Full deep neural network training on a pruned weight budget

Maximilian Golub [1 2] **Guy Lemieux** [1] **Mieszko Lis** [1]

Abstract

We introduce a DNN training technique that learns only a fraction of the full parameter set without incurring an accuracy penalty. To do this, our algorithm constrains the total number of weights updated during backpropagation to those with the highest total gradients. The remaining weights are not tracked, and their initial value is regenerated at every access to avoid storing them in memory. This can dramatically reduce the number of off-chip memory accesses during both training and inference, a key component of the energy needs of DNN accelerators. By ensuring that the total weight diffusion remains close to that of baseline unpruned SGD, networks pruned using our technique are able to retain state-of-the-art accuracy across network architectures — including networks previously identified as difficult to compress, such as Densenet and WRN. With ResNet18 on ImageNet, we observe an 11.7× weight reduction with no accuracy loss, and up to 24.4× with a small accuracy impact.

1 Introduction

On-device inference has become an important market, and a number of vendors offer low-power deep neural network accelerators that reduce computation costs to specifically target mobile and embedded applications (Intel, 2017; Samsung, 2018; Qualcomm, 2017; Kingsley-Hughes, 2017). To enable inference despite the comparatively smaller memories and reduced memory bandwidths in mobile devices — an order of magnitude less capacity and two orders of magnitude less bandwidth than a datacentre-class GPU — many researchers have proposed pruning, quantizing, and compressing neural networks, often trading off accuracy versus network size (LeCun et al., 1990; Hassibi et al., 1993; Han et al., 2015; 2016b; Wu et al., 2015; Choi et al., 2016; Alvarez & Salzmann, 2017; Ge et al., 2017; Zhou et al., 2017; Luo et al., 2017; Yang et al., 2017, etc.).

Comparatively little attention, however, has been paid to the problem of on-device *training*, which has so far been limited to simple models (e.g., Apple, 2017). Training takes much more energy than inference: one iteration on a single sample involves roughly thrice as many weight accesses, and typically many iterations on many samples are required. Off-chip memory accesses dominate, costing hundreds of times more energy than computation: in a 45nm process, for example, accessing a 32-bit value from DRAM costs over 700× more energy than an 32-bit floating-point compute operation (640pJ vs. 0.9pJ, Han et al., 2016a), with an even larger gap at smaller process nodes. Existing pruning, quantization, and reduction techniques are only applied *after* training, and do not ameliorate this energy bottleneck.

Most of the off-chip memory references during training come from accessing activations and weights. The ratio depends on the amount of weight reuse available in the DNN architecture being trained: for example, in an MLP weights are not reused within a single training sample and typically need more bandwidth than activations, while in a CNN each filter is reused for an entire layer and there can be an order of magnitude more activation accesses than weight accesses.

Several prior works have been able to reduce the footprint of activations by an order of magnitude, via techniques like gradient checkpointing (Chen et al., 2016), accelerator architectures that elide zero-valued or small activations (Albericio et al., 2016; Judd et al., 2017), and quantization during training (Jain et al., 2018). In any case, low-end devices with fewer compute resources may not even benefit from weight reuse across batches, as training on small batches and single images is often effective (Masters & Luschi, 2018).

In this paper, therefore, we focus on reducing the number of *weights* that must be stored during the training process. This is possible because deep networks have many more parameters than are needed to capture the intrinsic complexity of the tasks they are being asked to solve (Li et al., 2018). Importantly, training only a fraction of the weights to change naturally results in a pruned model *without* the need to refine the pruned weight set via additional training.

Our algorithm, Dropback, is a training-time pruning technique that greedily selects the most promising subset of

[1]Department of Electrical Engineering, The University of British Columbia, Vancouver, Canada [2]Mercedes-Benz Research & Development North America, Seattle, WA, USA. Correspondence to: Mieszko Lis <mieszko@ece.ubc.ca>.

Proceedings of the 2nd SysML Conference, Palo Alto, CA, USA, 2019.

weights to train. It relies on three observations:

(a) that the parameters that have accumulated the highest total gradients account for most of the learning,

(b) that the values of these parameters are predicated on the initialization values chosen for the remaining parameters, and

(c) that initialization values can be cheaply recomputed on-the-fly without needing to access memory.

In contrast to prior pruning techniques, which train an unconstrained network, prune it, and retrain the pruned network, Dropback prunes the network immediately at the start of training and does not require a retraining pass. Only the weights that remain require memory, and most of the weight set is never stored. Because Dropback recomputes initialization parameters, it can prune layers like batch normalization or parametric ReLU, which are not pruned in existing approaches.

Dropback outperforms best-in-class pruning-only methods (which require retraining) on network architectures that are already dense and have been found particularly challenging to compress (Liu et al., 2017; Louizos et al., 2017; Li et al., 2017). On Densenet, we achieve 5.86% validation error with 4.5× weight reduction (vs. best prior 5.65% / 2.9× reduction), and on WRN-28-10 we achieve 3.85%–4.20% error with 4.5×–7.3× weight reduction (vs. best prior 3.75% uncompressed and best pruned prior 16.6% / 4× reduction). Using ResNet18 on ImageNet, we observe an 11.7× weight reduction without accuracy loss, and up to 24.4× with a small accuracy impact. Like other pruning techniques, it is orthogonal to, and can be combined with, quantization and value compression.

2 Dropback: training pruned networks

2.1 Approach and key insights

Existing deep neural network pruning techniques rely on the observation that most tasks can be solved using many fewer parameters than the total size of the model (Li et al., 2018) — in other words, many parameters do not contribute useful information to the final output, and can be removed. Because the exact values of the surviving weights still depend on the values of the zeroed parameters, the network must still be retrained; only after retraining is it possible to recover close to original accuracy with an order of magnitude fewer weights (Han et al., 2016b; Zhu et al., 2017; Ge et al., 2017; Masana et al., 2017; Luo et al., 2017; Ullrich et al., 2017; Yang et al., 2017, etc.). While they reduce the energy expended on memory accesses during inference, they require retraining and therefore actually *increase* the number of energy-consuming memory accesses during training.

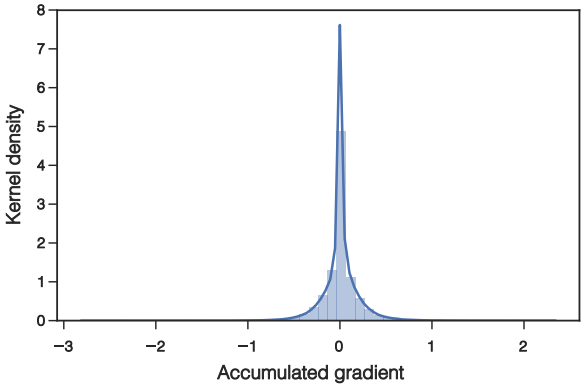

Figure 1. Distribution of accumulated gradients over 100 epochs of standard SGD training on MNIST using a 90,000-weight MLP.

