DropBack: Continuous Pruning During Training

Maximilian Golub **Guy Lemieux** **Mieszko Lis**
{mgolub2,lemieux,mieszko}@ece.ubc.ca

Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, BC

Abstract

We introduce a technique that compresses deep neural networks both during and after training by constraining the total number of weights updated during backpropagation to those with the highest total gradients. The remaining weights are forgotten and their initial value is regenerated at every access to avoid storing them in memory. This dramatically reduces 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 DropBack are able to maintain high accuracy across network architectures. We observe weight compression of 25× with LeNet-300-100 on MNIST while maintaining accuracy. On CIFAR-10, we see an approximately 5× weight compression on 3 models: an already 9× reduced VGG-16, Densenet, and WRN-28-10—all with zero or negligible accuracy loss. On Densenet and WRN, which are particularly challenging to compress, Both Densenet and WRN improve on the state of the art, achieving higher compression with better accuracy than prior pruning techniques.

1 Introduction

The past year has seen the arrival of on-device inference, with a number of vendors announcing 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 address the comparatively smaller memories 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, typically making a tradeoff between accuracy versus network size (Krizhevsky et al., 1990; Hassibi et al., 1993; Han et al., 2015b,a; Wu et al., 2015; Choi et al., 2016; Alvarez & Salzmann, 2017; Ge et al., 2017; Zhou et al., 2017; Luo 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 (Apple, 2017). Training is fundamentally limited by off-chip memory accesses, which cost 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., 2016), with an even larger gap at smaller process nodes. Existing pruning, quantization, and compression techniques are only applied *after* training, and do not ameliorate this energy bottleneck.

In this paper, we propose DropBack, a pruning technique that dramatically reduces the representation size and the number of memory accesses both *during* and *after* training. Our proposal relies on three

Preprint. Work in progress.

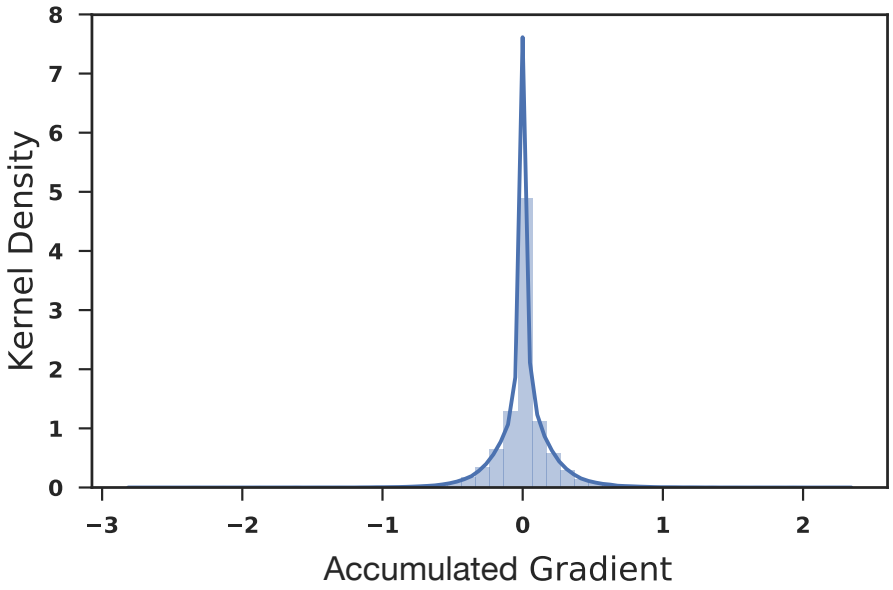

Figure 1: Distribution of accumulated gradients over 100 epochs of standard SGD training on MNIST using a 90,000-weight MLP.

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 recomputed on-the-fly without needing to access memory.

Unlike prior pruning techniques, which train an unconstrained network, prune it, and retrain the pruned network, DropBack prunes the network from the start of training and does not require a retraining pass. Also, it only needs enough weight memory to store the unpruned weights. Because DropBack recomputes initialization parameters, it can prune layers like batch normalization or parametric ReLU, which cannot be pruned using existing approaches.

DropBack outperforms best-in-class pruning 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×), 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 prior with pruning 16.6% / 4×).

