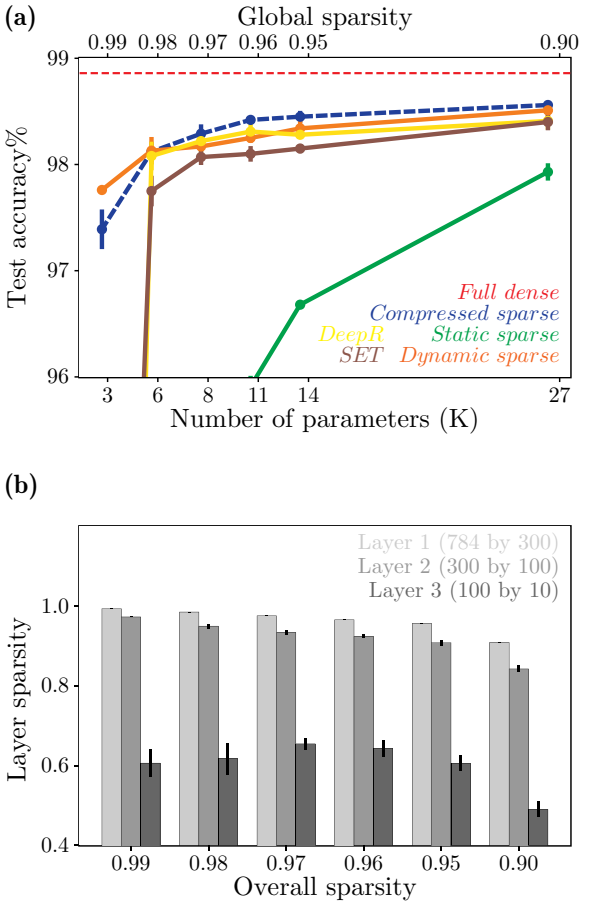

Figure 3: Test accuracy for LeNet-300-100-10 on MNIST for different training methods. Circular symbols mark the median of 5 runs, and error bars are the standard deviation. Parameter counts include all trainable parameters, i.e, parameters in sparse tensors plus all other dense tensors, such as those of batch normalization layers. Notice the failure of training at the highest sparsity level for *static sparse*, SET, and DeepR.

with such a tuning step.

References

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep Rewiring: Training very sparse deep networks. nov 2017. URL http://arxiv.org/abs/1711.05136.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., and Chen, Y. Compressing Neural Networks with the Hashing Trick. apr 2015. URL http://arxiv.org/abs/1504.04788.

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

Dai, X., Yin, H., and Jha, N. K. Grow and Prune Compact,

Fast, and Accurate LSTMs. may 2018. URL http://arxiv.org/abs/1805.11797.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and de Freitas, N. Predicting Parameters in Deep Learning. jun 2013. URL http://arxiv.org/abs/1306.0543.

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

He, Y., Zhang, X., and Sun, J. Channel Pruning for Accelerating Very Deep Neural Networks. jul 2017. URL http://arxiv.org/abs/1707.06168.

Huang, Z. and Wang, N. Data-Driven Sparse Structure Selection for Deep Neural Networks. jul 2017. URL https://arxiv.org/abs/1707.01213.

Lebedev, V. and Lempitsky, V. Fast ConvNets Using Group-wise Brain Damage. jun 2015. URL https://arxiv.org/abs/1506.02515.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning Filters for Efficient ConvNets. aug 2016. URL http://arxiv.org/abs/1608.08710.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning Efficient Convolutional Networks through Network Slimming. aug 2017. URL https://arxiv.org/abs/1708.06519.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the Value of Network Pruning. oct 2018. URL http://arxiv.org/abs/1810.05270.

Luo, J.-H., Wu, J., and Lin, W. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. jul 2017. URL http://arxiv.org/abs/1707.06342.

Mittal, D., Bhardwaj, S., Khapra, M. M., and Ravindran, B. Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks. jan 2018. URL http://arxiv.org/abs/1801.10447.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. 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.

Moczulski, M., Denil, M., Appleyard, J., and de Freitas, N. ACDC: A Structured Efficient Linear Layer. nov 2015. URL http://arxiv.org/abs/1511.05946.

Narang, S., Elsen, E., Diamos, G., and Sengupta, S. Exploring Sparsity in Recurrent Neural Networks. apr 2017. URL http://arxiv.org/abs/1704.05119.

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

Suau, X., Zappella, L., and Apostoloff, N. Network Compression using Correlation Analysis of Layer Responses. jul 2018. URL http://arxiv.org/abs/1807.10585.

Treister, E., Ruthotto, L., Sharoni, M., Zafrani, S., and Haber, E. Low-Cost Parameterizations of Deep Convolution Neural Networks. may 2018. URL http://arxiv.org/abs/1805.07821.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Yang, Z., Moczulski, M., Denil, M., de Freitas, N., Smola, A., Song, L., and Wang, Z. Deep Fried Convnets. dec 2014. URL http://arxiv.org/abs/1412.7149.