Dropback has a different purpose: we aim to reduce the memory footprints both *during* and *after* training, and so we must reduce the *number of parameters tracked at training time*. To choose which parameters to track, we are guided by three related but slightly different insights, described below.

Track the highest accumulated gradients. As post-training pruning techniques delete weights with the lowest *values*, it may seem natural to keep track of gradients for the weights with the highest values. However, this naïve approach is not effective during the first few training iterations. Informally, this is because the initial value of each weight, typically drawn from a scaled normal distribution (LeCun et al., 1998b), serves as scaffolding which gradient descent can amplify to train the network. To overcome this, pruning algorithms typically retrain the network after the lowest-value weights have been removed (Han et al., 2016b; Zhu et al., 2017; Ge et al., 2017; Masana et al., 2017; Luo et al., 2017; Ullrich et al., 2017; Yang et al., 2017, etc.).

Instead, our approach is to track gradients for a fixed number of weights which have learned the most overall — that is, the weights with the *highest accumulated gradients*. Figure 1 shows that most weights move very little from their initial values and most accumulated gradients are near 0, suggesting that we only need to track gradients for a small fraction of the weights — *provided* we keep the remaining weights at their initial values.

Note that the set of weights with the highest accumulated gradient may (and does) change as the training process explores different subspaces. Dropback allows weights to enter the tracked set if their gradient exceeds the accumulated gradient of a currently-tracked weight; in that case, the weight with the lowest accumulated gradient is evicted from the tracked set and its value is reset to its initialization value.

With untracked weights kept at their initialization values, this approach results in a weight diffusion that is very close

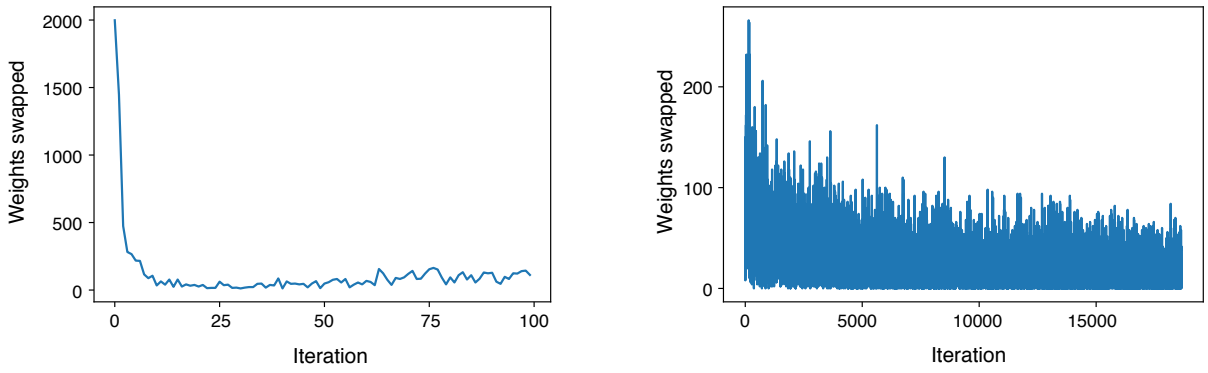

Figure 2. The number of weights added/removed to the top-2K gradient set in the first 10 mini-batches (left) and the remaining mini-batches (right) on MNIST. (Note different y-axis scales).

to that of the baseline SGD, a key factor in maintaining good generalization under different training regimes (Hoffer et al., 2017).

Recompute initialization-time values for untracked weights. While it may be intuitive to simply set untracked weights to zero, we observed that preserving the scaffolding provided by the initialization values is critical to the accuracy of the trained network. In our experiments on MNIST, we were able to reduce the tracked weights 60× if initialization values were preserved, but only 2× if untracked weights were zeroed. This is in line with the fact that prior pruning approaches require retraining after a subset of weights have been set to zero (Han et al., 2016b; Zhu et al., 2017; Ge et al., 2017; Masana et al., 2017; Luo et al., 2017; Ullrich et al., 2017; Yang et al., 2017, etc.).

However, storing the initialization-time values would require accessing off-chip memory to retrieve all of them during both the forward and backward passes of training — a costly proposition when a memory access consumes upwards of 700× more energy than a floating-point operation.

To avoid storing these weights, we observe that in practice the initial values are generated using a pseudo-random number source that is initialized using a single seed value and post-processed to fit a scaled normal distribution. Because each value only depends on the seed value and its index, it can be deterministically regenerated exactly when it is needed for computation, without ever being stored in memory.[1] For example, recomputing a normally distributed pseudo-random initialization value using the xorshift algorithm (Marsaglia, 2003) requires six 32-bit integer operations and one 32-bit floating point operation; this amounts to about 1.5pJ in a 45nm process, 427× less energy than a single off-chip memory access.

Regenerating untracked parameters to their initial values also works out-of-the-box for layers like Batch Normalization or Parametric ReLU, where the initialization strategy is typically a constant value (xorshift is not used for these). These layers are also pruned by Dropback, which to the best of our knowledge is unique.

Freeze the set of tracked weights after a few epochs. During training, gradients are still *computed* for the untracked weights, and those gradients can exceed the accumulated gradients of any weights that are being tracked; this is especially likely during the initial phases of training, when the optimization algorithm seeks the most productive direction. While the energy needed to compute the gradient is not significant, replacing the lowest tracked gradients with newly computed ones would require additional memory references, which in turn would expend additional energy.

Once the network has been trained for a few epochs, however, we would expect the accumulated gradients for the tracked weights to exceed any "new" gradients that could arise from the untracked weights. To verify this intuition, we trained a 90,000-parameter MLP on MNIST using standard SGD while keeping track of which parameters were in the top-2K gradients set. Figure 2 shows that the set of the highest-gradient weights stabilizes after the first few iterations. The "noise" of less than 0.04% of weights entering and leaving the highest-gradient set in the rest of the epochs remains throughout the training process, and has no effect on the final accuracy. This observation allows us to freeze the "tracked" parameter set after a small number of epochs, saving more energy-costly memory accesses.

2.2 The Dropback algorithm

Algorithm 1 shows the resulting Dropback training process.

