Compression of random neural networks: A signal propagation perspective

Namhoon Lee

University of Oxford

November 2019

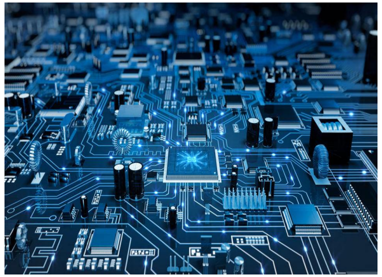
A challenge in deep learning: Overparameterization

Large neural networks require:

Critical to resource constrained environments

memory & computations

power consumption

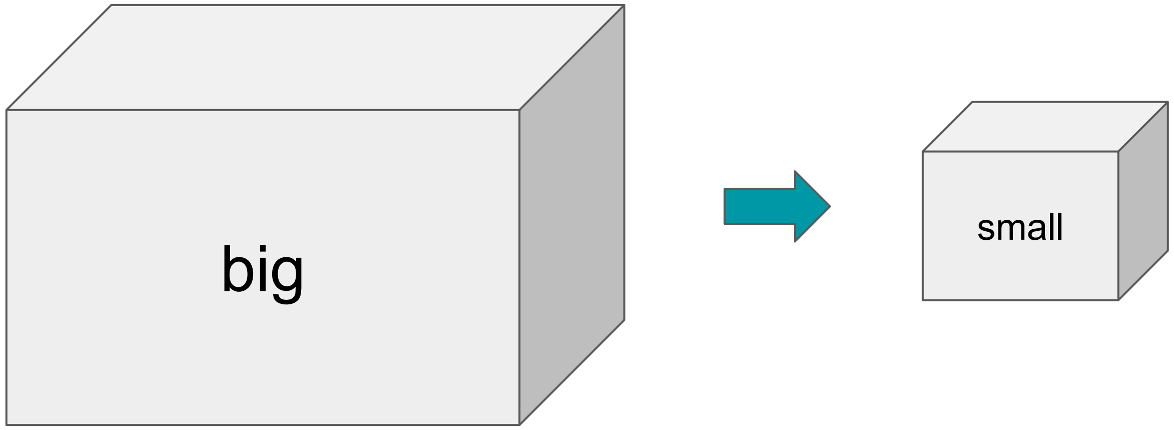


embedded systems
e.g., mobile devices

real-time tasks
e.g., autonomous car

Network compression

The goal is to reduce the ***size*** of neural network without compromising accuracy.



~ same accuracy

Approaches

- Network pruning
  : reduce the number of parameters
- Network quantization
  : reduce the precision of parameters

Others: knowledge distillation, conditional computation, etc.

Approaches

- Network pruning
  : reduce the number of parameters
- Network quantization
  : reduce the precision of parameters

Others: knowledge distillation, conditional computation, etc.

Network pruning

Different forms

- Parameters (weights, biases)
- Activations (neurons)

can be done structured way
(e.g., channel, filter, layer)

Different principles

- Magnitude based
- Hessian based
- Bayesian

Network pruning

Different forms

- Parameters (weights, biases)
- Activations (neurons)

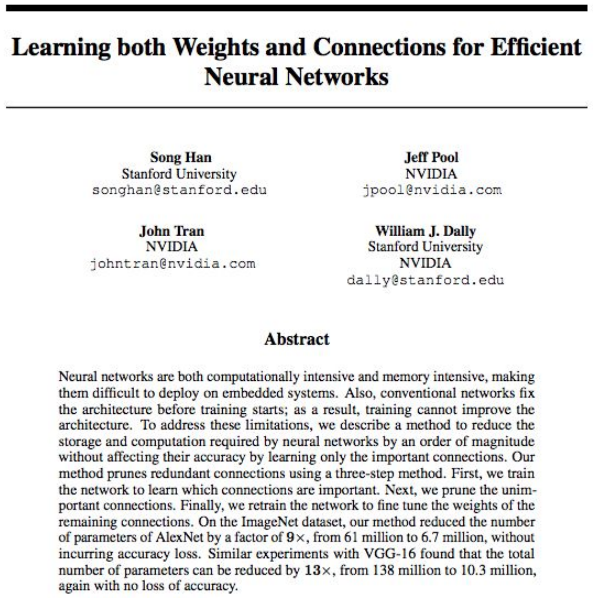
can be done structured way
(e.g., channel, filter, layer)

Different principles

- Magnitude based
- Hessian based
- Bayesian

$\boxed{\Rightarrow \text{ remove} > 90\% \text{ parameters}}$

Example: Han et al. (2015)



Learning both Weights and Connections for Efficient Neural Networks

Song Han
Stanford University
songhan@stanford.edu

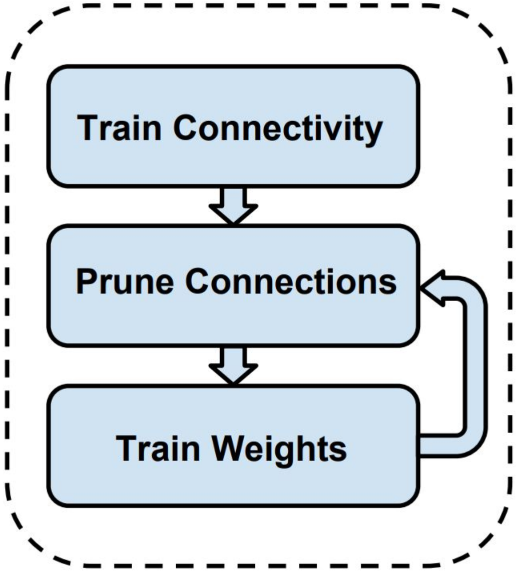
Jeff Pool
NVIDIA
jpool@nvidia.com

John Tran
NVIDIA
johntran@nvidia.com

William J. Dally
Stanford University
NVIDIA
dally@stanford.edu

Abstract

Neural networks are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems. Also, conventional networks fix the architecture before training starts; as a result, training cannot improve the architecture. To address these limitations, we describe a method to reduce the storage and computation required by neural networks by an order of magnitude without affecting their accuracy by learning only the important connections. Our method prunes redundant connections using a three-step method. First, we train the network to learn which connections are important. Next, we prune the unimportant connections. Finally, we retrain the network to fine tune the weights of the remaining connections. On the ImageNet dataset, our method reduced the number of parameters of AlexNet by a factor of **9**×, from 61 million to 6.7 million, without incurring accuracy loss. Similar experiments with VGG-16 found that the total number of parameters can be reduced by **13**×, from 138 million to 10.3 million, again with no loss of accuracy.



Drawbacks in existing approaches

- Hyperparameters with weakly-grounded heuristics (*e.g.,* stochastic pruning [2])
- Architecture specific requirements (*e.g.,* conv/fc separate prune [1])
- Optimization difficulty (*e.g.,* convergence [3, 6])
- Pretraining ([1,2,3,4,5,6])
- Iterative prune -- retrain cycle

References
[1] Learning both weights and connections for efficient neural network, Han et al. NIPS'15
[2] Dynamic network surgery for efficient dnns, Guo et al. NIPS'16.
[3] Learning-compression algorithms for neural net pruning, Carreira-Perpinan & Idelbayev. CVPR'18.
[4] Variational dropout sparsifies deep neural networks, Molchanov et al. ICML'17.
[5] Learning to prune deep neural networks via layer-wise optimal brain surgeon, Dong et al. NIPS'17.
[6] Learning Sparse Neural Networks through L0 Regularization, Louizos et al. ICLR'18

Drawbacks in existing approaches

- Hyperparameters with weakly-grounded heuristics (*e.g.,* stochastic pruning [2])
- Architecture specific requirements (*e.g.,* conv/fc separate prune [1])
- Optimization difficulty (*e.g.,* convergence [3, 6])
- Pretraining ([1,2,3,4,5,6])
- Iterative prune -- retrain cycle

⇒ *Complex and non-scalable*

References
[1] Learning both weights and connections for efficient neural network, Han et al. NIPS'15
[2] Dynamic network surgery for efficient dnns, Guo et al. NIPS'16.
[3] Learning-compression algorithms for neural net pruning, Carreira-Perpinan & Idelbayev. CVPR'18.
[4] Variational dropout sparsifies deep neural networks, Molchanov et al. ICML'17.
[5] Learning to prune deep neural networks via layer-wise optimal brain surgeon, Dong et al. NIPS'17.
[6] Learning Sparse Neural Networks through L0 Regularization, Louizos et al. ICLR'18

We want ..

No hyperparameters

No iterative prune -- retrain cycle

No pretraining

No large data

We want ..

No hyperparameters

No iterative prune -- retrain cycle

No pretraining

No large data

Single-shot pruning prior to training

UNIVERSITY OF OXFORD

SNIP: Single-shot Network Pruning based on Connection Sensitivity

Namhoon Lee, Thalaiyasingam Ajanthan, Philip Torr

International Conference on Learning Representations (*ICLR*) *2019*

Objective

- Identify important parameters in the network and remove unimportant ones

$$\min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \, ,$$
$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \underline{\|\mathbf{w}\|_0 \leq \kappa \, .}$$

- Directly optimizing with a sparsity enforcing regularization term is possible, but can be difficult.

