Low-Memory NN Training

Sparsity

Presenter: Kim Tae Hyeon

KAIST

Contents

1. **Introduction**
2. Recent Work
3. Method
4. Discussion

Introduction : Issue

How much memory is actually needed to train a neural network?

Model Memory	: consists of the model parameters
Optimizer Memory	: consists of the gradients and momentum vectors
Activation Memory	: consists of the intermediate network activations

Introduction : Issue

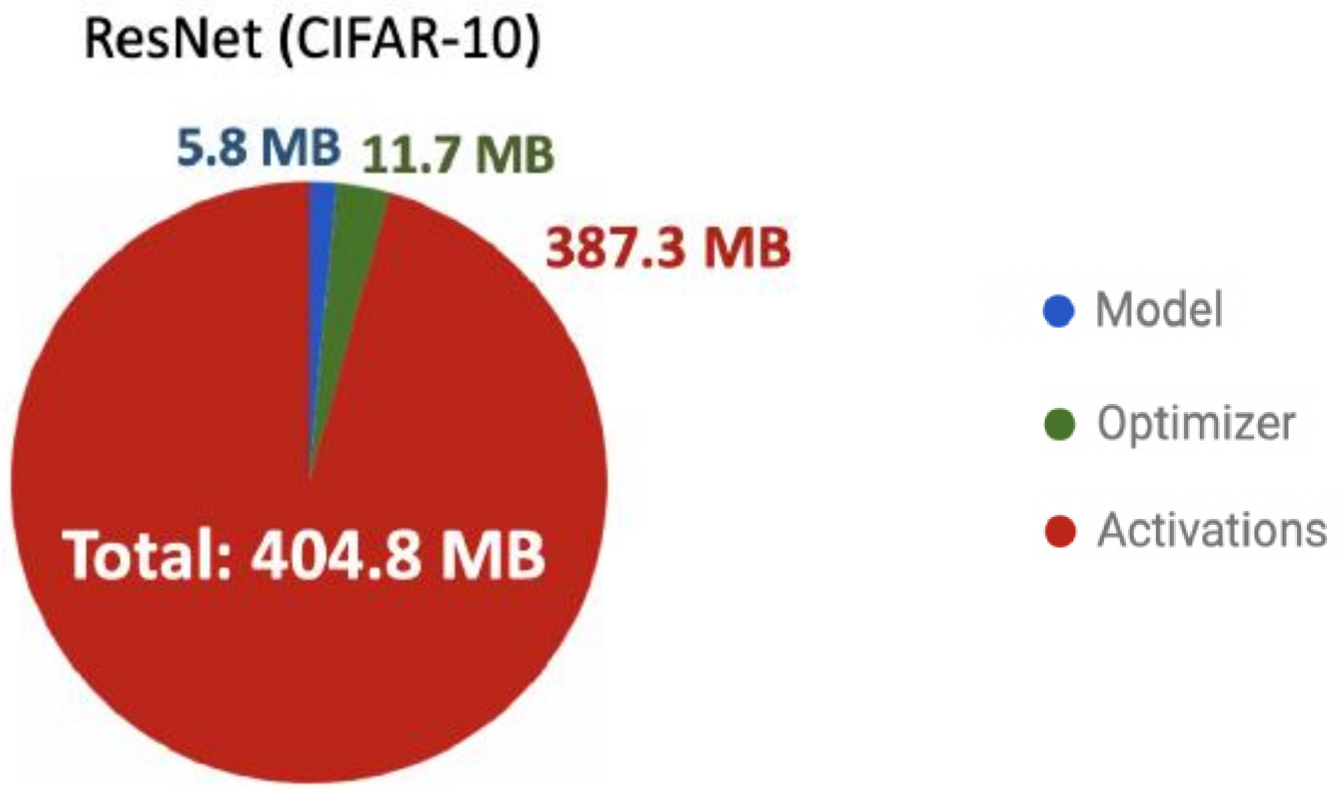


Figure 1: Pie charts of training memory consumption for WideResNet on CIFAR-10 [1]

Introduction : Issue

Model Memory

It is simply **the memory used to store the model parameters,**
i.e. the weights and biases of each layer in the network.