[1] In a CPU or GPU, short-lived temporary values are stored in on-chip register files or scratchpads, reading which costs a fraction of the energy needed to access off-chip DRAM.

Algorithm 1: Dropback training. $N(0, \sigma)$ is generated from the xorshift pseudo-RNG. W_{trk} and W_{utrk} = tracked and untracked weights; T and U = tracked and untracked accumulated gradients; S = sorted accumulated gradients; k = number of gradients to track and λ = lowest tracked cumulative gradient; α = learning rate. *mask* indicates a boolean matrix with the same shape as the weights. $\overline{mask}$ indicates the logical inverse of the mask.

Initialization: $W^{(0)}$ with $W^{(0)} \sim N(0, \sigma)$
Output: $W^{(t)}$
while *not converged* **do**
  $T = \left\{ \left| \sum_{i=0}^{t-1} \frac{\alpha \partial f\left(W^{(i-1)}; x^{(i-1)}\right)}{\partial w} \right| \text{ s.t. } w \in W_{trk} \right\}$
  if not frozen:
    $U = \left\{ \left| \frac{\alpha \partial f\left(W^{(i-1)}; x^{(i-1)}\right)}{\partial w} \right| \text{ s.t. } w \in W_{utrk} \right\}$
  else:
    $U = \{\}$
  $S = \text{sort}(T \cup U)$
  $\lambda = S_k$
  $mask = \mathbb{1}(S > \lambda)$
  $W^{(t)} = mask \cdot \left(W^{(t-1)} - \alpha \nabla f\left(W^{(t-1)}; x^{(t-1)}\right)\right) + \overline{mask} \cdot W^{(0)}$
  $t = t + 1$
end

Weights are initialized from a scaled normal distribution (LeCun et al., 1998b), computed via the xorshift pseudo-random number generator (Marsaglia, 2003). In every iteration, gradients are computed and the highest k accumulated gradients are stored and carried to the next iteration.

For clarity of exposition, Algorithm 1 shows all gradients as recomputed and sorted to determine the highest-total-gradient set of k elements. In a practical implementation, however, the tracked accumulated gradient set is stored a priority queue of size k where the minimum elements are evicted when incoming gradients are higher than the stored minimum. Note that it suffices to store only the tracked set T: W^t can be computed from T and $W^{(0)}$, and $W^{(0)}$ can be determinstically regenerated when needed using the weight index and the xorshift random-number generator.

After a manually chosen iteration cutoff, the tracked set is "frozen" and gradients are only computed and updated for the weights already tracked; this saves additional computation time and energy. As in standard stochastic gradient descent, training ends once the network is considered to have converged.

Dropback versus sparsity-based regularization. Dropback can be contrasted with techniques like Dropout (Srivastava et al., 2014) or DSD (Han et al., 2017), which *temporarily* restrict the gradients that can be updated, essentially as a regularization scheme. Dropout restricts randomly selected gradient updates during each training iteration, while DSD repeatedly alternates sparse phases (where the lowest-absolute-value weights are deleted) and dense refinement phases (where all weights may be updated). In both cases, the training process is allowed to update all weights much of the time, even though a only a subset of weights may update in a specific phase of training.

In contrast, Dropback limits the set of weights that can be updated *throughout* the entire training process — there is never a phase where non-tracked weights are updated. Weights that have not substantially contributed to the overall optimization gradient are not trained, and retain their initialization values. Because the final performance of a Dropback-trained network depends only on the initialization values and the accumulated gradients of the tracked subset of weights, retraining is not needed.

3 Experiments

3.1 Methods

We implemented Dropback using the Chainer deep neural network toolkit (Tokui et al., 2015); models were trained on an NVIDIA 1080Ti GPU. We compared Dropback to a baseline implementation without any pruning, as well as three representative pruning techniques:

(a) a straightforward magnitude-based pruning implementation where only the highest weights are kept after each iteration,

(b) variational dropout (Kingma et al., 2015), which can progressively remove weights during training, and

(c) network slimming (Liu et al., 2017), a modern train-prune-retrain pruning method that achieves state-of-the-art results on modern network architectures.

All networks were optimized using stochastic gradient descent with momentum, as all other optimization strategies cost significant extra memory.

We evaluated all techniques on the MNIST (LeCun, 1998), CIFAR-10 (Krizhevsky, 2009), and ImageNet (Russakovsky et al., 2015) datasets. For MNIST, we used both LeNet-300-100 (LeCun et al., 1998a) and a simpler network with only 100 hidden units, which we refer to as MNIST-100-100. For CIFAR-10, we used three networks: Densenet (Huang et al., 2016), WRN-28-10 (Zagoruyko & Komodakis, 2016), and VGG-S, a reduced VGG-16-like model with dropout, batch normalization, and two FC layers of 512 neurons including the output layer (a total of 15M parameters vs. the 138M of

MNIST-300-100	Validation Error	Weight reduction	Best Epoch	Freeze Epoch
Baseline 267K	1.41%	0×	65	N/A
Dropback 50K	1.51%	5.33×	24	100
Dropback 5K	2.58%	53.32×	32	20
Dropback 1.5K	3.84%	177.74×	97	40
MNIST-100-100	**Validation Error**	**Weight reduction**	**Best Epoch**	**Freeze Epoch**
Baseline 90K	1.70%	0×	47	N/A
Dropback 50K	1.58%	1.8×	24	5
Dropback 20K	1.70%	4.5×	32	5
Dropback 1.5K	3.78%	60×	26	30

Table 1. The MNIST digit dataset using LeNet-300-100 (top) and MNIST-100-100, a smaller MLP with 100 hidden neurons (bottom). Dropback 50K refers to a configuration where 50,000 gradients are retained during training, Dropback 5K refers to a configuration with 5,000 retained gradients, and so on.