2 DropBack: continuous pruning during training

2.1 Approach and key insights

Existing deep neural network pruning techniques rely on the observation that 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, but can recover close to original accuracy with an order of magnitude fewer weights (Han et al., 2015a; Zhu et al., 2017; Ge et al., 2017; Masana et al., 2017; Luo et al., 2017; Ullrich et al., 2017, etc.). While they reduce the energy expended on memory accesses during inference, they require retraining and therefore *increase* the number of energy-consuming memory accesses at training time.

DropBack has a different purpose: we aim to reduce the energy 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 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., 1998), serves as scaffolding which gradient descent can amplify to train the network.

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.

When untracked weights are kept at their initialization values (see below), this approach results in a weight diffusion speed that is very close 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 0, 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 0.

However, storing the initialization-time values would require accessing off-chip memory to retrieve them all during both the forward and backward passes of training—a costly proposition when a memory accesse 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 postprocessed 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] 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 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 and 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.

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

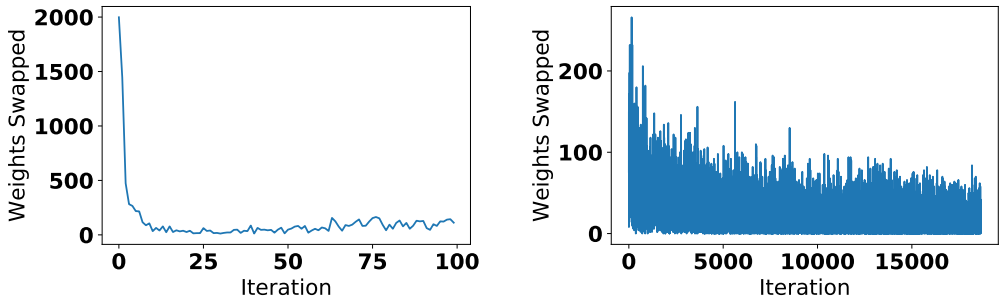

Figure 2: Number of weights added/removed to the top-2K gradient set in the first 10 mini-batches (left) and the remaining mini-batches (right) on MNIST. (Note different y-axis scales.)

2.2 The DropBack algorithm

Algorithm 1 shows the resulting DropBack training process. Weights are initialized from a scaled normal distribution (LeCun et al., 1998), generated by the xorshift pseudo-random number generator. For clarity of exposition, the listing recomputes all gradients and sorts them to determine the highest-total-gradient set of k elements; in a practical implementation the tracked accumulated gradient set is stored a priority queue of size k, with incoming gradients higher than the stored minimum evicting the minimum elements. After a manually chosen iteration cutoff, the tracked set is "frozen" and gradients are only updated for the weights already tracked; this saves additional computation time and energy. Note that he tracked set T requires no storage: its elements are recomputed when needed from $W^{t-1} - W^{(0)}$. As in standard stochastic gradient descent, training ends once the network is considered to have converged.

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 which weight are tracked using a boolean matrix with the same shape as the weight matrix.

Initialization: $W^{(0)}$ with $W^{(0)} \sim N(0, \sigma)$
Output: $W^{(t)}$
while *not converged* **do**
  $T = \left\{ \left| \sum_{i=0}^{t-1} \frac{\alpha \partial f(W^{(i-1)}; x^{(i-1)})}{\partial w} \right| \text{ s.t. } w \in W_{trk} \right\}$
  if not frozen:
    $U = \left\{ \left| \frac{\alpha \partial f(W^{(i-1)}; x^{(i-1)})}{\partial w} \right| \text{ s.t. } w \in W_{utrk} \right\}$
  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

Differences from sparsity-based regularization Note that DropBack is very different from techniques like Dropout (Srivastava et al., 2014) or DSD (Han et al., 2017), which *temporarily* restrict the gradients that can be updated as a regularization technique. Dropout restricts randomly selected gradient updates during each training iteration, while DSD repeatedly alternates sparse phases (where

MNIST-300-100	Validation Error	Weight Compression	Best Epoch	Freeze Epoch
Baseline 267k	1.41%	0×	65	N/A
DropBack 50k	1.51%	5.33×	24	100
DropBack 20k	1.78%	13.33×	77	35
DropBack 1.5k	3.84%	177.74×	97	40
MNIST-100-100	Validation Error	Weight Compression	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 to a configuration with 5,000 retained gradients, and so on.

the lowest-absolute-value weights are deleted) and dense refinement phases (where all weights may be updated). In contrast, DropBack specifically restricts updates to gradients that have not substantially contributed to the overall optimization gradient, and does not involve retraining phases on either the pruned or dense network.