Idea

- Measure the effect of removing each parameter on the loss

$$\Delta L_j(\mathbf{w}; \mathcal{D}) = L(\mathbf{1} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{1} - \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D})$$

- The greedy way is prohibitively expensive to perform: $O(m!)$

SNIP

The effect on the loss can be approximated by

1. auxiliary variables representing the connectivity of parameters
2. derivative of the loss w.r.t. these indicator variables

SNIP

1. Introduce c

$$\begin{aligned}\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c}\odot\mathbf{w};\mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{c}\odot\mathbf{w};(\mathbf{x}_i,\mathbf{y}_i))\ ,\\ \text{s.t.}\quad \mathbf{w}\in\mathbb{R}^m\ ,\\ \mathbf{c}\in\{0,1\}^m,\quad \|\mathbf{c}\|_0\le\kappa\ ,\end{aligned}$$

SNIP

1. Introduce c

$$\begin{aligned}\min_{\mathbf{c},\mathbf{w}} L(\underline{\mathbf{c}\odot\mathbf{w}};\mathcal{D}) &= \min_{\mathbf{c},\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{c}\odot\mathbf{w};(\mathbf{x}_i,\mathbf{y}_i))\ ,\\ \text{s.t.}\quad & \mathbf{w}\in\mathbb{R}^m\ ,\\ & \underline{\mathbf{c}\in\{0,1\}^m,}\quad \|\mathbf{c}\|_0\le\kappa\ ,\end{aligned}$$

SNIP

1. Introduce c

$$\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{c} \odot \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \,,$$
$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m \,,$$
$$\mathbf{c} \in \{0,1\}^m, \quad \|\mathbf{c}\|_0 \le \kappa \,,$$

2. Derivative w.r.t. c

$$\Delta L_j(\mathbf{w}; \mathcal{D}) \approx g_j(\mathbf{w}; \mathcal{D}) = \left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial c_j}\right|_{\mathbf{c}=\mathbf{1}} = \lim_{\delta \to 0} \left.\frac{L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{c} - \delta\, \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D})}{\delta}\right|_{\mathbf{c}=\mathbf{1}}$$

SNIP

1. Introduce c

$$\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{c} \odot \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) ,$$

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m ,$$

$$\mathbf{c} \in \{0,1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa ,$$

2. Derivative w.r.t. c

$$\Delta L_j(\mathbf{w}; \mathcal{D}) \approx g_j(\mathbf{w}; \mathcal{D}) = \left. \frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial c_j} \right|_{\mathbf{c}=\mathbf{1}} = \lim_{\delta \to 0} \left. \frac{L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) - L(\boxed{(\mathbf{c} - \delta\, \mathbf{e}_j)} \odot \mathbf{w}; \mathcal{D})}{\delta} \right|_{\mathbf{c}=\mathbf{1}}$$

- ∂L/∂cj is an infinitesimal version of ∆Lj
- measures the rate of change of L w.r.t. infinitesimal change in cj from 1 → 1 − δ
- computed efficiently in one forward-backward pass using auto differentiation, for all j at once

Reference: Understanding black-box predictions via influence functions, Koh & Liang. ICML'17

Difference between ∂L/∂c and ∂L/∂w

(∂L/∂c)

$\lim_{\delta \to 0} (L(c \odot w) - L(c \odot w - \delta e_j \odot w)) / \delta \ \big|_{c=1}$

$\rightarrow$ perturbing the parameter w_j by the scaled amount δw_j

(∂L/∂w)

$\lim_{\delta \to 0} (L(c \odot w) - L(c \odot w - c \odot \delta e_j)) / \delta \ \big|_{c=1}$

$\rightarrow$ perturbing the parameter w_j by the absolute amount δ

(example)

$m=3, j=2 \rightarrow [w_1, w_2 - \delta w_2, w_3]$ vs. $[w_1, w_2 - \delta, w_3]$

⇒ This can alleviate the dependency on the weights when computing CS.

SNIP

1. Introduce c

$$\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{c} \odot \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \,,$$
$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m \,,$$
$$\mathbf{c} \in \{0,1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa \,,$$

2. Derivative w.r.t. c

$$\Delta L_j(\mathbf{w}; \mathcal{D}) \approx g_j(\mathbf{w}; \mathcal{D}) = \left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial c_j}\right|_{\mathbf{c}=\mathbf{1}} = \lim_{\delta \to 0} \left.\frac{L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{c} - \delta\, \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D})}{\delta}\right|_{\mathbf{c}=\mathbf{1}}$$

3. Connection sensitivity

$$\boxed{s_j = \frac{|g_j(\mathbf{w}; \mathcal{D})|}{\sum_{k=1}^{m} |g_k(\mathbf{w}; \mathcal{D})|} \,.}$$

Algorithm 1 SNIP: Single-shot Network Pruning based on Connection Sensitivity

```
Require: Loss function $L$, training dataset $\mathcal{D}$, sparsity level $\kappa$        ▷ Refer Equation 3
Ensure: $\|\mathbf{w}^*\|_0 \leq \kappa$
 1: $\mathbf{w} \leftarrow$ VarianceScalingInitialization        ▷ Refer Section 4.2
 2: $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{b} \sim \mathcal{D}$        ▷ Sample a mini-batch of training data
 3: $s_j \leftarrow \frac{|g_j(\mathbf{w};\mathcal{D}^b)|}{\sum_{k=1}^{m} |g_k(\mathbf{w};\mathcal{D}^b)|}, \quad \forall j \in \{1 \dots m\}$        ▷ Connection sensitivity
 4: $\tilde{\mathbf{s}} \leftarrow \text{SortDescending}(\mathbf{s})$
 5: $c_j \leftarrow \mathbb{1}[s_j - \tilde{s}_\kappa \geq 0], \quad \forall j \in \{1 \dots m\}$        ▷ Pruning: choose top-$\kappa$ connections
 6: $\mathbf{w}^* \leftarrow \arg\min_{\mathbf{w} \in \mathbb{R}^m} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})$        ▷ Regular training
 7: $\mathbf{w}^* \leftarrow \mathbf{c} \odot \mathbf{w}^*$
```

Pruning at initialization

- Measure CS on untrained networks prior to training

  → Or zero gradients at pretrained

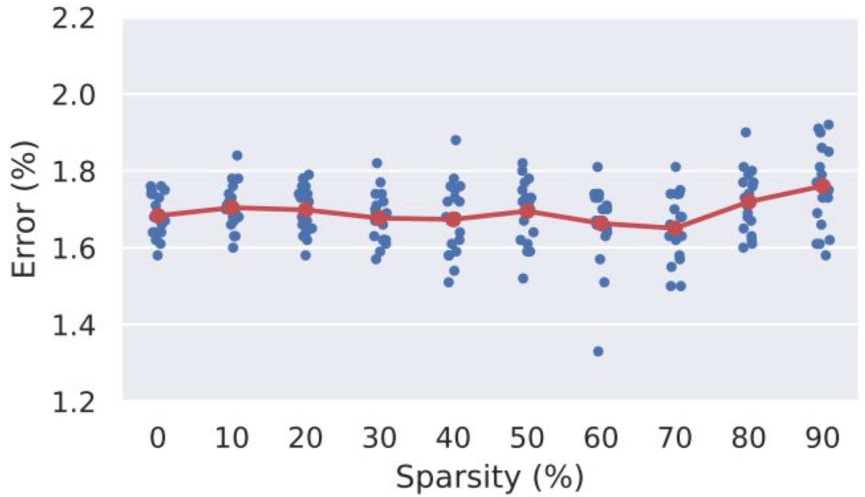
- Sample weights from a dist. with architecture aware variance

  → Ensure the variance of weights to remain throughout the network (Glorot and Bengio, 2010)

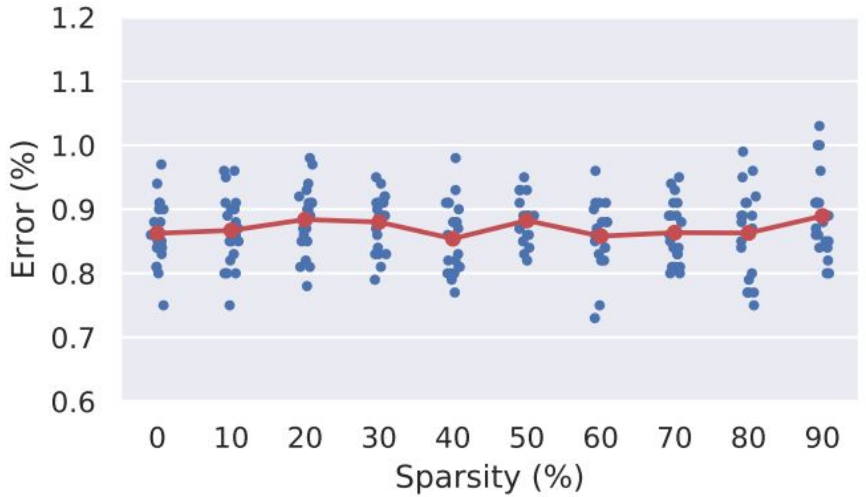
- Alleviate the dependency on the weights when computing CS