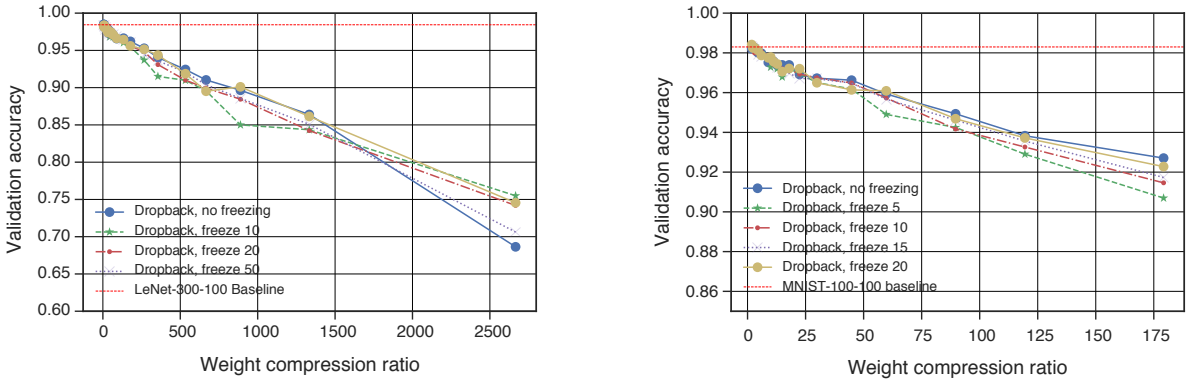

Figure 3. Tracked parameter tradeoff vs. network accuracy, with several parameter selection freezing points for LeNet-300-100 (left) and MLP-100-100 (right) on MNIST. Freeze 0 represents a network trained without freezing.

VGG-16). We specifically chose Densenet and WRN because they represent modern network architectures that are very challenging to prune with existing techniques (Liu et al., 2017; Louizos et al., 2017; Li et al., 2017). For ImageNet, we trained the ResNet18 model (He et al., 2016); as at the time of writing we have been unable to access the test set evaluation server for ImageNet, we report all results on the validation set.

In what follows, we report reductions in network size as the ratio of the unpruned weight count to the number of weights tracked during training. Note that this number refers to weight pruning only, and does *not* include other reduction techniques such as quantization or compression, which are orthogonal to our approach.

3.2 MNIST digit recognition

We first evaluated Dropback on the MNIST handwritten digits dataset using a small multi-layer perceptron (MLP) with approximately 90,000 weights, as well as the LeNet-300-100 MLP, which has approximately 266,600 weights. Training was allowed for up to 100 epochs, and the initial learning rate of 0.4 was cut in half four times during training. The best epoch was chosen as the epoch with the highest validation accuracy after five epochs of no improvement.

Weight reduction and accuracy. Table 1 shows the results for the baseline (unpruned) model and three configurations of Dropback, retaining respectively 50,000 weights (1.8× reduction), 20,000 weights (4.5× reduction), and 1,500 weights (60× reduction). With MNIST-100-100 and a modest 2× reduction in weights, Dropback slightly exceeds the accuracy of the baseline model. This matches the trend reported in train-prune-retrain work such as DSD (Han et al., 2017), where a sparse model that omits 30%–50% weights outperforms the baseline dense (all-weights) model. The larger MLP sees a slight drop in accuracy, but at 50,000 tracked weights is compressed many more times, and reaches maximum accuracy nearly 3× faster. Further reducing the model to 20,000 weights (4.5× reduction) results in nearly the same accuracy as the baseline, and convergence in a comparable number of epochs.

layer	Dropback 1500	Dropback 10000	Baseline
fc1 (100×784)	734 (52.3%)	7223 (72.2%)	78500 (87.6%)
fc2 (100×100)	512 (34.1%)	2128 (21.3%)	10100 (11.3%)
fc3 (100×10)	254 (16.9%)	549 (5.5%)	1010 (1.1%)
Total	1500	10000	89610

Table 2. Number of gradients for each layer retained in the final trained MNIST network

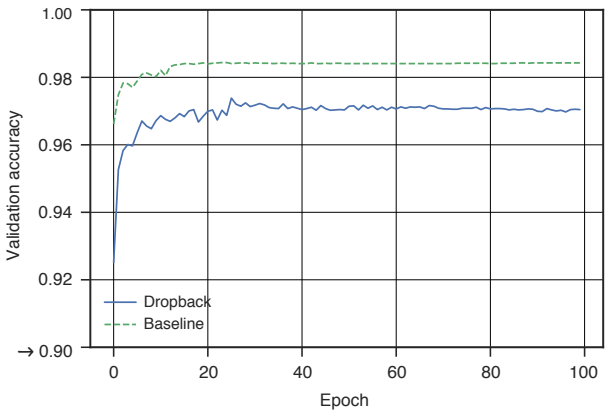

Figure 4. Rate of convergence for LeNet-300-100 for our technique our and the baseline model. Note that the y-axis starts as 0.90, and the accuracies are within 1% of each other.

We also investigated an extreme reduction configuration with weight storage reduced drastically to 1,500 weights. Not surprisingly, the error rate increases (over 2×) and twice the number of epochs are needed to achieve convergence. Still, the nearly 60× reduction in the weight count for the smaller model and 177× for the larger model potentially offer an attractive design point for low-power embedded accelerators in future mobile and edge devices.

Tracked weight set freezing. Figure 3 shows the effect of different freezing times for the tracked gradient set versus the tracked gradient counts. For both MNIST networks, freezing sooner to reduce the computational overhead results in lower achieved accuracy — especially for very high reduction ratios — but for smaller reduction ratios freezing early has little effect on the overall accuracy. Figure 4 shows the rate of convergence for Dropback and the baseline on the LeNet-300-100 network; despite different tracked parameter counts, both methods have similar convergence behaviour.

Retained weight distribution. Table 2 shows that the number of parameters retained per layer varies depending on the number of tracked weights. The smaller Dropback 1.5K network allocates a much higher amount of its weights to the later layers compared to the Dropback 10K network and the baseline — an indication that, in the smaller network, proportionally more neurons in the later layers are critical for decision-making.

3.3 CIFAR-10 image classification

We also studied the training performance of Dropback using VGG-S, Densenet, and WRN-28-10 on the CIFAR-10 dataset. This is a much more challenging task than MNIST, and the networks were specifically selected for being already relatively dense. Models were trained for 300 epochs on VGG-S and 500 epochs on Densenet and WRN; the best epoch was selected. No data augmentation of CIFAR-10 was performed. The learning rate was initialized to 0.4 and decayed 0.5× every 25 epochs.

Weight reduction and accuracy. Table 3 shows how Dropback compares to variational dropout, network slimming, and magnitude-based pruning on VGG-S, Densenet, and WRN-28-10. Overall, Dropback is able to achieve comparable (or even slightly improved) accuracy on VGG-S and Densenet with five-fold weight reduction, and up to 20×–30× if some accuracy is sacrificed. On WRN-28-10, Dropback achieves 5× and 7× reduction with less than 0.5% accuracy drop.