3 Experiments

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 create weight sparsity 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 without momentum, as all other optimization strategies cost significant extra memory.

We evaluated all techniques on the MNIST (LeCun, 1998) and CIFAR-10 (Krizhevsky, 2009) datasets. For MNIST, we used both LeNet-300-100 (Lecun et al., 1998) 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 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).

MNIST. 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 exponentially reduced four times by a factor of 0.5. The best epoch was chosen by highest validation accuracy after 5 epochs of no improvement.

Compression and accuracy. Table 1 shows the results for the baseline (unpruned) models and three configurations of DropBack, retaining respectively 50,000 weights, 20,000 weights, and 1,500 weights. 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 prior work such as DSD (Han et al., 2017), where a sparse layer that omits 30%–50% weights outperforms the baseline dense (all-weights) model. However, DSD first trains the network to convergence on the *complete* parameter set, and only then prunes some weights and retrains the resulting sparse network. In contrast, DropBack achieves improvement *without* any training in the dense configuration, and without repeatedly retraining the network to convergence. 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

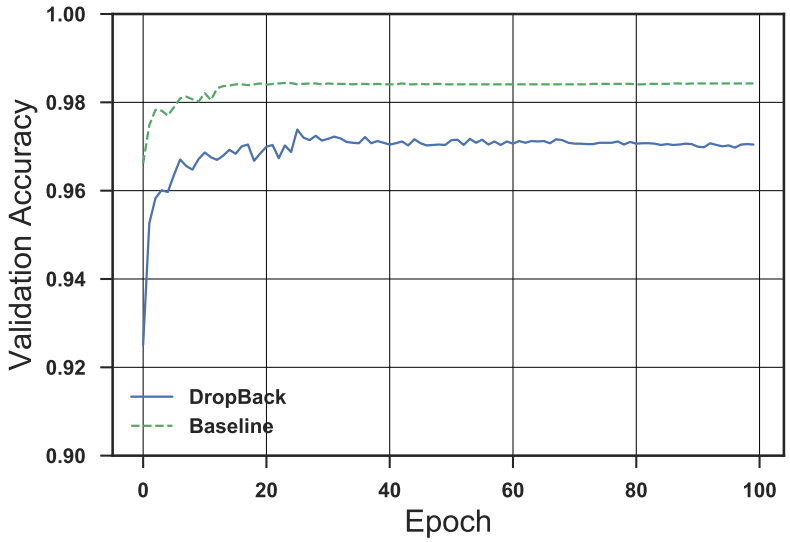

Figure 3: Rate of convergence for LeNet-300-100 for our technique our and the baseline model. Note that the y-axis starts as 0.90, and the final accuracies are within 1% of each other.

the model to 20,000 weights results in nearly the same accuracy as the baseline on both models, and convergence in a comparable number of epochs.

We also investigated an extreme compression configuration with weight storage reduced drastically to 1,500 weights. Not surprisingly, the error rate increases (over 2×). The nearly 60× reduction in the weight storage requirement for the smaller model and 177× for the larger model offers an attractive design point for low-power embedded accelerators in future mobile and edge devices.

Tracked weight set freezing. For both MNIST networks, freezing sooner to reduce the computational overhead results in lower achieved accuracy—especially for very high compression ratios—but for smaller compression ratios freezing early has little effect on the overall accuracy. Figure 3 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 behavior.

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.

CIFAR-10. 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

layer	Baseline	DropBack 10000	DropBack 1500
fc1 (100×784)	78500	7223 (10.9×)	734 (107.0×)
fc2 (100×100)	10100	2128 (4.8×)	512 (19.7×)
fc3 (100×10)	1010	549 (1.8×)	254 (4.0×)
Total	89610	10000 (9.0×)	1500 (60.0×)

Table 2: Number of gradients for each layer retained in the final trained MNIST network, with the compression ratio reported per layer.

CIFAR-10	Validation error	Weight compression	Best epoch	Freeze epoch
VGG-S 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	90%	N/A×	N/A	N/A
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	90%	N/A×	N/A	N/A
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 compression ratio of several pruned networks on CIFAR-10.