  → Remove the pretraining requirement, architecture dependent hyperparameters

LeNets

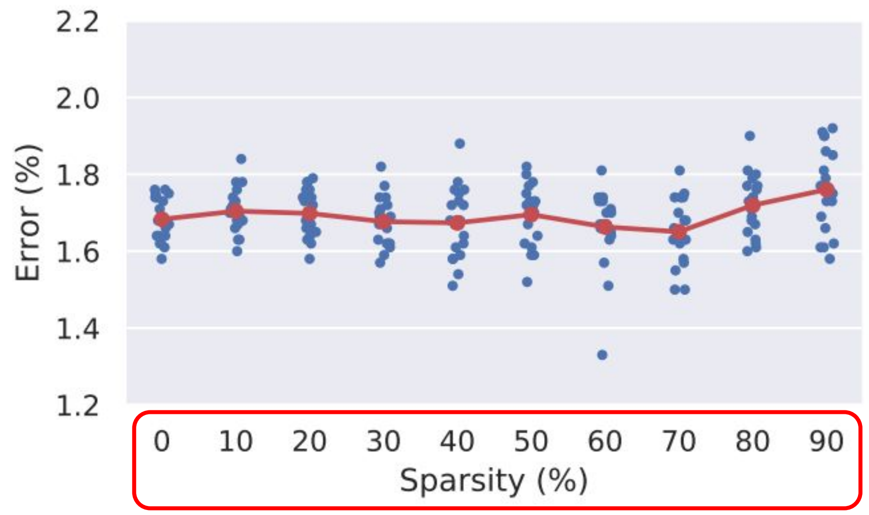


(a) LeNet-300-100

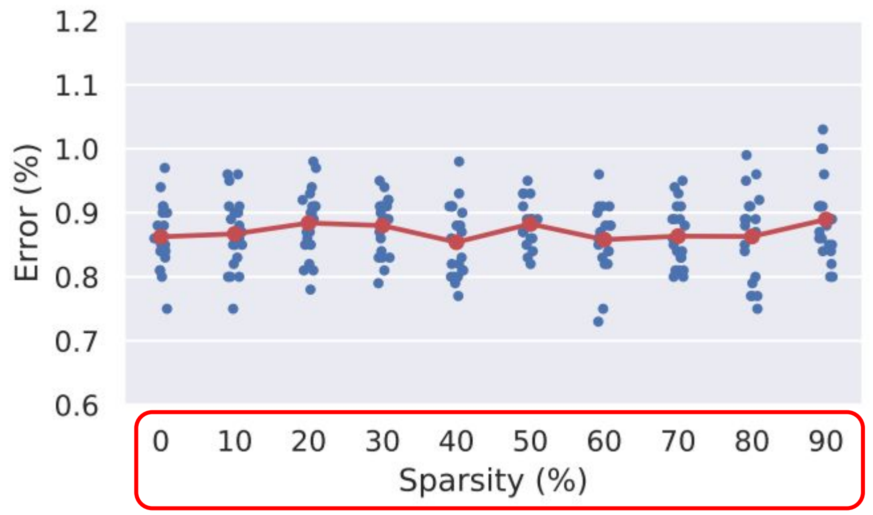


(b) LeNet-5-Caffe

LeNets

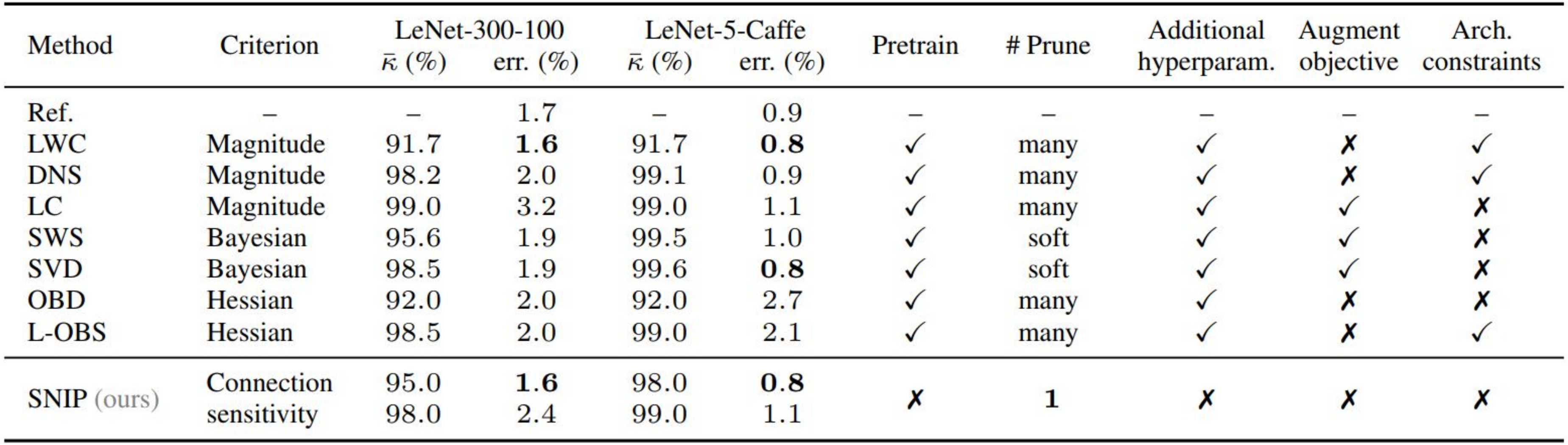


(a) LeNet-300-100



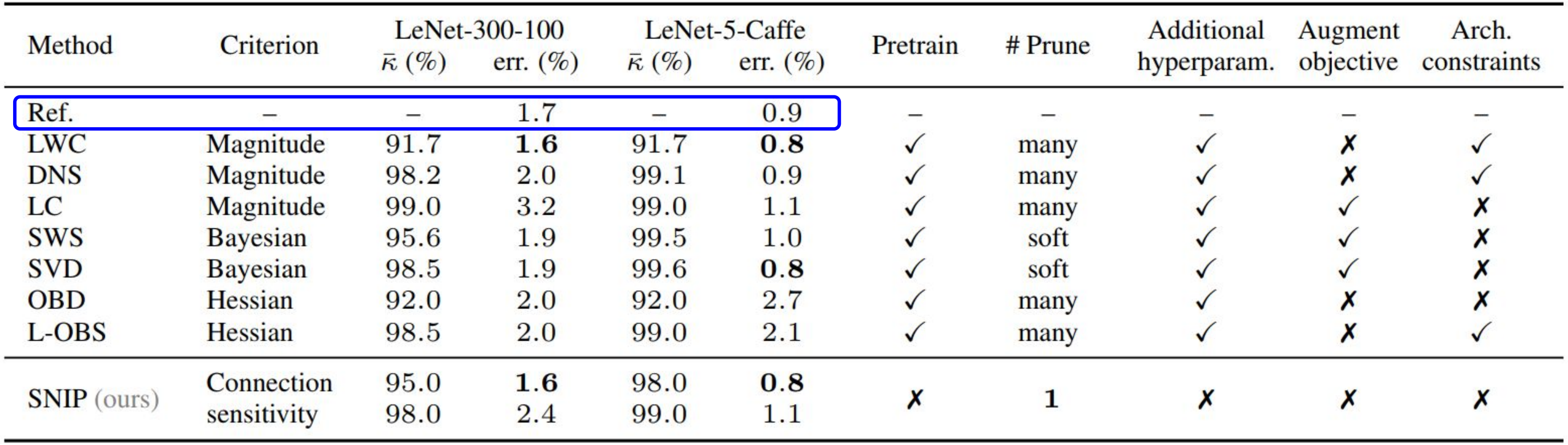
(b) LeNet-5-Caffe

LeNets: comparison to SOTA



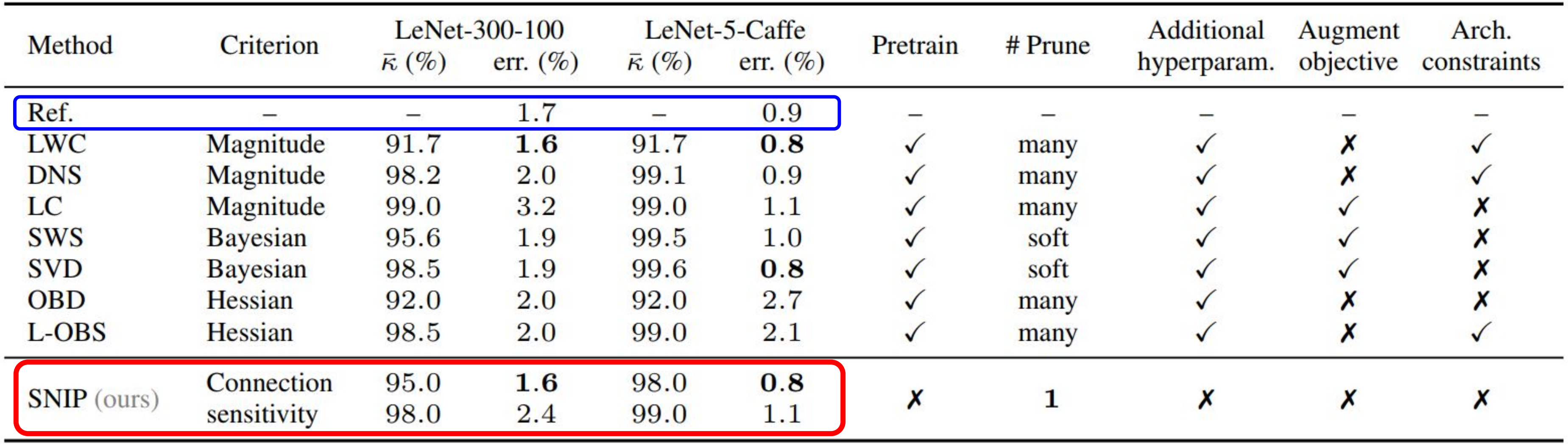
Method	Criterion	LeNet-300-100		LeNet-5-Caffe		Pretrain	# Prune	Additional hyperparam.	Augment objective	Arch. constraints
		$\bar{\kappa}$ (%)	err. (%)	$\bar{\kappa}$ (%)	err. (%)					
Ref.	–	–	1.7	–	0.9	–	–	–	–	–
LWC	Magnitude	91.7	**1.6**	91.7	**0.8**	✓	many	✓	✗	✓
DNS	Magnitude	98.2	2.0	99.1	0.9	✓	many	✓	✗	✓
LC	Magnitude	99.0	3.2	99.0	1.1	✓	many	✓	✓	✗
SWS	Bayesian	95.6	1.9	99.5	1.0	✓	soft	✓	✓	✗
SVD	Bayesian	98.5	1.9	99.6	**0.8**	✓	soft	✓	✓	✗
OBD	Hessian	92.0	2.0	92.0	2.7	✓	many	✓	✗	✗
L-OBS	Hessian	98.5	2.0	99.0	2.1	✓	many	✓	✗	✓
SNIP (ours)	Connection sensitivity	95.0	**1.6**	98.0	**0.8**	✗	**1**	✗	✗	✗
		98.0	2.4	99.0	1.1					