Note that these networks are challenging, as they are already quite dense for the accuracy level they achieve. Variational dropout works only on VGG-S, and fails to converge on Densenet and WRN due to gradient explosion. Magnitude-based pruning does not achieve better accuracy than Dropback despite less weight reduction. Finally, network slimming achieves slightly better top accuracy on Densenet with 50% less reduction, but results in dramatic accuracy loss when applied to WRN; this corresponds to recent work which has shown that WRN is hard to compress more than about 2× without losing significant accuracy (Liu et al., 2017; Louizos et al., 2017; Li et al., 2017). Dropback, in contrast, is universally able to achieve weight reduction on the order of 5× with little to no accuracy loss.

Effects of freezing. Figure 5(right) shows how freezing the tracked parameter set after 10, 20, and 50 epochs affects the reduction and accuracy on VGG-S. At under 10× weight pruning, accuracy does not suffer against the uncompressed baseline, and freezing does not affect either reduction or

CIFAR-10	Validation error	Weight reduction	Best epoch	Freeze epoch
Baseline 15M	10.08%	0×	214	N/A
VGG-S Dropback 5M	9.75%	3×	127	5
VGG-S Dropback 3M	9.90%	5×	128	20
VGG-S Dropback 0.75M	13.49%	20×	269	35
VGG-S Dropback 0.5M	20.85%	30×	201	15
VGG-S Var. Dropout	13.50%	3.4×	200	N/A
VGG-S Mag. Pruning .80	9.42%	5.0×	182	N/A
VGG-S Slimming	11.08%	3.8×	196	N/A
Densenet Baseline 2.7M	6.48%	0×	382	N/A
Densenet Dropback 600K	5.86%	4.5×	409	N/A
Densenet Dropback 100K	9.42%	27×	307	N/A
Densenet Var. Dropout	FAIL	FAIL	FAIL	FAIL
Densenet Mag. Pruning .75	6.41%	4.0×	480	N/A
Densenet Slimming	5.65%	2.9×	N/A	N/A
WRN-28-10 Baseline 36M	3.75%	0×	326	N/A
WRN-28-10 Dropback 8M	3.85%	4.5×	384	N/A
WRN-28-10 Dropback 7M	4.02%	5.2×	417	N/A
WRN-28-10 Dropback 5M	4.20%	7.3×	304	N/A
WRN-28-10 Var. Dropout	FAIL	FAIL	FAIL	FAIL
WRN-28-10 Mag. Pruning .75	26.52%	4×	109	N/A
WRN-28-10 Slimming .75	16.640%	4×	173	N/A

Table 3. Validation accuracy and reduction ratios on CIFAR-10. FAIL = training did not converge due to gradient explosion.

accuracy. With more reduction, however, the tracked gradient set struggles to maintain accuracy, and freezing the gradients early results in substantial accuracy drops at 30× reduction.

Convergence. Figure 5(left) shows that Dropback initially learns slightly more slowly than the uncompressed baseline, but exhibits the same convergence behaviour after about 20 epochs (VGG-S). Variational dropout, in contrast, learns more quickly initially but converges on a substantially lower accuracy.

3.4 ImageNet image classification

Finally, we studied whether Dropback can be applied in the more realistic setting of the ImageNet dataset. This dataset is substantially more challenging than CIFAR-10 — there are 1,000 instead of 10 categories and the images are 64× larger — and represents the high end of tasks for which inference is possible in a mobile-device setting.

ResNet18 was trained for 100 epochs using stochastic gradient descent with momentum, and the best epoch was selected. The initial learning rate of 0.2 was decayed by 0.97× in each epoch. Freezing was not applied during Dropback training, and no data augmentation was performed.

Weight reduction and accuracy. Table 4 and Figure 6(left) show top-1 results on the ImageNet dataset using the ResNet18 network (11 million weights uncompressed) for different weight reduction ratios. Dropback is able to match or improve the accuracy of the baseline uncompressed network with weight reduction ratios of up to 11.7×, and can reduce the weight count over 23× with a modest drop in accuracy (39.5% vs 31.6% baseline). In comparison, magnitude-based pruning stops improving after epoch 6 and never matches baseline accuracy: even a modest 4× reduction in weight count incurs the same cost in accuracy as Dropback pays for a 23.4× reduction.

Convergence. Figure 6(right) shows that Dropback learns at the same rate as the uncompressed baseline up to 5.85× weight reduction. At 11.7× weight reduction, the model initially learns more slowly and takes substantially longer to converge, but eventually matches the baseline accuracy.

4 Discussion

To investigate why Dropback consistently achieves good reduction/accuracy tradeoffs across a wide range of networks, we considered the training process analysis due to Hoffer et al. (2017). Briefly, they observe that when DNNs are trained using SGD, the ℓ^2 distance of weights $\mathbf{w}_t$ at time t from the initial weights $\mathbf{w}_0$ grows logarithmically, i.e., $\|\mathbf{w}_t - \mathbf{w}_0\| \sim \log t$ (this is the same as the gradient accumulated until time t). They therefore model SGD as a random

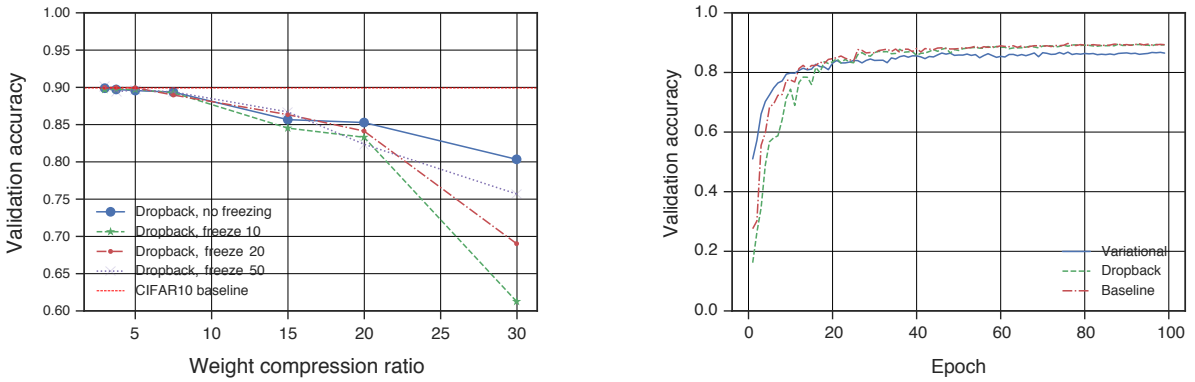

Figure 5. VGG-S CIFAR-10. Left: tracked parameter tradeoff vs. network accuracy, with several parameter selection freezing points (note the y-axis range). Right: epoch vs. validation accuracy for our method at 5M tracked parameters, variational dropout, and the baseline model.