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 starting learning rate of 0.4 decayed 0.5× every 25 epochs.

Compression 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 compression, and up to 20×–30× if some accuracy is sacrificed. On WRN-28-10, DropBack achieves 5× and 7× compression 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 well only on VGG-S, and fails to converge on Densenet and WRN. Magnitude-based pruning does not achieve better accuracy than DropBack despite less weight compression. Finally, network slimming achieves slightly better top accuracy on Densenet with 50% less compression, 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 compression on the order of 5× with little to no accuracy loss.

Effects of freezing. At under 10× weight pruning, accuracy does not suffer against the uncompressed baseline, and freezing does not affect either compression or accuracy. With more compression, however, the tracked gradient set struggles to maintain accuracy, and freezing the gradients early results in substantial accuracy drops at 30× compression.

Convergence. Figure 4 shows that DropBack initially learns slightly more slowly than the uncompressed baseline, but exhibits the same convergence behavior after about 20 epochs (VGG-S). Variational dropout, in contrast, learns more quickly during the initial 20 epochs but converges on a substantially lower accuracy.

4 Discussion

To investigate why DropBack consistently achieves good compression/accuracy tradeoffs across a wide range of networks, we considered the training process analysis from Hoffer et al. (2017). Briefly,

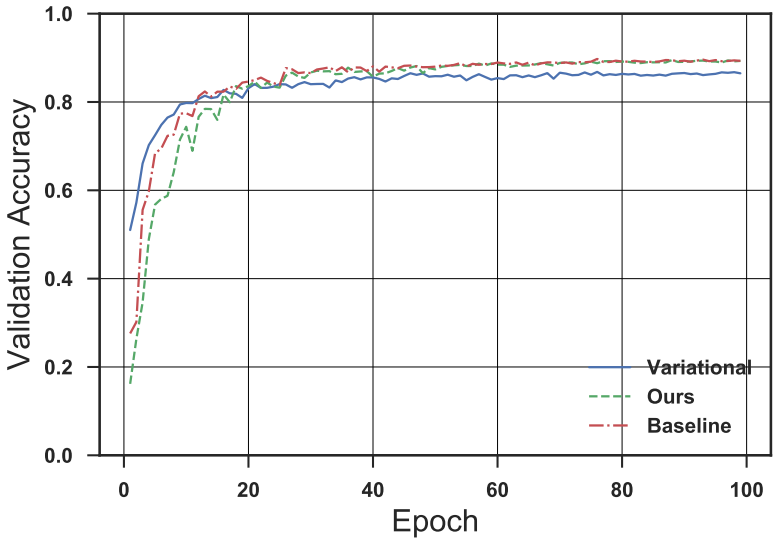

Figure 4: VGG-S CIFAR-10 epoch vs. validation accuracy for our method at 5M tracked parameters, variational dropout, and the baseline model.

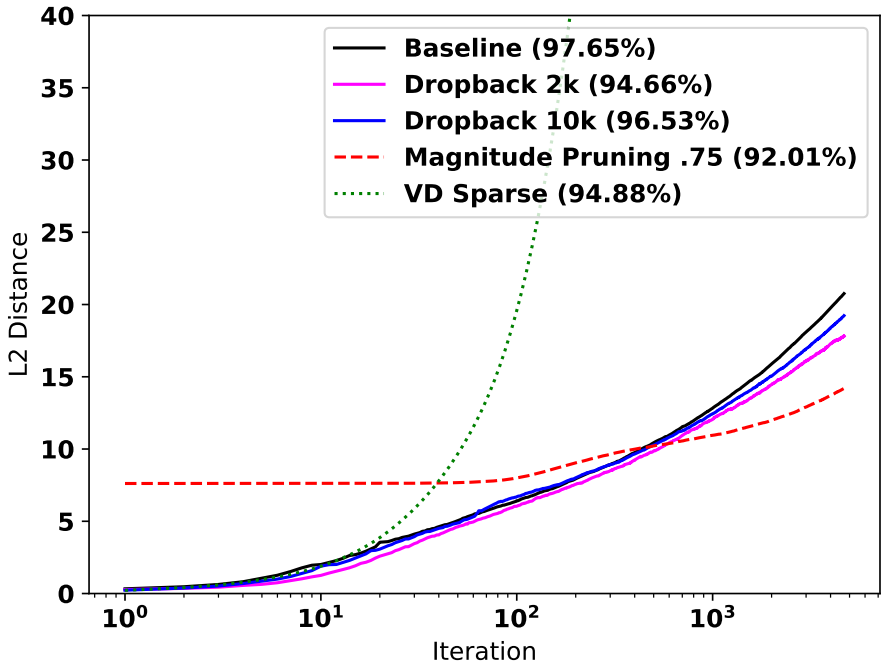

Figure 5: Diffusion (ℓ^2) distance vs. training time on MNIST-100-100 (note log time scale).