Only major memory type that is common across both **inference** and **training** for the model architecture.

Introduction : Issue

Optimizer Memory

It refers to **the memory used to store the gradients and any momentum buffers during training.**

During backpropagation, compute the gradient with respect to all trainable model parameters.

Introduction : Issue

Activation Memory

During training, the outputs, or activations, of each layer in the network are stored for reuses in the backward pass of backpropagation

(They also count the network inputs as part of the forward activation memory if they need to be stored for reuse in the backward pass.)

Introduction : Technique

How to control these memories?

Of course, there are trade-offs between accuracies and memory reduction

Four memory reduction techniques

1. Sparsity
2. Low precision
3. Microbatching
4. Gradient checkpointing

Introduction : Technique

1. Sparsity

What is the definition of sparse?

➔ A sparse tensor is a tensor in which most of the entries are zero.
➔ The zero entries do not need to be explicitly stored, so it can reduce the memory usage.

Techniques

- Traditional : apply sparsity to compress pre-trained model with little or no loss of accuracy.
- Recent : maintain sparsity throughout the entire training process.

Introduction : Technique

2. Low Precision

Precision

➔ Generally, model is trained in single precision (FP32) (IEEE 32-bit] arithmetic.

➔ It enables the model both to reduce all components of memory and to decrease the total amount of required computations.

Techniques

- Recent : Half precision (FP16) [IEEE 16-bit] arithmetic is generally sufficient for training

Introduction : Technique

3. Microbatching

Issue

➔ The memory required for computing and storing the network activation can far outstrip the memory required to store the model itself.

➔ Two possible solutions are to downsample the input data to a smaller size, or to make the model "narrower" {(ex) reducing the number of output channels of each convolution)

What is microbatching?

➔ The minibatch is split into smaller groups called microbatching

➔ The gradients are accumulated in a separate buffer until the desired number of examples is processed, and only then are the parameters and momentum buffers updated.

Introduction : Technique

4. Gradient checkpointing

What is Gradient checkpointing?

➔ It is an algorithm to save memory on the network activations by only storing the activations of a subset of layers, rather than those of each layer as usual.

➔ The activations that are discarded are recomputed when necessary during backpropagation.

➔ Checkpointing is particularly convenient because the outcome of the forward and backward passes is mathematically and numerically equivalent whether or not checkpointing is used, and so there is no memory-accuracy tradeoff.

Contents

1. Introduction
2. **Recent Work**
3. Method
4. Discussion

Recent Work : Sparsity

Paper
2019 ICML

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

Recent Work : Sparsity

Issue

- Training remains memory-inefficient despite the compact size of the resultant network produced by compression.
- Training-time parameter efficiency.

Hypothesis

Overparameterization during training might not be a strict necessity and alternative training or reparameterization methods might exist to discover and train compact networks directly.

Recent Work : Sparsity

Previous works

1. Architecture
   - Skip Connection, Global average pooling, Depthwise separable convolution.

2. Reparameterization
   - Any differentiable reparameterization can be used to augment training of a given model.
   - if Origianl Network **:** $y = f(x;\theta), parameterized\ by\ \theta \in \Theta$, then
   - Reparameterization : $\phi \in \Phi\ and\ \psi \in \Psi\ through\ \theta = g(\phi, \psi)$, where g is differentiable w.r.t. ϕ but not necessarily w.r.t. ψ
   - Denote the reparameterized network by f_ψ, considering ψ as metaparameters:

$$y = f(x; g(\phi, \psi)) \triangleq f_\psi(x;\phi)$$

Recent Work : Sparsity

Sparse reparameterization

$$y = f(x; g(\phi, \psi)) \triangleq f_{\psi}(x; \phi)$$

It is a special case where g is a linear projection; ϕ is non-zero entries (i.e. weights) and ψ their indices (i.e. connectivity) in the original parameter tensor θ.