LeNets: comparison to SOTA



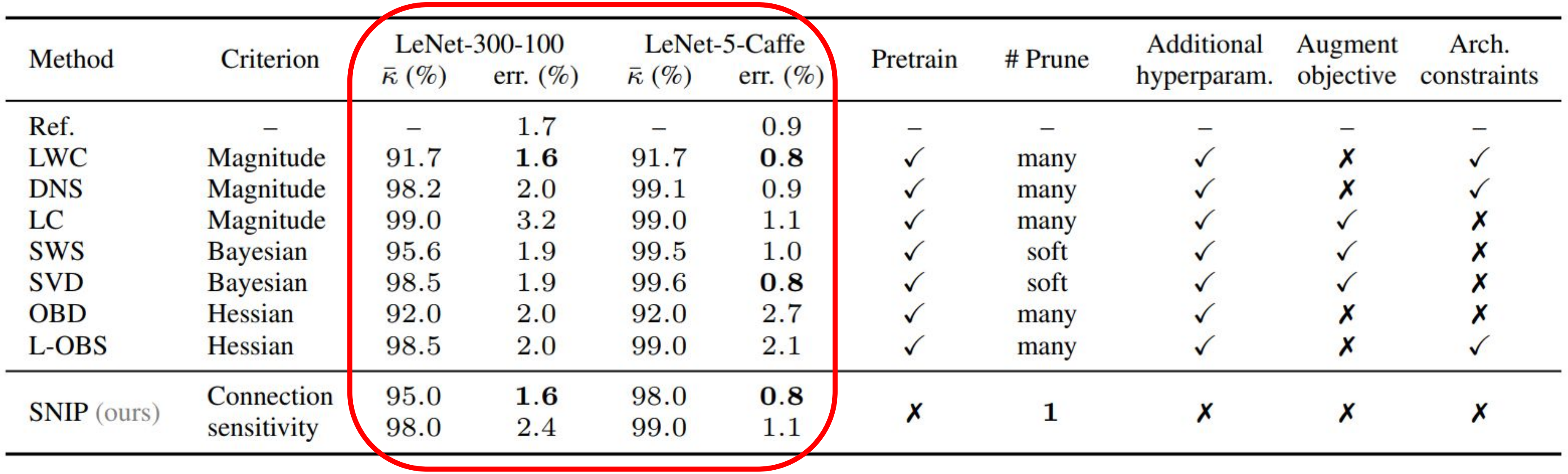
Method	Criterion	LeNet-300-100		LeNet-5-Caffe		Pretrain	# Prune	Additional hyperparam.	Augment objective	Arch. constraints
		$\bar{\kappa}$ (%)	err. (%)	$\bar{\kappa}$ (%)	err. (%)					
Ref.	–	–	1.7	–	0.9	–	–	–	–	–
LWC	Magnitude	91.7	**1.6**	91.7	**0.8**	✓	many	✓	✗	✓
DNS	Magnitude	98.2	2.0	99.1	0.9	✓	many	✓	✗	✓
LC	Magnitude	99.0	3.2	99.0	1.1	✓	many	✓	✓	✗
SWS	Bayesian	95.6	1.9	99.5	1.0	✓	soft	✓	✓	✗
SVD	Bayesian	98.5	1.9	99.6	**0.8**	✓	soft	✓	✓	✗
OBD	Hessian	92.0	2.0	92.0	2.7	✓	many	✓	✗	✗
L-OBS	Hessian	98.5	2.0	99.0	2.1	✓	many	✓	✗	✓
SNIP (ours)	Connection sensitivity	95.0	**1.6**	98.0	**0.8**	✗	**1**	✗	✗	✗
		98.0	2.4	99.0	1.1					

LeNets: comparison to SOTA



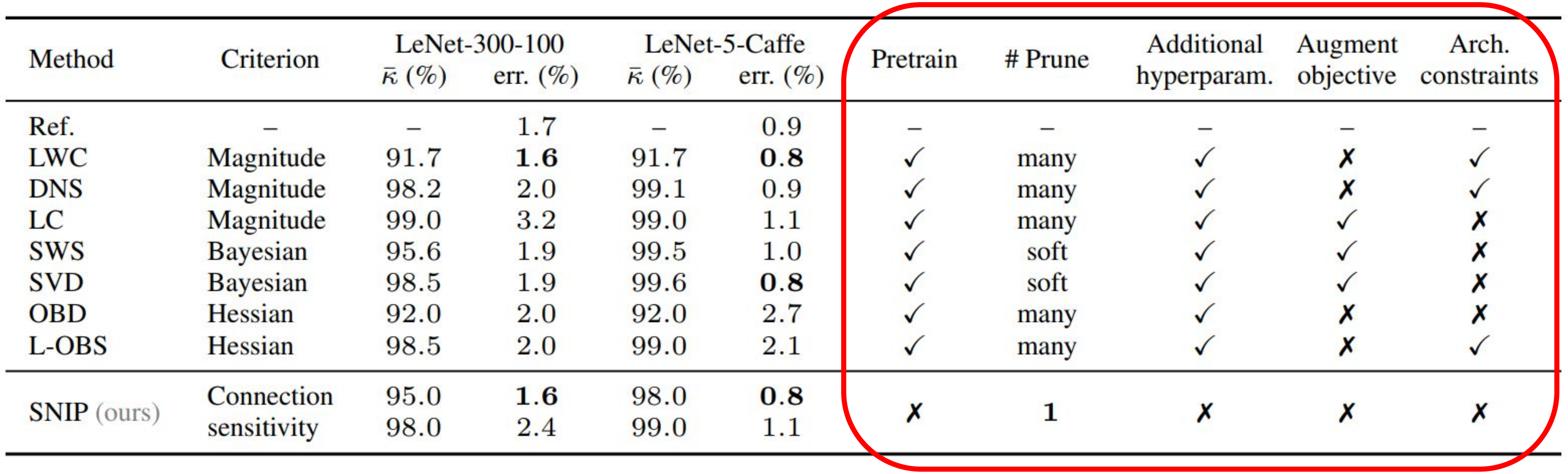
Method	Criterion	LeNet-300-100		LeNet-5-Caffe		Pretrain	# Prune	Additional hyperparam.	Augment objective	Arch. constraints
		$\bar{\kappa}$ (%)	err. (%)	$\bar{\kappa}$ (%)	err. (%)					
Ref.	–	–	1.7	–	0.9	–	–	–	–	–
LWC	Magnitude	91.7	**1.6**	91.7	**0.8**	✓	many	✓	✗	✓
DNS	Magnitude	98.2	2.0	99.1	0.9	✓	many	✓	✗	✓
LC	Magnitude	99.0	3.2	99.0	1.1	✓	many	✓	✓	✗
SWS	Bayesian	95.6	1.9	99.5	1.0	✓	soft	✓	✓	✗
SVD	Bayesian	98.5	1.9	99.6	**0.8**	✓	soft	✓	✓	✗
OBD	Hessian	92.0	2.0	92.0	2.7	✓	many	✓	✗	✗
L-OBS	Hessian	98.5	2.0	99.0	2.1	✓	many	✓	✗	✓
SNIP (ours)	Connection sensitivity	95.0	**1.6**	98.0	**0.8**	✗	**1**	✗	✗	✗
		98.0	2.4	99.0	1.1					