ResNet18	Validation error	Weight reduction	Best epoch
Baseline 11M	31.59%	1.00×	26
DropBack 4M	30.31%	2.92×	41
DropBack 2M	29.95%	5.85×	44
DropBack 1M	32.01%	11.69×	49
DropBack 0.5M	39.53%	23.39×	47
Var. Dropout	FAIL	FAIL	FAIL
Mag. Pruning 2.75M	38.43%	4×	6

Table 4. Validation accuracy and reduction ratio of Dropback applied to ResNet18 on the ImageNet dataset. Dropback is able to reduce the weight count by 11.7× without accuracy loss, and nearly 24× with some degradation. In comparison, magnitude-based pruning at 4× reduction suffers the same loss as Dropback at 23.4×. All of our Variational Dropout experiments on ResNet18 failed due to numerical instability.

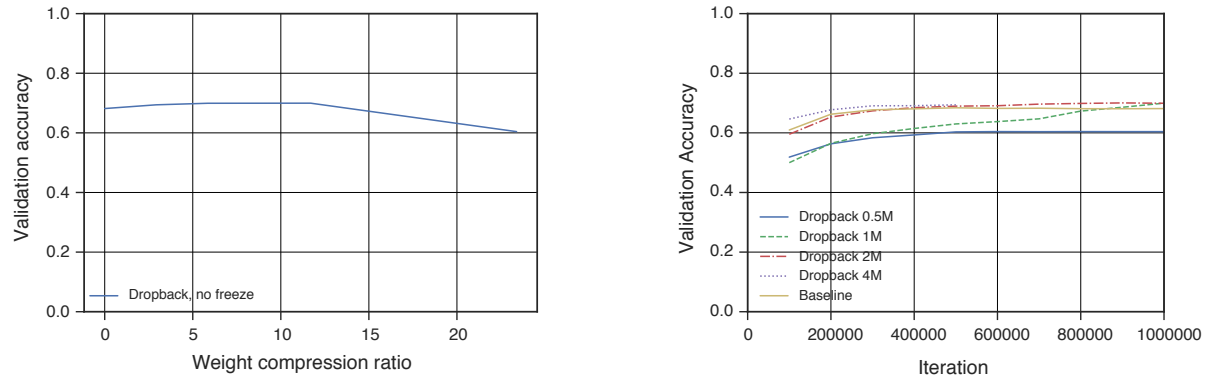

Figure 6. ResNet18 on ImageNet. Left: tracked parameter tradeoff vs. validation accuracy of the trained network. Right: epoch vs. validation accuracy at different weight reduction ratios, as well as the uncompressed baseline (which has 11M weights).

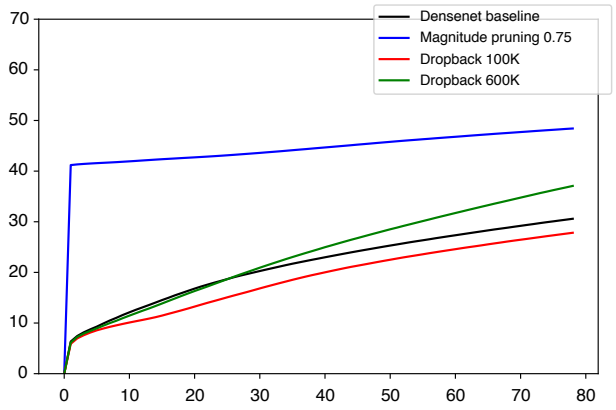

Figure 7. Diffusion (ℓ^2) distance vs. training time for Densenet on CIFAR-10. Unlike magnitude-based pruning, Dropback weight evolution follows an ℓ^2 distance profile very similar to that of the uncompressed baseline throughout the training process.

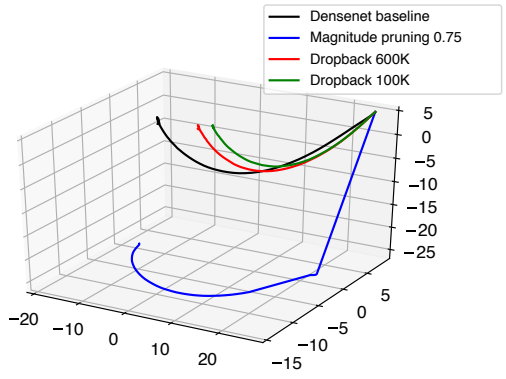

Figure 8. Evolution of weights under SGD projected into 3D space using PCA, training Densenet on CIFAR-10. To clearly show the training process, all projections were translated to align at their initial points. Unlike magnitude-based pruning, Dropback training remains close to the uncompressed baseline.

walk on a random potential surface, which exhibits the same logarithmic distance effect (known as ultra-slow diffusion). Hoffer et al. (2017) argue that SGD configurations (for them, batch sizes) that preserve the ultra-slow diffusion effect result in models that generalize well.

We reasoned that Dropback maximally preserves the ℓ^2 diffusion distance of the baseline training scheme because (a) Dropback tracks the highest gradients, and (b) most of the remaining gradients are very close to zero (cf. Figure 1). To verify this intuition, we measured the diffusion distance for the baseline uncompressed network, Dropback, and magnitude-based pruning, all on the Densenet network classifying CIFAR-10.[2] Figure 7(left) shows that under Dropback weights diffuse similarly to the baseline training scheme, with the overall ℓ^2 distance negligibly lower because the untracked weights remain at their initialization values.

To visualize how the weight values evolve under Dropback compared to the baseline and pruning, we projected the parameter space down to 3D using principal component analysis (PCA). We repeated the experiment for Dropback, baseline, and magnitude-based pruning; for each run, we saved weight values every 100 batches, and fit the weight history to a 3D PCA space. While each run starts with exactly the same initial weights, the PCA process maps these to different points in the 3D space depending on the full weight history; to clearly show the divergence of the training process, we translated all of the lines to align at the same initialization point in the post-PCA 3D space as well.

Figure 8 shows that under Dropback, the principal components of the trained weight vector stay very close to those of the baseline-trained weight vector, whereas that of magnitude-based pruning diverges significantly. If we imagine the training path of the baseline uncompressed configuration to be optimal, Dropback results in a close-to-optimal evolution.

[2]Network slimming, being a train-prune-retrain technique, is not amenable to this type of analysis, and variational dropout failed to converge on that task.