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

Table 1: Hyperparameters for all experiments presented in the paper

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	1 - 60:	0.100	1 - 30:	0.1000
	26 - 50:	0.020	61 - 120:	0.020	31 - 60:	0.0100
	51 - 75:	0.040	121 - 160:	0.040	61 - 90:	0.0010
	76 - 100:	0.008	161 - 200:	0.008	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 sparse compression (*compressed sparse*) (Zhu & Gupta, 2017)						
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
Total number of pruning epochs		40		50		50
Learning rate schedule during pruning (epoch range: learning rate)	1 - 20:	0.0200	1 - 25:	0.0200	1 - 25:	0.0100
	21 - 30:	0.0040	25 - 35:	0.0040	26 - 35:	0.0010
	31 - 40:	0.0008	36 - 50:	0.0008	36 - 50:	0.0001
Hyperparameters for dynamic sparse reparameterization (*dynamic sparse*) (ours)						
Number of parameters to prune (N_p)		600		20,000		200,000
Fractional tolerance of N_p (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	1 - 25:	100	1 - 25:	1000
	26 - 50:	200	26 - 80:	200	26 - 50:	2000
	51 - 75:	400	81 - 140:	400	51 - 75:	4000
	76 - 100:	800	141 - 200:	800	76 - 100:	8000
Hyperparameters for Sparse Evolutionary Training (*SET*) (Mocanu et al., 2018)						
Number of parameters to prune at each re-parameterization step		600		20,000		200,000
Reparameterization period (P) schedule (epoch range: P)	1 - 25:	100	1 - 25:	100	1 - 25:	1000
	26 - 50:	200	26 - 80:	200	26 - 50:	2000
	51 - 75:	400	81 - 140:	400	51 - 75:	4000
	76 - 100:	800	141 - 200:	800	76 - 100:	8000
Hyperparameters for Deep Rewiring (*DeepR*) (Bellec et al., 2017)						
L^1 regularization multiplier (α)		10^{-4}		10^{-5}		10^{-5}
Temperature (T) schedule (epoch range: T)	1 - 25:	10^{-3}	1 - 25:	10^{-5}	1 - 25:	10^{-5}
	26 - 50:	10^{-4}	26 - 80:	10^{-8}	26 - 50:	10^{-8}
	51 - 75:	10^{-5}	81 - 140:	10^{-12}	51 - 75:	10^{-12}
	76 - 100:	10^{-6}	141 - 200:	10^{-15}	76 - 100:	10^{-15}

Table 2: Test accuracy% (top-1, top-5) of Resnet-50 on Imagenet for *dynamic sparse* vs. *HashedNet*. Numbers in square brackets are differences from the *full dense* baseline.

Final global sparsity (# Parameters)	0.8 (7.3M)		0.9 (5.1M)	
HashedNet	70.0 [-4.9]	89.6 [-2.8]	66.9 [-8.0]	87.4 [-5.0]
Dynamic sparse (ours)	**73.3** [**-1.6**]	**92.4** [**0.0**]	**71.6** [**-3.3**]	**90.5** [**-1.9**]

Table 3: Representative examples of training methods that yield "sparse" deep CNNs

Method	**Strict parameter budget throughout training and inference**	**Granularity of sparsity**	**Automatic layer sparsity**
Dynamic Sparse Reparameterization (Ours)	yes	non-structured	yes
Sparse Evolutionary Training (SET) (Mocanu et al., 2018)	yes	non-structured	no
Deep Rewiring (DeepR) (Bellec et al., 2017)	yes	non-structured	no
NN Synthesis Tool (NeST) (Dai et al., 2017; 2018)	no	non-structured	yes
`tf.contrib.model_pruning` (Zhu & Gupta, 2017)	no	non-structured	no
RNN Pruning (Narang et al., 2017)	no	non-structured	no
Deep Compression (Han et al., 2015)	no	non-structured	no
Group-wise Brain Damage (Lebedev & Lempitsky, 2015)	no	channel	no
L^1-norm Channel Pruning (Li et al., 2016)	no	channel	no
Structured Sparsity Learning (SSL) (Wen et al., 2016)	no	channel/kernel/layer	yes
ThiNet (Luo et al., 2017)	no	channel	no
LASSO-regression Channel Pruning (He et al., 2017)	no	channel	no
Network Slimming (Liu et al., 2017)	no	channel	yes
Sparse Structure Selection (SSS) (Huang & Wang, 2017)	no	layer	yes
Principal Filter Analysis (PFA) (Suau et al., 2018)	no	channel	yes/no

We provide examples of different categories of methods. This is not a complete list of methods.

Table 4: Test accuracy% (top-1, top-5) of Resnet-50 on Imagenet for different levels of granularity of sparsity. Numbers in square brackets are differences from the *full dense* baseline.

Final overall sparsity (# Parameters)	0.8 (7.3M)		0.9 (5.1M)	
Thin dense	72.4 [-2.5]	90.9 [-1.5]	70.7 [-4.2]	89.9 [-2.5]
Dynamic sparse (kernel granularity)	72.6 [-2.3]	91.0 [-1.4]	70.2 [-4.7]	89.8 [-2.6]
Dynamic sparse (non-structured)	**73.3** [**-1.6**]	**92.4** [**0.0**]	**71.6** [**-3.3**]	**90.5** [**-1.9**]