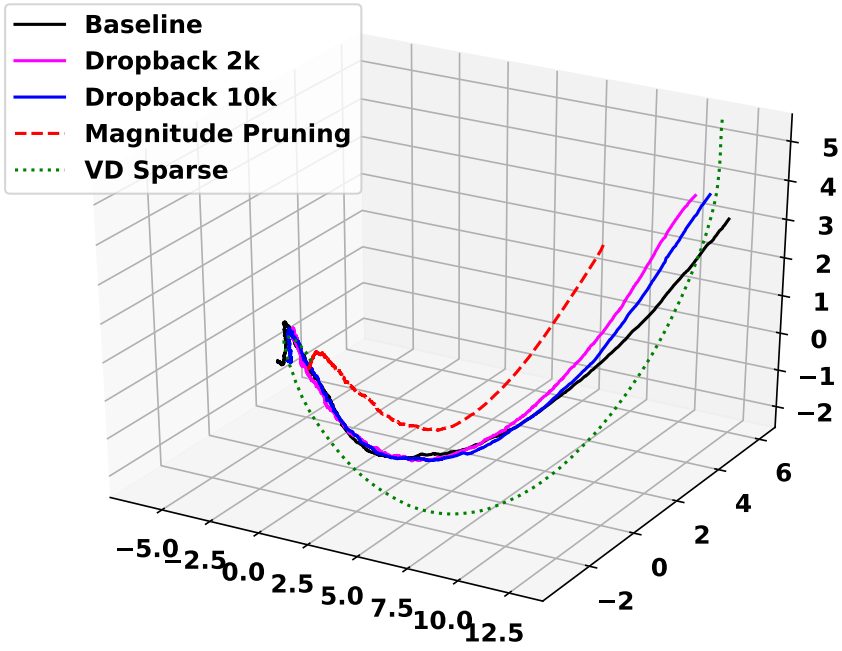

Figure 6: Evolution of weights under SGD projected into 3D space using PCA for DropBack, baseline, magnitude based pruning, and variational dropout.

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 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. Fig. 1). To verify this intuition, we measured the diffusion distance for the baseline uncompressed network, DropBack, variational dropout, and magnitude-based pruning, all on the MNIST-100-100 network.[2] Figure 5 shows that under DropBack weights diffuse very similarly to the baseline training scheme, with the overall ℓ^2 distance diverging only very slightly because the untracked weights remain at their initialization values. In contrast, magnitude-based pruning begins with a large ℓ^2 distance (because many initialization weights are zeroed), and does not provide enough scaffolding structure for SGD to train well. Finally, variational dropout drastically alters the loss surface of the network, and so diffuses much faster than baseline and DropBack which results in numerical instability as Hoffer et al. (2017) predict, corresponding to its failure to converge on the denser networks (see Section 3).

To visualize how the weight values evolve under DropBack compared to the baseline and the two pruning techniques, we projected the parameter space down to 3D using PCA. Figure 6 shows that under DropBack, the principal components of the trained weight vector stay very close to those of the baseline-trained weight vector, whereas those of magnitude-based pruning and variational dropout diverge significantly. If we imagine the training path of the baseline uncompressed configuration to be optimal, DropBack results in a near-optimal evolution.

[2]Network slimming, being a train-prune-retrain technique, is not amenable to this type of analysis.

5 Related Work

Pruning. Pruning networks to enforce sparsity after training has been effective in Han et al. (2015a); 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. (2013). 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. While accuracy is higher, the additional memory and energy cost of the is undesirable in embedded systems.

Other work has focused on pruning while training to either improve accuracy or to increase sparsity. 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. Alvarez & Salzmann (2017) focused on reducing the rank of the parameter matrices during training for later compression. Babaeizadeh et al. (2016) inject random noise into a network to finds and merge the most correlated neurons. Finally, Langford et al. (2008) 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.