LeNets: comparison to SOTA



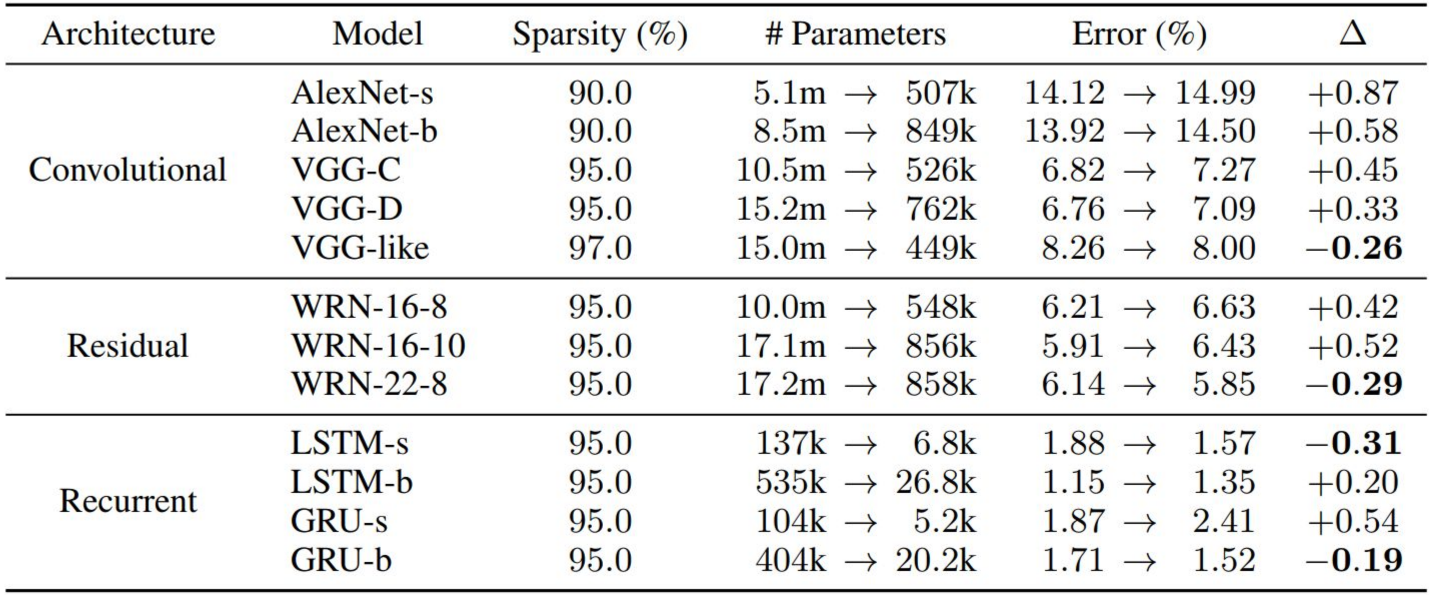
Method	Criterion	LeNet-300-100		LeNet-5-Caffe		Pretrain	# Prune	Additional hyperparam.	Augment objective	Arch. constraints
		$\bar{\kappa}$ (%)	err. (%)	$\bar{\kappa}$ (%)	err. (%)					
Ref.	–	–	1.7	–	0.9	–	–	–	–	–
LWC	Magnitude	91.7	**1.6**	91.7	**0.8**	✓	many	✓	✗	✓
DNS	Magnitude	98.2	2.0	99.1	0.9	✓	many	✓	✗	✓
LC	Magnitude	99.0	3.2	99.0	1.1	✓	many	✓	✓	✗
SWS	Bayesian	95.6	1.9	99.5	1.0	✓	soft	✓	✓	✗
SVD	Bayesian	98.5	1.9	99.6	**0.8**	✓	soft	✓	✓	✗
OBD	Hessian	92.0	2.0	92.0	2.7	✓	many	✓	✗	✗
L-OBS	Hessian	98.5	2.0	99.0	2.1	✓	many	✓	✗	✓
SNIP (ours)	Connection sensitivity	95.0	**1.6**	98.0	**0.8**	✗	**1**	✗	✗	✗
		98.0	2.4	99.0	1.1					

LeNets: comparison to SOTA



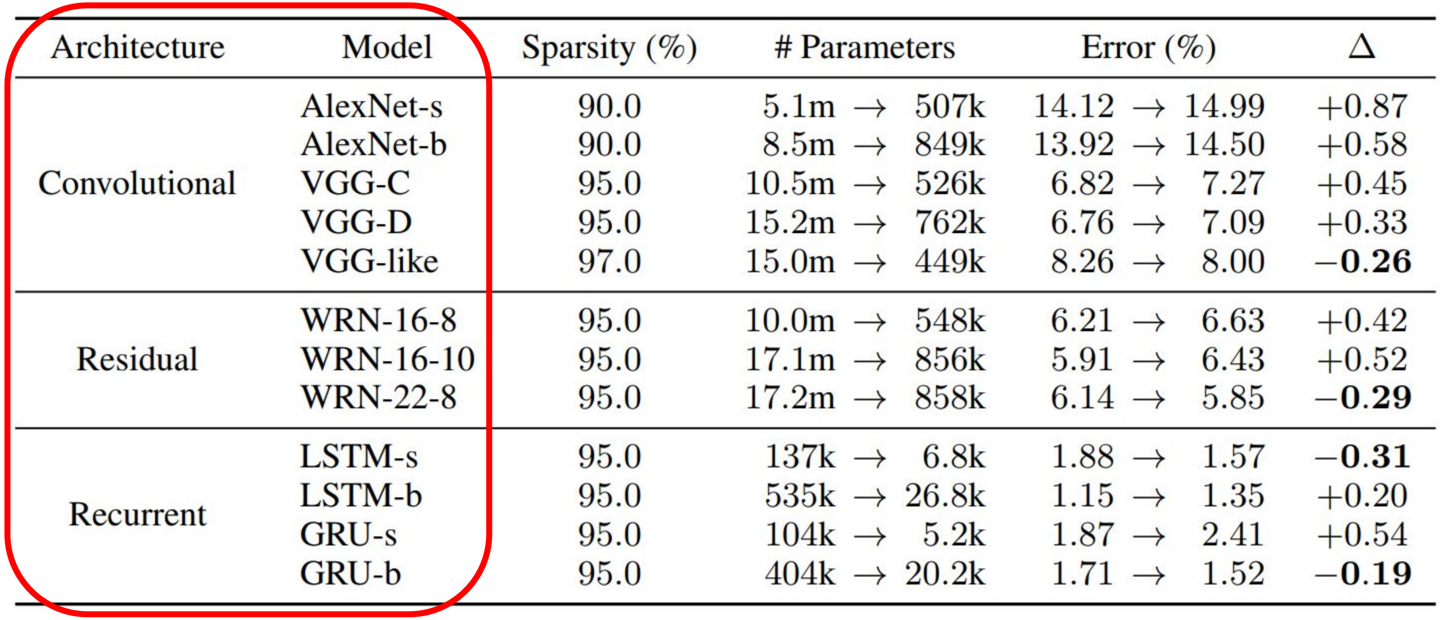
Method	Criterion	LeNet-300-100		LeNet-5-Caffe		Pretrain	# Prune	Additional hyperparam.	Augment objective	Arch. constraints
		$\bar{\kappa}$ (%)	err. (%)	$\bar{\kappa}$ (%)	err. (%)					
Ref.	–	–	1.7	–	0.9	–	–	–	–	–
LWC	Magnitude	91.7	**1.6**	91.7	**0.8**	✓	many	✓	✗	✓
DNS	Magnitude	98.2	2.0	99.1	0.9	✓	many	✓	✗	✓
LC	Magnitude	99.0	3.2	99.0	1.1	✓	many	✓	✓	✗
SWS	Bayesian	95.6	1.9	99.5	1.0	✓	soft	✓	✓	✗
SVD	Bayesian	98.5	1.9	99.6	**0.8**	✓	soft	✓	✓	✗
OBD	Hessian	92.0	2.0	92.0	2.7	✓	many	✓	✗	✗
L-OBS	Hessian	98.5	2.0	99.0	2.1	✓	many	✓	✗	✓
SNIP (ours)	Connection sensitivity	95.0	**1.6**	98.0	**0.8**	✗	**1**	✗	✗	✗
		98.0	2.4	99.0	1.1					

Various architectures & models



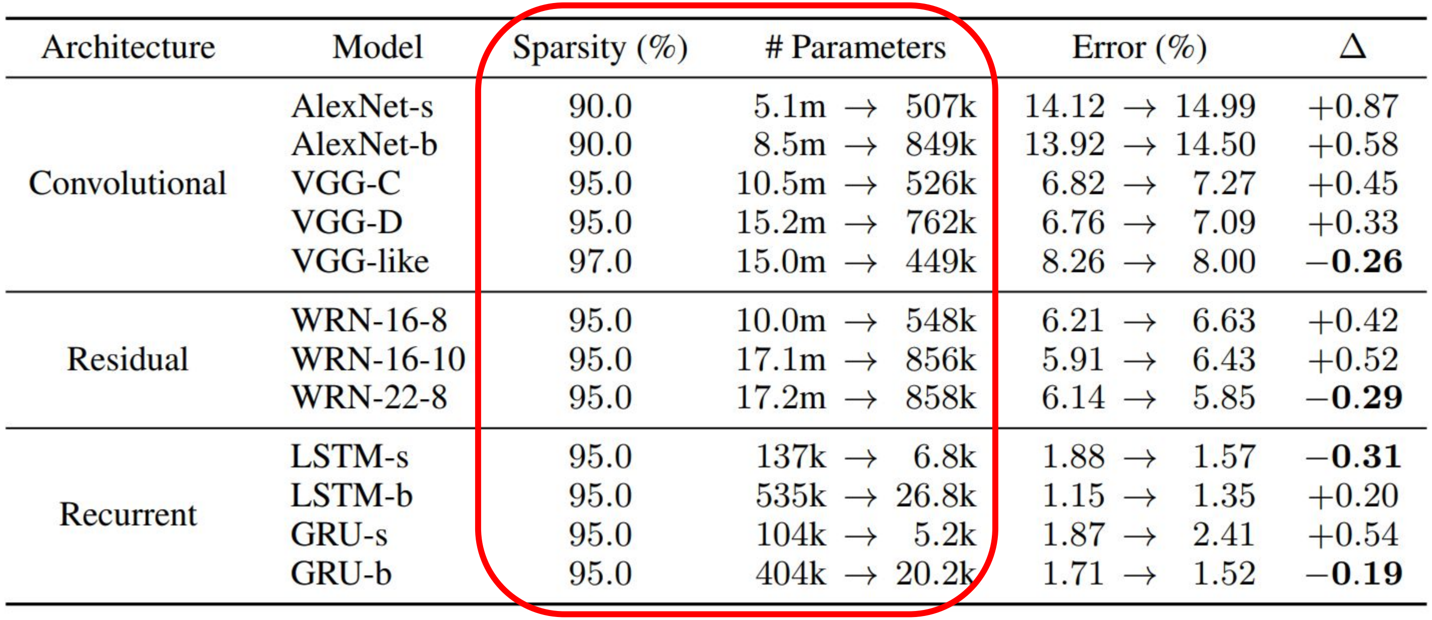
Architecture	Model	Sparsity (%)	# Parameters	Error (%)	Δ
Convolutional	AlexNet-s	90.0	5.1m → 507k	14.12 → 14.99	+0.87
	AlexNet-b	90.0	8.5m → 849k	13.92 → 14.50	+0.58
	VGG-C	95.0	10.5m → 526k	6.82 → 7.27	+0.45
	VGG-D	95.0	15.2m → 762k	6.76 → 7.09	+0.33
	VGG-like	97.0	15.0m → 449k	8.26 → 8.00	**−0.26**
Residual	WRN-16-8	95.0	10.0m → 548k	6.21 → 6.63	+0.42
	WRN-16-10	95.0	17.1m → 856k	5.91 → 6.43	+0.52
	WRN-22-8	95.0	17.2m → 858k	6.14 → 5.85	**−0.29**
Recurrent	LSTM-s	95.0	137k → 6.8k	1.88 → 1.57	**−0.31**
	LSTM-b	95.0	535k → 26.8k	1.15 → 1.35	+0.20
	GRU-s	95.0	104k → 5.2k	1.87 → 2.41	+0.54
	GRU-b	95.0	404k → 20.2k	1.71 → 1.52	**−0.19**