5 Related Work

5.1 Parameter and optimization landscape studies

The idea that deep neural networks have more parameters than is necessary to solve the task at hand has been well established (Dauphin & Bengio, 2013; Denil et al., 2013). This observation has driven the development of pruning techniques (see below), and Dropback relies on this insight as well.

Dropback is also inspired by studies on optimization landscapes. Dauphin et al. (2014) demonstrated that local critical points tend to be saddle-points, suggesting that optimization gradients are productive along many dimensions even at the critical point. Optimization paths that lead from the initial point to the optimum are often direct (Goodfellow et al., 2015); this suggests that an algorithm like Dropback that explores only a subspace of the optimization domain has a good chance of arriving at the optimum.

Closest in spirit to Dropback is the study by Li et al. (2018), which trained a limited-dimensionality subspace of a full model and used random matrices to project the training subspace to the full model in an approach reminiscent of compressive sensing. Because the dimensions of the training subspace are *not* a subset of the dimensions of the full parameter space, however, the subspace must still be projected to the full model for evaluation. Unfortunately, this approach is not suitable for training with limited resources, as it actually

increases the storage required and substantially increases the computation required for training (Li et al., 2018). Dropback shares the idea of training only a subspace, but instead actively selects a productive subset of the dimensions of the original parameter space, making it possible to reduce both storage and compute requirements.

5.2 Pruning

Pruning networks to enforce sparsity after training has been effective in Han et al. (2016b); Zhu et al. (2017); Ge et al. (2017); Masana et al. (2017); Luo et al. (2017); Ullrich et al. (2017); LeCun et al. (1990); Srivastava et al. (2014); Wan et al.; Yang et al. (2017), while other work has focused on reducing the rank of the parameter matrices during training (Alvarez & Salzmann, 2017) for later reduction. In contrast to Dropback, pruning post-training requires a retraining step to regain lost accuracy, and both pruning and low rank constraints still require full dense backpropagation during the training phase, which is undesirable in embedded systems due to the extra memory and energy cost. Dropback requires no retraining steps, and during both training and inference only ever stores the small subset of weights that are tracked. As a result, it only incurs a small portion of the total memory access costs during the training process.

Other work has focused on pruning while training to either improve accuracy or to increase the eventual sparsity of the trained network. Zhu & Gupta (2017) gradually increase the number of weights masked from contributing to the network, while Molchanov et al. (2017) extend variational dropout (Kingma et al., 2015) with per-parameter dropout rates to increase sparsity. Babaeizadeh et al. (2016) inject random noise into a network to find and merge the most correlated neurons. Finally, Langford et al. (2009) decay weights every k steps (for a somewhat large value of k), inducing sparsity gradually. Unlike Dropback, all of these techniques require at least as much memory to train as the unpruned network initially, and all of them take longer to converge.

5.3 Quantization

Reducing storage costs is often approached through quantizing weights to smaller bit widths. This can be preformed either after training, as in Ge et al. (2017); Wu et al. (2015); Choi et al. (2016); Han et al. (2016b); Ullrich et al. (2017); Gysel (2016), during an iterative retraining process (Zhou et al., 2017), or during training, as in Cai et al. (2017); Zhou et al. (2016); Courbariaux et al. (2016); Rastegari et al. (2016); Gupta et al. (2015); Mishra et al. (2017); Hubara et al. (2016); Courbariaux et al. (2015); Simard & Graf (1994); Holt & Baker. Out of all of these methods, only Gupta et al. (2015); Mishra et al. (2017); Hubara et al. (2016); Courbariaux et al. (2015) use reduced precision *while training* to lower the train time storage costs; other methods other methods store the full, non-quantized weights during backpropagation just like the post-training methods. Quantization is orthogonal to, and has been combined with, pruning techniques (Han et al., 2016b); as such, it is also orthogonal to Dropback.

5.4 Other compression methods

A few compression methods do not naturally fall into the quantization categories. HashedNets (Chen et al., 2015) use a hash function to group neuron connections into buckets with the same weight value. In effect, this is a kind of quantization, where the quantized values are limited in number but each is a full 32-bit floating-point number. Huffman coding (Huffman, 1952) has also been used for compression: for example, Deep Compression (Han et al., 2016b) applies Huffman coding to a pruned and quantized network. Both HashedNets and Deep Compression require the full network to be trained first, so are not directly comparable to Dropback. Compression in general, however, is orthogonal to Dropback.

6 Conclusion

Deploying DNN inference to mobile and other low-power devices has become possible in part due to advances in pruning and quantization of trained DNN models. The goal of *training* on these devices has, however, remained elusive largely due to limitations on storage and bandwidth for both weights and activations.

In this paper, we focus on reducing the storage and bandwidth needed for weights. Dropback is a training-time pruning technique that reduces weight storage both *during* and *after* training by (a) tracking only the weights with the highest accumulated gradients, and (b) recomputing the remaining weights on the fly.

Because Dropback prunes exactly those weights that have learned the least, its weight diffusion profile during training is very close to that of standard (unconstrained) SGD, in contrast to other pruning techniques. This allows Dropback to achieve better accuracy and weight reduction than prior methods on dense modern networks like Densenet, WRN, and ResNet, which have proven challenging to prune using existing techniques: Dropback is able to reduce weight counts 5×–11× with no accuracy loss.

Perhaps most attractively, Dropback has the potential to dramatically reduce the memory footprint and memory bandwidth needed to store and access weights *during* training. Because of this, Dropback can potentially be used to train networks an order of magnitude larger than currently possible with typical hardware, or, equivalently, to train/retrain standard-size networks on limited-capacity hardware, something not possible with current training techniques.

References

Albericio, J., Judd, P., Hetherington, T., Aamodt, T., Jerger, N. E., and Moshovos, A. Cnvlutin: ineffectual-neuron-free deep neural network computing. In *ISCA*, 2016.

Alvarez, J. M. and Salzmann, M. Compression-aware Training of Deep Networks. In *NIPS*, November 2017. arXiv: 1711.02638.

Apple. An On-device Deep Neural Network for Face Detection. *Apple Machine Learning Journal*, November 2017.

Babaeizadeh, M., Smaragdis, P., and Campbell, R. H. NoiseOut: A simple way to prune neural networks. *arXiv:1611.06211 [cs]*, 2016.

Cai, Z., He, X., Sun, J., and Vasconcelos, N. Deep Learning with Low Precision by Half-wave Gaussian Quantization. *arXiv:1702.00953 [cs]*, February 2017. arXiv: 1702.00953.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training Deep Nets with Sublinear Memory Cost. *arXiv:1604.06174 [cs]*, April 2016. arXiv: 1604.06174.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., and Chen, Y. Compressing neural networks with the hashing trick. In *ICML*, 2015.