Quantization. Reducing storage costs is often approached through quantizing parameters to lower bit widths, preformed either post training as in Ge et al. (2017); Wu et al. (2015); Choi et al. (2016); Han et al. (2015a); Zhou et al. (2017); Ullrich et al. (2017); Gysel (2016), 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. (2014); Simard & Graf (1994); Holt & Baker (1991). Out of all of these methods, only Gupta et al. (2015); Mishra et al. (2017); Hubara et al. (2016); Courbariaux et al. (2014) use reduced precision *while training* to lower the train time storage costs; other methods require the real value parameters to be stored during backpropagation. Quantization is orthogonal to DropBack, and the two techniques can be combined.

6 Conclusion

In this paper, we introduce DropBack, a 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. This recomputation costs 427× less than accessing a weight from off chip memory.

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 techniques. This allows DropBack to achieve better accuracy and weight compression than prior methods on dense modern networks like Densenet and WRN, which have proven challenging to prune using existing techniques: DropBack achieves a 5×–7× weight reduction on these networks with nearly no accuracy loss.

Perhaps most attractively, DropBack is the first technique to dramatically reduce the memory footprint and memory bandwidth needed to store and access weights *during* training. Because of this, DropBack can be used to train networks 5×–10× larger than currently possible with typical hardware, or to train/retrain standard-size networks on small mobile and embedded devices, something not possible with current training techniques.

References

Alvarez, Jose M. and Salzmann, Mathieu. Compression-aware Training of Deep Networks. *arXiv:1711.02638 [cs]*, November 2017. arXiv: 1711.02638.

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

Babaeizadeh, Mohammad, Smaragdis, Paris, and Campbell, Roy H. NoiseOut: A simple way to prune neural networks. 2016.

Cai, Zhaowei, He, Xiaodong, Sun, Jian, and Vasconcelos, Nuno. Deep Learning with Low Precision by Half-wave Gaussian Quantization. *arXiv:1702.00953 [cs]*, February 2017. arXiv: 1702.00953.

Choi, Yoojin, El-Khamy, Mostafa, and Lee, Jungwon. Towards the Limit of Network Quantization. *arXiv:1612.01543 [cs]*, December 2016. arXiv: 1612.01543.

Courbariaux, Matthieu, Bengio, Yoshua, and David, Jean-Pierre. Training deep neural networks with low precision multiplications. *arXiv:1412.7024 [cs]*, December 2014. arXiv: 1412.7024.

Courbariaux, Matthieu, Hubara, Itay, Soudry, Daniel, El-Yaniv, Ran, and Bengio, Yoshua. 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.

Ge, S., Luo, Z., Zhao, S., Jin, X., and Zhang, X. Y. Compressing deep neural networks for efficient visual inference. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 667–672, July 2017. doi: 10.1109/ICME.2017.8019465.

Gupta, Suyog, Agrawal, Ankur, Gopalakrishnan, Kailash, and Narayanan, Pritish. Deep learning with limited numerical precision. 2015.

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

Han, Song, Mao, Huizi, and Dally, William J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*, October 2015a. arXiv: 1510.00149.

Han, Song, Pool, Jeff, Tran, John, and Dally, William. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015b.

Han, Song, Liu, Xingyu, Mao, Huizi, Pu, Jing, Pedram, Ardavan, Horowitz, Mark A., and Dally, William J. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2016.

Han, Song, Pool, Jeff, Narang, Sharan, Mao, Huizi, Gong, Enhao, Tang, Shijian, Elsen, Erich, Vajda, Peter, Paluri, Manohar, Tran, John, Catanzaro, Bryan, and Dally, William J. DSD: Dense-Sparse-Dense Training for Deep Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Hassibi, Babak, Stork, David G, and Wolff, Gregory J. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks, 1993.,*, pp. 293–299. IEEE, 1993.

Hoffer, Elad, Hubara, Itay, and Soudry, Daniel. 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-91-Seattle International Joint Conference on Neural Networks*, volume ii, pp. 121–126 vol.2, 1991.

Huang, Gao, Liu, Zhuang, van der Maaten, Laurens, and Weinberger, Kilian Q. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, August 2016. arXiv: 1608.06993.