Various architectures & models



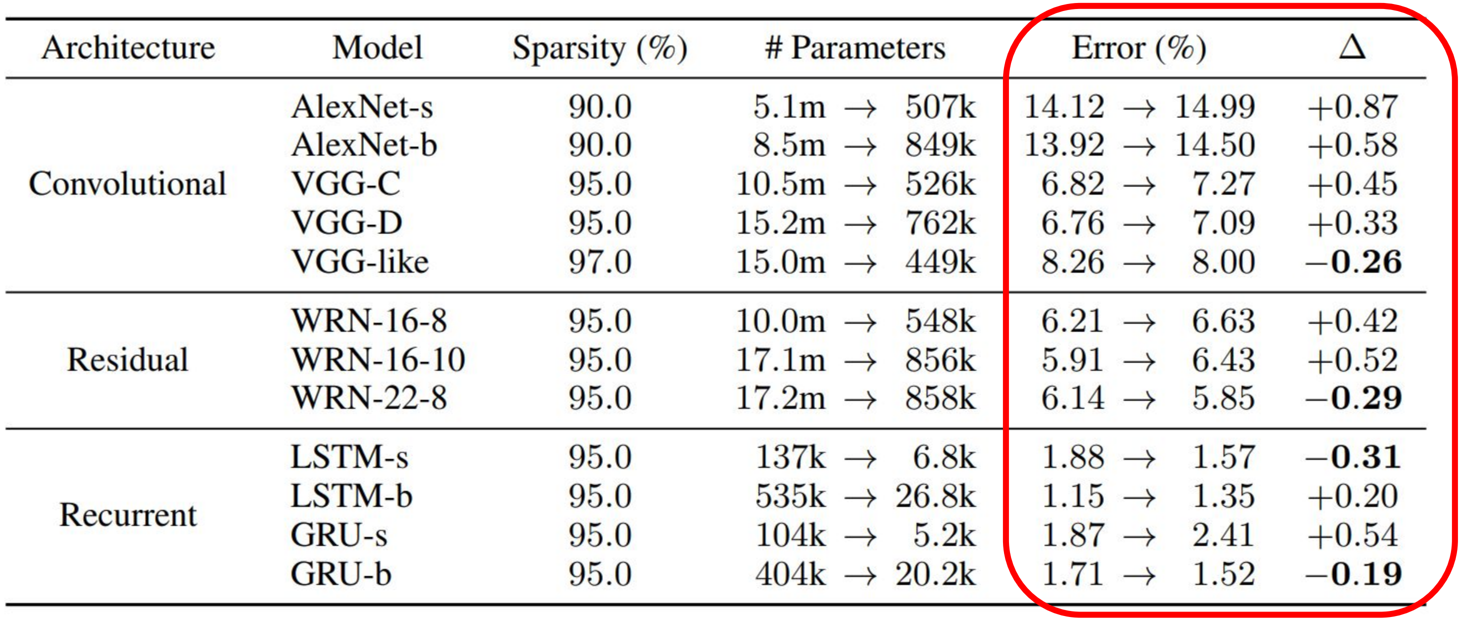
Architecture	Model	Sparsity (%)	# Parameters	Error (%)	Δ
Convolutional	AlexNet-s	90.0	5.1m $\rightarrow$ 507k	14.12 $\rightarrow$ 14.99	+0.87
	AlexNet-b	90.0	8.5m $\rightarrow$ 849k	13.92 $\rightarrow$ 14.50	+0.58
	VGG-C	95.0	10.5m $\rightarrow$ 526k	6.82 $\rightarrow$ 7.27	+0.45
	VGG-D	95.0	15.2m $\rightarrow$ 762k	6.76 $\rightarrow$ 7.09	+0.33
	VGG-like	97.0	15.0m $\rightarrow$ 449k	8.26 $\rightarrow$ 8.00	−**0.26**
Residual	WRN-16-8	95.0	10.0m $\rightarrow$ 548k	6.21 $\rightarrow$ 6.63	+0.42
	WRN-16-10	95.0	17.1m $\rightarrow$ 856k	5.91 $\rightarrow$ 6.43	+0.52
	WRN-22-8	95.0	17.2m $\rightarrow$ 858k	6.14 $\rightarrow$ 5.85	−**0.29**
Recurrent	LSTM-s	95.0	137k $\rightarrow$ 6.8k	1.88 $\rightarrow$ 1.57	−**0.31**
	LSTM-b	95.0	535k $\rightarrow$ 26.8k	1.15 $\rightarrow$ 1.35	+0.20
	GRU-s	95.0	104k $\rightarrow$ 5.2k	1.87 $\rightarrow$ 2.41	+0.54
	GRU-b	95.0	404k $\rightarrow$ 20.2k	1.71 $\rightarrow$ 1.52	−**0.19**

Various architectures & models



Architecture	Model	Sparsity (%)	# Parameters	Error (%)	Δ
Convolutional	AlexNet-s	90.0	5.1m → 507k	14.12 → 14.99	+0.87
	AlexNet-b	90.0	8.5m → 849k	13.92 → 14.50	+0.58
	VGG-C	95.0	10.5m → 526k	6.82 → 7.27	+0.45
	VGG-D	95.0	15.2m → 762k	6.76 → 7.09	+0.33
	VGG-like	97.0	15.0m → 449k	8.26 → 8.00	**−0.26**
Residual	WRN-16-8	95.0	10.0m → 548k	6.21 → 6.63	+0.42
	WRN-16-10	95.0	17.1m → 856k	5.91 → 6.43	+0.52
	WRN-22-8	95.0	17.2m → 858k	6.14 → 5.85	**−0.29**
Recurrent	LSTM-s	95.0	137k → 6.8k	1.88 → 1.57	**−0.31**
	LSTM-b	95.0	535k → 26.8k	1.15 → 1.35	+0.20
	GRU-s	95.0	104k → 5.2k	1.87 → 2.41	+0.54
	GRU-b	95.0	404k → 20.2k	1.71 → 1.52	**−0.19**

Various architectures & models



Architecture	Model	Sparsity (%)	# Parameters	Error (%)	Δ
Convolutional	AlexNet-s	90.0	5.1m → 507k	14.12 → 14.99	+0.87
	AlexNet-b	90.0	8.5m → 849k	13.92 → 14.50	+0.58
	VGG-C	95.0	10.5m → 526k	6.82 → 7.27	+0.45
	VGG-D	95.0	15.2m → 762k	6.76 → 7.09	+0.33
	VGG-like	97.0	15.0m → 449k	8.26 → 8.00	−**0.26**
Residual	WRN-16-8	95.0	10.0m → 548k	6.21 → 6.63	+0.42
	WRN-16-10	95.0	17.1m → 856k	5.91 → 6.43	+0.52
	WRN-22-8	95.0	17.2m → 858k	6.14 → 5.85	−**0.29**
Recurrent	LSTM-s	95.0	137k → 6.8k	1.88 → 1.57	−**0.31**
	LSTM-b	95.0	535k → 26.8k	1.15 → 1.35	+0.20
	GRU-s	95.0	104k → 5.2k	1.87 → 2.41	+0.54
	GRU-b	95.0	404k → 20.2k	1.71 → 1.52	−**0.19**

Which parameters are being pruned?

Consider: LeNet-300-100 ($w_{l=1} \in R^{784\times300}$)

Steps:

1. Curate a mini-batch of examples
2. Compute the connection sensitivity
3. Prune: create the mask c
4. Visualize c for the first layer
   (i.e., $c_{l=1} \in \{0,1\}^{784\times300} \rightarrow R^{784} \rightarrow R^{28\times28}$)

Which parameters are being pruned?