Parameter Sharing

$$y = f(x; g(\phi, \psi)) \triangleq f_{\psi}(x; \phi)$$

It is a similar special case of linear reparameterization where ϕ is the tied parameters and ψ the indices at which each parameter is placed (with repetition) in the original parameter tensor θ.

Recent Work : Sparsity

Dense reparameterization

- low-rank decomposition
- Fastfood transform
- ACDC transform
- Hashed-Net
- Low displacement rank
- Block-circulant matrix parameterization

Sparse reparameterization

- Iteratively pruning and retraining
- Pretrained & pruning and fine-tuning
- Lottery Ticket Hypothesis (2019 ICLR Oral)

Recent Work : Sparsity

Definition

1. Static
metaparameters ψ are fixed during the course of training, the reparameterization is static.

2. Dynamic
if ψ is adjusted adaptively during training, we call it dynamic reparameterization.

Method

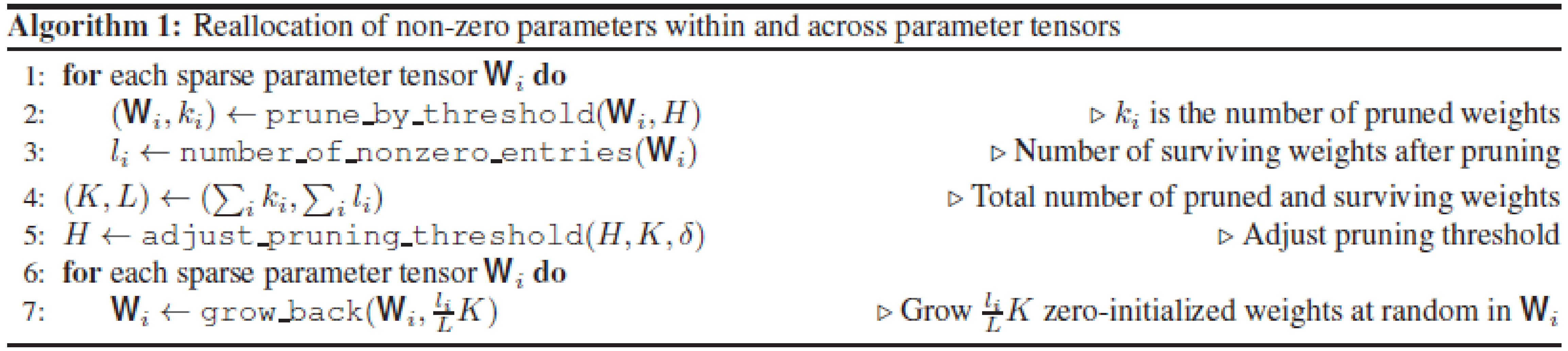
A Novel dynamic reparameterization method that yielded the highest parameter efficiency in training sparse deep residual networks, outperforming existing static and dynamic reparameterization methods.

Contents

1. Introduction
2. Recent Work
3. **Method**
4. Discussion

Method

Algorithm



Algorithm 1: Reallocation of non-zero parameters within and across parameter tensors

```
1: for each sparse parameter tensor $\mathbf{W}_i$ do
2:     $(\mathbf{W}_i, k_i) \leftarrow$ prune_by_threshold($\mathbf{W}_i, H$)        ▷ $k_i$ is the number of pruned weights
3:     $l_i \leftarrow$ number_of_nonzero_entries($\mathbf{W}_i$)        ▷ Number of surviving weights after pruning
4: $(K, L) \leftarrow (\sum_i k_i, \sum_i l_i)$        ▷ Total number of pruned and surviving weights
5: $H \leftarrow$ adjust_pruning_threshold($H, K, \delta$)        ▷ Adjust pruning threshold
6: for each sparse parameter tensor $\mathbf{W}_i$ do
7:     $\mathbf{W}_i \leftarrow$ grow_back($\mathbf{W}_i, \frac{l_i}{L}K$)        ▷ Grow $\frac{l_i}{L}K$ zero-initialized weights at random in $\mathbf{W}_i$
```

Case.