Choi, Y., El-Khamy, M., and Lee, J. Towards the Limit of Network Quantization. *arXiv:1612.01543 [cs]*, December 2016. arXiv: 1612.01543.

Courbariaux, M., Bengio, Y., and David, J.-P. Training deep neural networks with low precision multiplications. In *ICLR*, December 2015. arXiv: 1412.7024.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]*, February 2016. arXiv: 1602.02830.

Dauphin, Y., Pascanu, R., Gülçehre, Ç., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *CoRR*, 2014.

Dauphin, Y. N. and Bengio, Y. Big neural networks waste capacity. In *CoRR*, 2013.

Denil, M., Shakibi, B., Dinh, L., De Freitas, N., et al. Predicting parameters in deep learning. In *NIPS*, 2013.

Ge, S., Luo, Z., Zhao, S., Jin, X., and Zhang, X. Y. Compressing deep neural networks for efficient visual inference. In *ICME*, July 2017. doi: 10.1109/ICME.2017.8019465.

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. Qualitatively characterizing neural network optimization problems. In *ICLR*, 2015.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. *arXiv:1502.02551 [cs, stat]*, 2015.

Gysel, P. Ristretto: Hardware-Oriented Approximation of Convolutional Neural Networks. *arXiv:1605.06402 [cs]*, May 2016. arXiv: 1605.06402.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ISCA*, 2016a.

Han, S., Mao, H., and Dally, W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, October 2016b. arXiv: 1510.00149.

Han, S., Pool, J., Narang, S., Mao, H., Gong, E., Tang, S., Elsen, E., Vajda, P., Paluri, M., Tran, J., Catanzaro, B., and Dally, W. J. DSD: Dense-Sparse-Dense Training for Deep Neural Networks. In *ICLR*, 2017.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *ICNN*, 1993.

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In *CVPR*, June 2016.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *NIPS*, 2017.

Holt, J. L. and Baker, T. E. Back propagation simulations using limited precision calculations. In *IJCNN*.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, 2016.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv:1609.07061 [cs]*, 2016.

Huffman, D. A. A method for the construction of minimum-redundancy codes. *Proc. IRE*, 40(9):1098–1101, 1952. ISSN 0096-8390.

Intel. Intel® Movidius™ Myriad™ X VPU, 2017.

Jain, A., Phanishayee, A., Mars, J., Tang, L., and Pekhimenko, G. Gist: Efficient data encoding for deep neural network training. In *ISCA*, 2018.

Judd, P., Delmas, A., Sharify, S., and Moshovos, A. Cnvlutin²: Ineffectual-activation-and-weight-free deep neural network computing. *arXiv:1705.00125*, 2017.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. In *NIPS*, 2015.

Kingsley-Hughes, A. Inside Apple's new A11 Bionic processor. ZDNet, September 2017.

Krizhevsky, A. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Langford, J., Li, L., and Zhang, T. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.

LeCun, Y. The MNIST database of handwritten digits. Technical report, NEC Research Institute, 1998.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *NIPS*, 1990.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a. ISSN 0018-9219.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Effiicient BackProp. In *Neural Networks: Tricks of the Trade*, pp. 9–50, London, UK, UK, 1998b. Springer-Verlag. ISBN 3-540-65311-2.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. Measuring the intrinsic dimension of objective landscapes. In *ICLR*, 2018.

Li, H., De, S., Xu, Z., Studer, C., Samet, H., and Goldstein, T. Training Quantized Nets: A Deeper Understanding. *arXiv:1706.02379 [cs, stat]*, June 2017. arXiv: 1706.02379.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning Efficient Convolutional Networks through Network Slimming. *arXiv:1708.06519 [cs]*, August 2017. arXiv: 1708.06519.

Louizos, C., Welling, M., and Kingma, D. P. Learning Sparse Neural Networks through L_0 Regularization. *arXiv:1712.01312 [cs, stat]*, December 2017. arXiv: 1712.01312.

Luo, J.-H., Wu, J., and Lin, W. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *arXiv:1707.06342 [cs]*, July 2017. arXiv: 1707.06342.

Marsaglia, G. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.

Masana, M., van de Weijer, J., Herranz, L., Bagdanov, A. D., and Alvarez, J. M. Domain-adaptive deep network compression. *arXiv:1709.01041 [cs]*, September 2017. arXiv: 1709.01041.

Masters, D. and Luschi, C. Revisiting Small Batch Training for Deep Neural Networks. *arXiv:1804.07612 [cs, stat]*, 2018.

Mishra, A., Nurvitadhi, E., Cook, J. J., and Marr, D. WRPN: Wide reduced-precision networks. *arXiv:1709.01134 [cs]*, 2017.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. *arXiv:1701.05369 [cs, stat]*, 2017.

Qualcomm. Snapdragon Neural Processing Engine for AI, December 2017.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv:1603.05279 [cs]*, March 2016. arXiv: 1603.05279.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Samsung. Samsung Optimizes Premium Exynos 9 Series 9810 for AI Applications and Richer Multimedia Content, January 2018.

Simard, P. Y. and Graf, H. P. Backpropagation without multiplication. In *NIPS*, pp. 232–239, 1994.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Tokui, S., Oono, K., Hido, S., and Clayton, J. Chainer: a next-generation open source framework for deep learning. In *LearningSys*, 2015.

Ullrich, K., Meeds, E., and Welling, M. Soft Weight-Sharing for Neural Network Compression. In *ICLR*, February 2017. arXiv: 1702.04008.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. Regularization of neural networks using DropConnect. In *PMLR*, pp. 1058–1066.

Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. Quantized Convolutional Neural Networks for Mobile Devices. *arXiv:1512.06473 [cs]*, December 2015. arXiv: 1512.06473.

Yang, T.-J., Chen, Y.-H., and Sze, V. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *CVPR*, 2017.

Zagoruyko, S. and Komodakis, N. Wide Residual Networks. *arXiv:1605.07146 [cs]*, May 2016. arXiv: 1605.07146.

Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. In *ICLR*, February 2017. arXiv: 1702.03044.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv:1606.06160 [cs]*, June 2016. arXiv: 1606.06160.

Zhu, J., Jiang, J., Chen, X., and Tsui, C.-Y. SparseNN: An Energy-Efficient Neural Network Accelerator Exploiting Input and Output Sparsity. *arXiv:1711.01263 [cs]*, November 2017. arXiv: 1711.01263.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv:1710.01878 [cs, stat]*, 2017.