Consider: LeNet-300-100 ($w_{l=1} \in R^{784\times300}$)

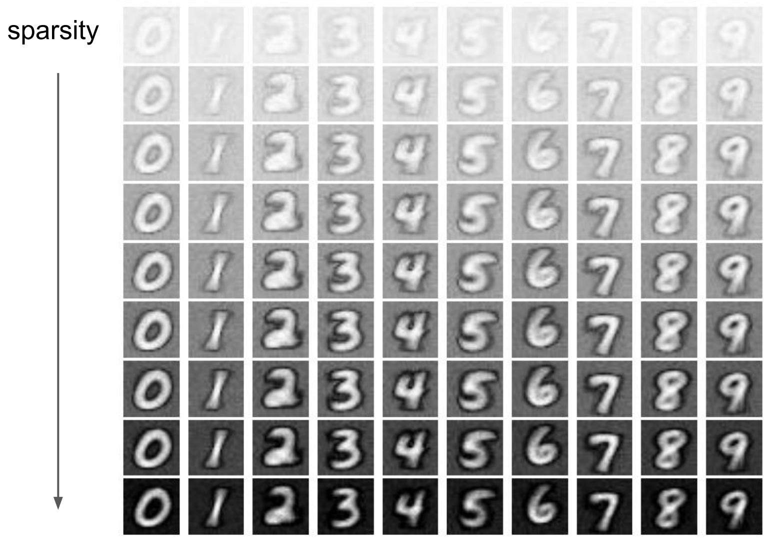
Steps:

1. Curate a mini-batch of examples
2. Compute the connection sensitivity
3. Prune: create the mask c
4. Visualize c for the first layer
   (i.e., $c_{l=1} \in \{0,1\}^{784\times300} \rightarrow R^{784} \rightarrow R^{28\times28}$)

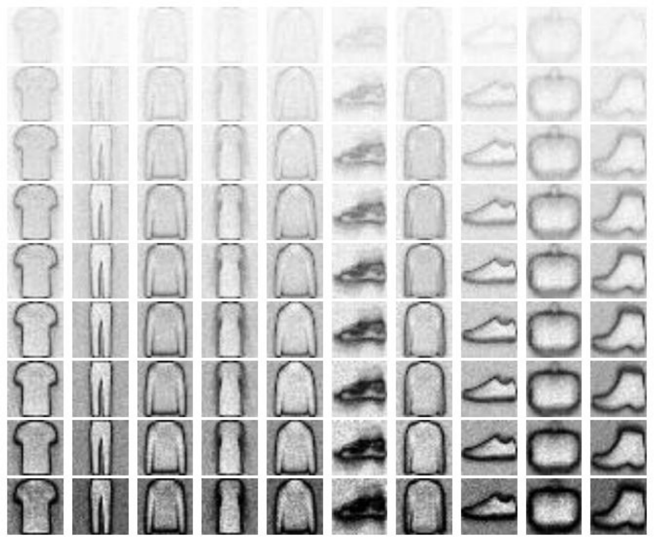
The input was digit 8.

<u>The parameters outside the digit tend to be pruned away, letting the network to receive information selectively from a distinctive region.</u>

Which parameters are being pruned?



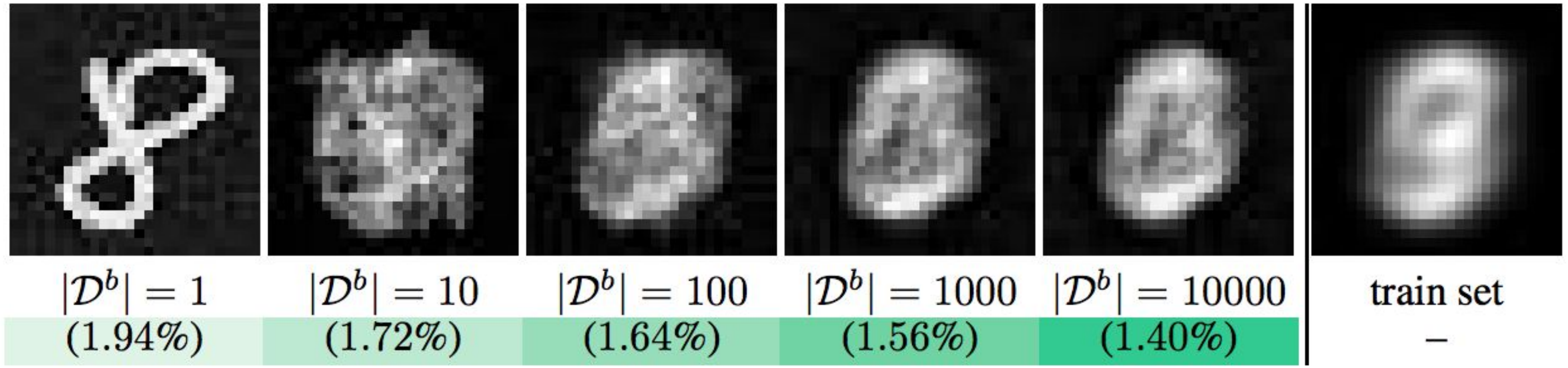
(a) MNIST



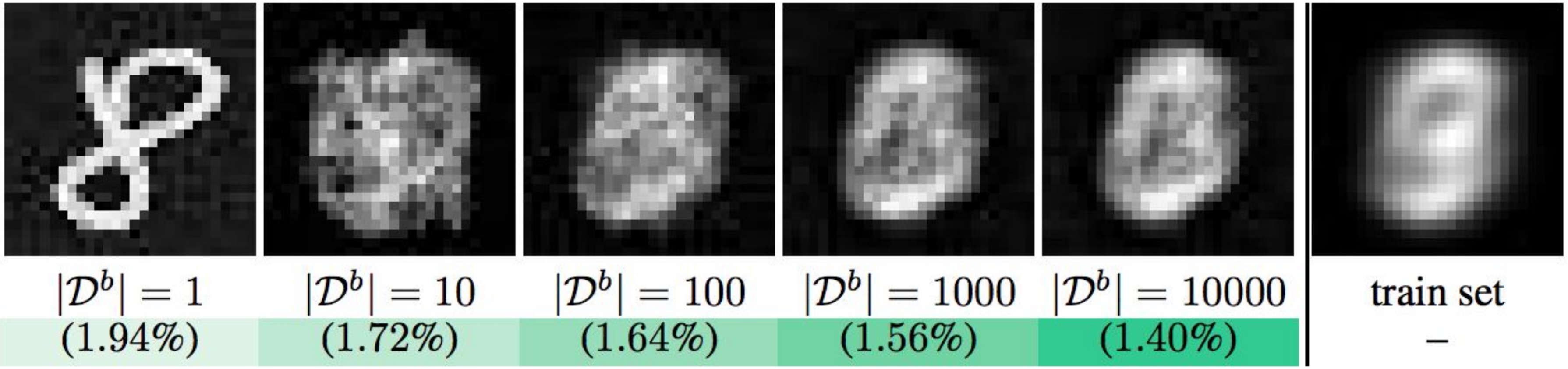
(b) Fashion-MNIST

<u>The parameters connected to the discriminative part of image are retained.</u>

Which parameters are being pruned?



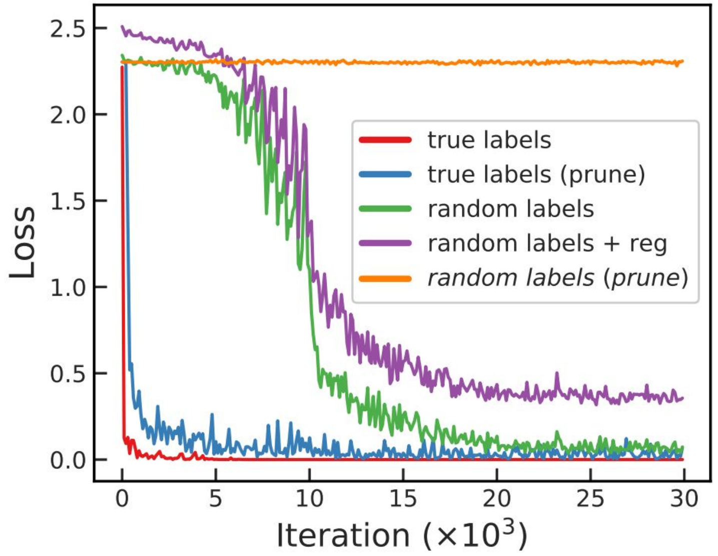
Which parameters are being pruned?



$|\mathcal{D}^b| = 1$ (1.94%)
$|\mathcal{D}^b| = 10$ (1.72%)
$|\mathcal{D}^b| = 100$ (1.64%)
$|\mathcal{D}^b| = 1000$ (1.56%)
$|\mathcal{D}^b| = 10000$ (1.40%)
train set –

Carrying out such inspection is not straightforward with other methods.

Prevent memorization



[Fitting random labels]
Understanding deep learning requires rethinking generalization, Zhang et al. ICLR'17

The pruned network does not have sufficient capacity to fit the random labels, but is capable of performing the task.

SNIP

Paper:

https://arxiv.org/abs/1810.02340

Code:

https://github.com/namhoonlee/snip-public

Contact:

http://www.robots.ox.ac.uk/~namhoon/

A signal propagation perspective for pruning neural networks at initialization

Namhoon Lee[1], Thalaiyasingam Ajanthan[2], Stephen Gould[2], Philip Torr[1]

[1]University of Oxford, [2]Australian National University

Network pruning

Designing pruning algorithms has been often purely based on ad-hoc intuition lacking rigorous underpinning, partly because pruning was typically carried out after training the model as a post-processing step or interwove with the training procedure, without adequate tools to analyze.

Motivation

Recently, Lee et al. (2019) have shown that pruning can be done on randomly initialized neural networks in a single-shot prior to training (i.e., pruning at initialization). They empirically showed that as long as the initial random weights are drawn from appropriately scaled Gaussians (e.g., Glorot & Bengio (2010)), their pruning criterion called connection sensitivity can be used to prune deep neural networks, often to an extreme level of sparsity while maintaining good accuracy once trained.

However, it remains unclear as to why pruning at initialization is effective, how it should be understood theoretically and whether it can be extended further.

Motivation

Recently, Lee et al. (2019) have shown that pruning can be done on randomly initialized neural networks in a single-shot prior to training (i.e., pruning at initialization). They empirically showed that as long as the initial random weights are drawn from appropriately scaled Gaussians (e.g., Glorot & Bengio (2010)), their pruning criterion called connection sensitivity can be used to prune deep neural networks, often to an extreme level of sparsity while maintaining good accuracy once trained.

However, it remains unclear as to why pruning at initialization is effective, how it should be understood theoretically and whether it can be extended further.