If a layer has been heavily pruned, this indicates that, for a large portion of its parameters, the training loss gradients are not large or consistent enough to counter act the pull towards zero exerted by weight-decay regularization. This layer is therefore to receive a smaller share of new free parameters during the growth phase.

Method

Results

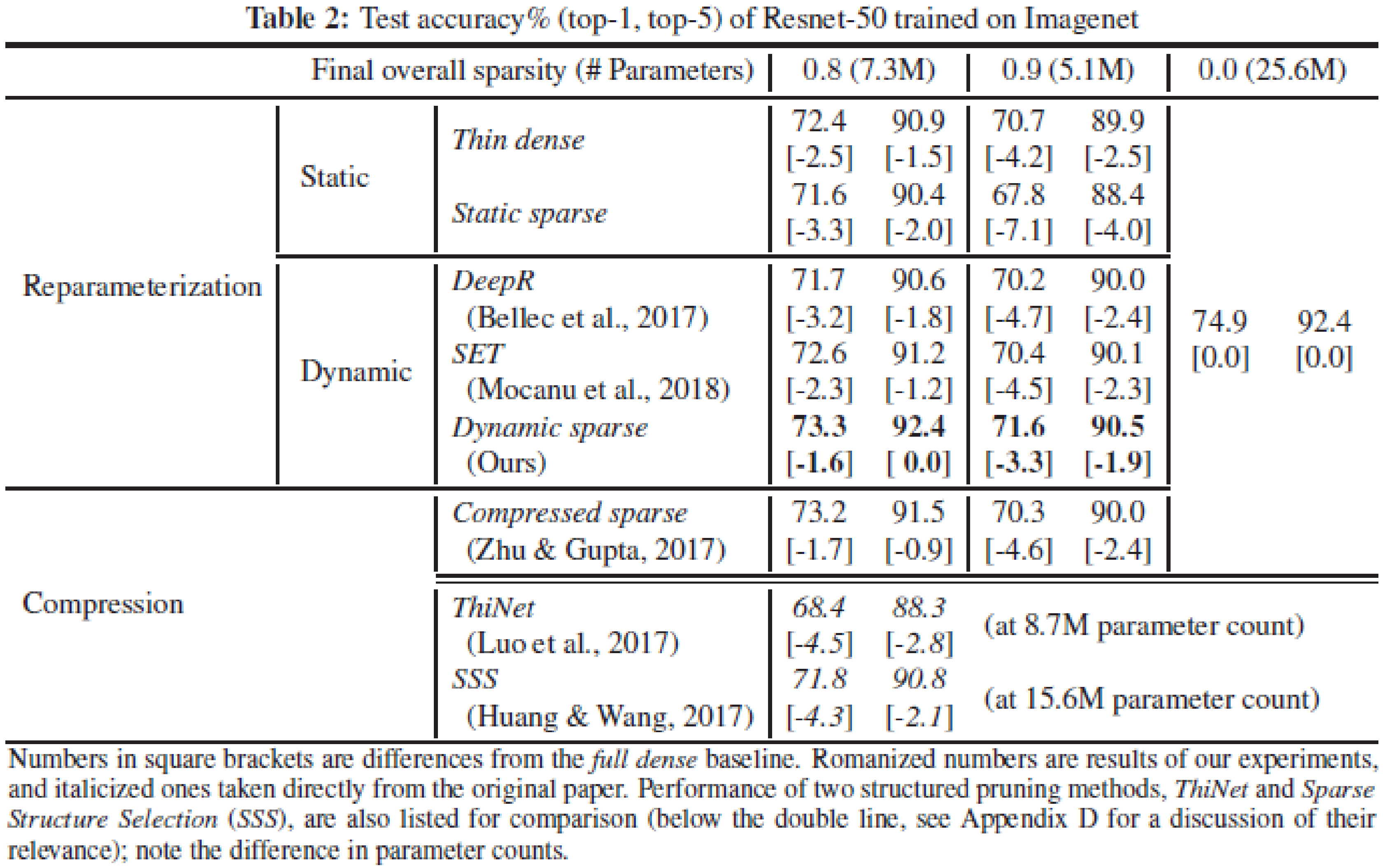


Table 2: Test accuracy% (top-1, top-5) of Resnet-50 trained on Imagenet

		Final overall sparsity (# Parameters)	0.8 (7.3M)		0.9 (5.1M)		0.0 (25.6M)	
Reparameterization	Static	*Thin dense*	72.4 [-2.5]	90.9 [-1.5]	70.7 [-4.2]	89.9 [-2.5]		
		Static sparse	71.6 [-3.3]	90.4 [-2.0]	67.8 [-7.1]	88.4 [-4.0]		
	Dynamic	*DeepR* (Bellec et al., 2017)	71.7 [-3.2]	90.6 [-1.8]	70.2 [-4.7]	90.0 [-2.4]	74.9 [0.0]	92.4 [0.0]
		SET (Mocanu et al., 2018)	72.6 [-2.3]	91.2 [-1.2]	70.4 [-4.5]	90.1 [-2.3]		
		Dynamic sparse (Ours)	**73.3 [-1.6]**	**92.4 [0.0]**	**71.6 [-3.3]**	**90.5 [-1.9]**		
Compression		*Compressed sparse* (Zhu & Gupta, 2017)	73.2 [-1.7]	91.5 [-0.9]	70.3 [-4.6]	90.0 [-2.4]		
		ThiNet (Luo et al., 2017)	*68.4 [-4.5]*	*88.3 [-2.8]*	(at 8.7M parameter count)			
		SSS (Huang & Wang, 2017)	*71.8 [-4.3]*	*90.8 [-2.1]*	(at 15.6M parameter count)			

Numbers in square brackets are differences from the *full dense* baseline. Romanized numbers are results of our experiments, and italicized ones taken directly from the original paper. Performance of two structured pruning methods, *ThiNet* and *Sparse Structure Selection* (*SSS*), are also listed for comparison (below the double line, see Appendix D for a discussion of their relevance); note the difference in parameter counts.

Contents

1. Introduction
2. Recent Work
3. Method
4. **Discussion**

Discussion

1. Why this mechanism yield high performance?