Hubara, Itay, Courbariaux, Matthieu, Soudry, Daniel, El-Yaniv, Ran, and Bengio, Yoshua. Quantized neural networks: Training neural networks with low precision weights and activations. 2016.

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

Kingma, Diederik P., Salimans, Tim, and Welling, Max. Variational dropout and the local reparameterization trick. 2015.

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

Krizhevsky, Alex. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pp. 1097–1105. Curran Associates, Inc., 1990.

Langford, John, Li, Lihong, and Zhang, Tong. Sparse online learning via truncated gradient. 2008.

LeCun, Y. The MNIST database of handwritten digits. 1998.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. 86(11):2278–2324, 1998. ISSN 0018-9219.

LeCun, Yann, Denker, John S., and Solla, Sara A. Optimal brain damage. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2*, pp. 598–605. Morgan-Kaufmann, 1990.

LeCun, Yann, Bottou, Léon, Orr, Genevieve B., and Müller, Klaus-Robert. Effiicient BackProp. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pp. 9–50, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65311-2.

Li, Hao, De, Soham, Xu, Zheng, Studer, Christoph, Samet, Hanan, and Goldstein, Tom. Training Quantized Nets: A Deeper Understanding. *arXiv:1706.02379 [cs, stat]*, June 2017. arXiv: 1706.02379.

Liu, Zhuang, Li, Jianguo, Shen, Zhiqiang, Huang, Gao, Yan, Shoumeng, and Zhang, Changshui. Learning Efficient Convolutional Networks through Network Slimming. *arXiv:1708.06519 [cs]*, August 2017. arXiv: 1708.06519.

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

Luo, Jian-Hao, Wu, Jianxin, and Lin, Weiyao. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *arXiv:1707.06342 [cs]*, July 2017. arXiv: 1707.06342.

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

Masana, Marc, van de Weijer, Joost, Herranz, Luis, Bagdanov, Andrew D., and Alvarez, Jose M. Domain-adaptive deep network compression. *arXiv:1709.01041 [cs]*, September 2017. arXiv: 1709.01041.

Mishra, Asit, Nurvitadhi, Eriko, Cook, Jeffrey J., and Marr, Debbie. WRPN: Wide reduced-precision networks. 2017.

Molchanov, Dmitry, Ashukha, Arsenii, and Vetrov, Dmitry. Variational dropout sparsifies deep neural networks. 2017.

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

Rastegari, Mohammad, Ordonez, Vicente, Redmon, Joseph, and Farhadi, Ali. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv:1603.05279 [cs]*, March 2016. arXiv: 1603.05279.

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

Simard, Patrice Y and Graf, Hans Peter. Backpropagation without multiplication. In *Advances in Neural Information Processing Systems*, pp. 232–239, 1994.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Tokui, Seiya, Oono, Kenta, Hido, Shohei, and Clayton, Justin. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in (NIPS)*, volume 5, 2015.

Ullrich, Karen, Meeds, Edward, and Welling, Max. Soft Weight-Sharing for Neural Network Compression. *arXiv:1702.04008 [cs, stat]*, February 2017. arXiv: 1702.04008.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, Cun, Yann Le, and Fergus, Rob. Regularization of neural networks using DropConnect. In *PMLR*, pp. 1058–1066, 2013.

Wu, Jiaxiang, Leng, Cong, Wang, Yuhang, Hu, Qinghao, and Cheng, Jian. Quantized Convolutional Neural Networks for Mobile Devices. *arXiv:1512.06473 [cs]*, December 2015. arXiv: 1512.06473.

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

Zhou, Aojun, Yao, Anbang, Guo, Yiwen, Xu, Lin, and Chen, Yurong. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *arXiv:1702.03044 [cs]*, February 2017. arXiv: 1702.03044.

Zhou, Shuchang, Wu, Yuxin, Ni, Zekun, Zhou, Xinyu, Wen, He, and Zou, Yuheng. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv:1606.06160 [cs]*, June 2016. arXiv: 1606.06160.

Zhu, Jingyang, Jiang, Jingbo, Chen, Xizi, and Tsui, Chi-Ying. SparseNN: An Energy-Efficient Neural Network Accelerator Exploiting Input and Output Sparsity. *arXiv:1711.01263 [cs]*, November 2017. arXiv: 1711.01263.

Zhu, Michael and Gupta, Suyog. To prune, or not to prune: exploring the efficacy of pruning for model compression. 2017.