They found that neither the post hoc sparseness pattern, nor the combination of connectivity and initialization, managed to explain the high performance of sparse networks produced by their dynamic sparse training method. Thus, the value of dynamic parameter reallocation goes beyond discovering trainable sparse network structure: the evolutionary process of structural exploration itself seems helpful for SGD to converge to better weights.

Analysis (previous)

One hypothesis is that the discontinuous jumps in parameter space when parameters are reallocated across layers helped training escape sharp minima that generalize badly.

Discussion

2. Structural degrees of freedom

Structural degrees of freedom are qualitatively different from the degrees of freedom introduced by overparameterization. The latter directly expands the dimensionality of the parameter space in which SGD directly optimizes, whereas structural degrees of freedom are realized and explored using non-differentiable heuristics that only interact indirectly with the dynamics of gradient-based optimization, e.g. regularization pulling weights towards zero causing connections to be pruned. Their results suggest that, for residual CNNs under a given descriptive complexity (i.e. memory storage) constraint, it is better (in the sense of producing models that generalize better) to allocate some memory to describe and explore structural degrees of freedom, than to allocate all memory to conventional weights.

Discussion

Contribution

- it is possible to train deep sparse CNNs directly to reach generalization performances comparable to those achieved by iterative pruning and fine-tuning of pre-trained large dense models.
- dynamic exploration of structural degrees of freedom during training is crucial to effective learning.

Thank you

Reference

[1] Sohoni, N. S., Aberger, C. R., Leszczynski, M., Zhang, J., & Ré, C. (2019). Low-Memory Neural Network Training: A Technical Report. arXiv preprint arXiv:1904.10631.