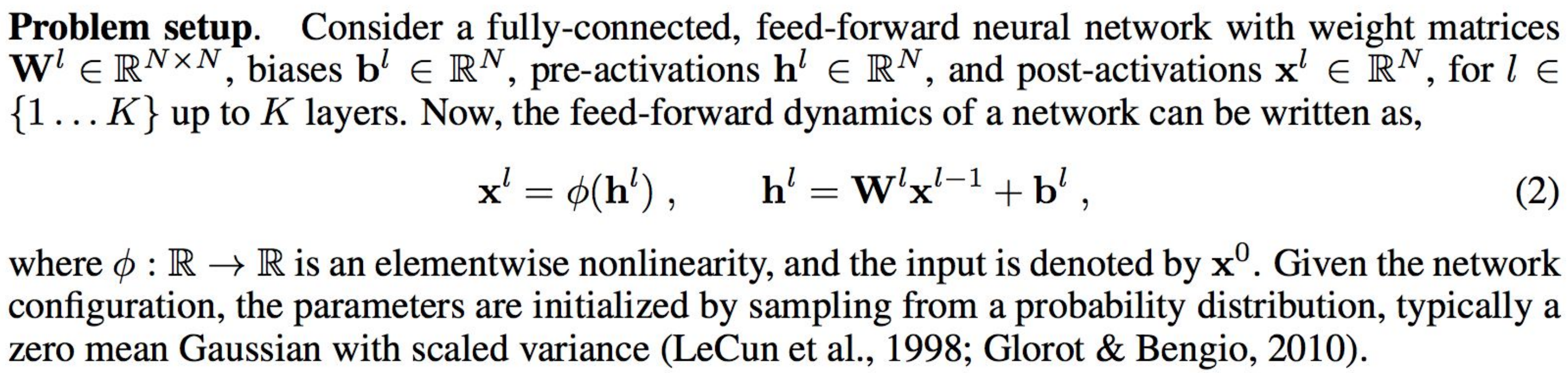
Motivation

Recently, Lee et al. (2019) have shown that pruning can be done on randomly initialized neural networks in a single-shot prior to training (i.e., pruning at initialization). They empirically showed that as long as the initial random weights are drawn from appropriately scaled Gaussians (e.g., Glorot & Bengio (2010)), their pruning criterion called connection sensitivity can be used to prune deep neural networks, often to an extreme level of sparsity while maintaining good accuracy once trained.

However, it remains unclear as to why pruning at initialization is effective, how it should be understood theoretically and whether it can be extended further.

Observations

Effect of initialization on pruning: Setup



Problem setup. Consider a fully-connected, feed-forward neural network with weight matrices $\mathbf{W}^l \in \mathbb{R}^{N \times N}$, biases $\mathbf{b}^l \in \mathbb{R}^N$, pre-activations $\mathbf{h}^l \in \mathbb{R}^N$, and post-activations $\mathbf{x}^l \in \mathbb{R}^N$, for $l \in \{1 \ldots K\}$ up to K layers. Now, the feed-forward dynamics of a network can be written as,

$$\mathbf{x}^l = \phi(\mathbf{h}^l) , \qquad \mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l , \tag{2}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is an elementwise nonlinearity, and the input is denoted by $\mathbf{x}^0$. Given the network configuration, the parameters are initialized by sampling from a probability distribution, typically a zero mean Gaussian with scaled variance (LeCun et al., 1998; Glorot & Bengio, 2010).

Effect of initialization on pruning: Setup

Variance scaling initialization schemes (VS-{L, G, H}):

$\sigma \rightarrow (\alpha / \psi_l)\, \sigma$, where ψ_l is a layerwise scalar that depends on an architecture specification (e.g., fan-in), and α is a global scalar.

Network with layers of the same width:

the variance can be controlled by a single scalar $\gamma = \alpha / \psi$ as $\psi_l = \psi$ for all layers l.

Effect of initialization on pruning: Setup

Variance scaling initialization schemes (VS-{L, G, H}):

$\sigma \rightarrow (\alpha / \psi_l)\, \sigma$, where ψ_l is a layerwise scalar that depends on an architecture specification (e.g., fan-in), and α is a global scalar.

Network with layers of the same width:

the variance can be controlled by a single scalar $\gamma = \alpha / \psi$ as $\psi_l = \psi$ for all layers l.

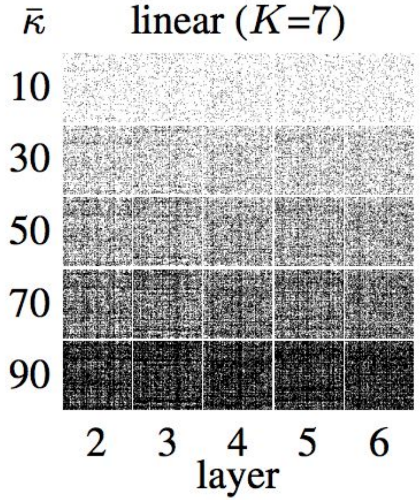
Consider:

Linear and tanh MLP networks of $K = 7$ and $N = 100$ on MNIST with $\sigma = 1$ as the default, similar to Saxe et al. (2014).

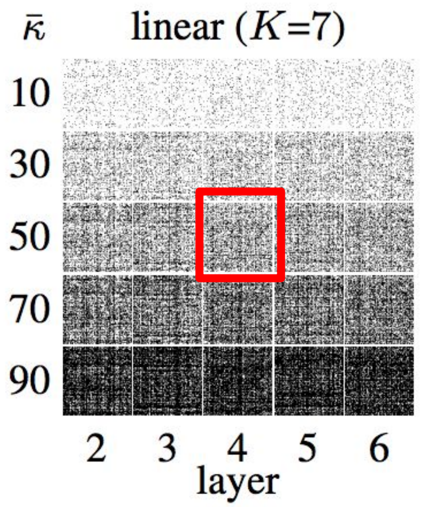
Experiments:

Initialize with different γ, compute CS, prune, and then visualize c.

Effect of initialization on pruning: Results

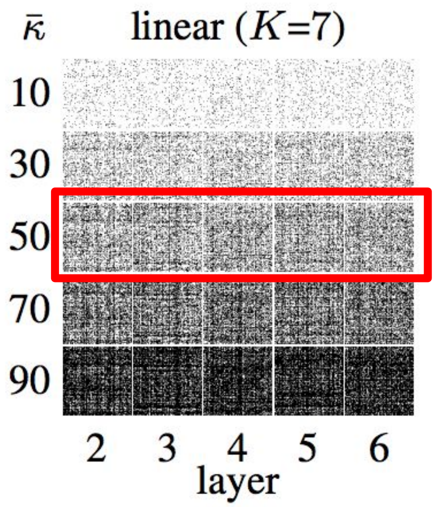


Effect of initialization on pruning: Results



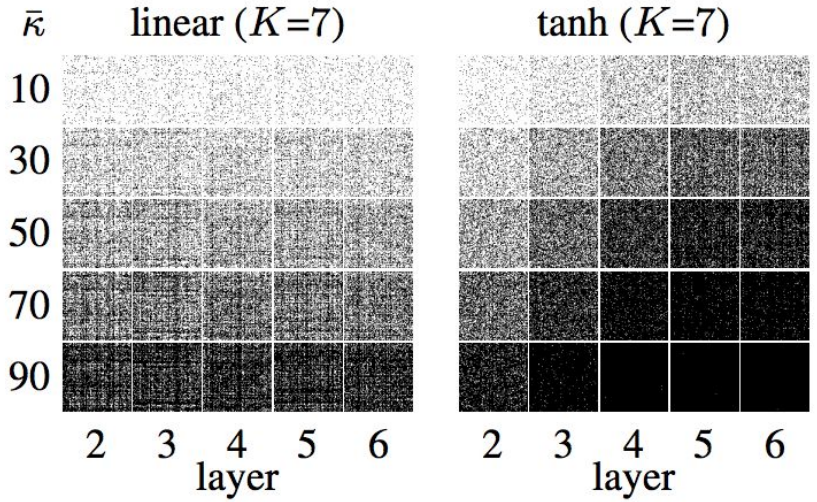
(each box) the pruning mask $c_l \in \{0,1\}^{100\text{x}100}$

Effect of initialization on pruning: Results



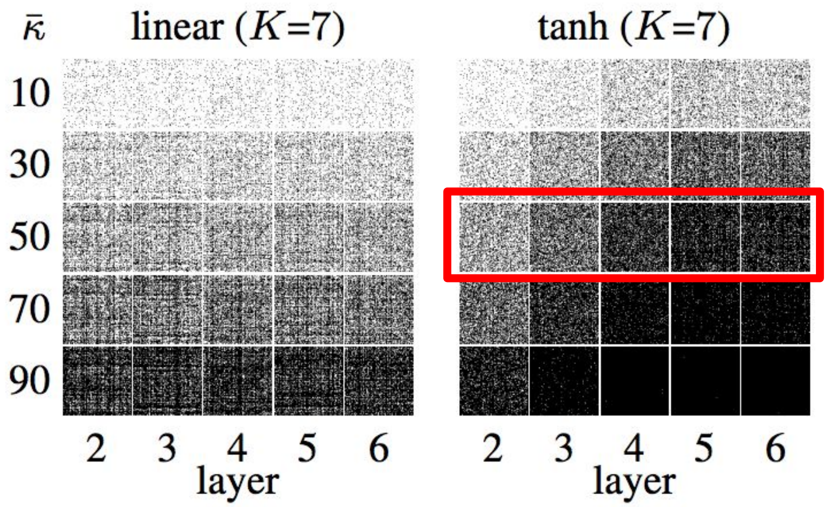
For the linear network, parameters are pruned uniformly throughout the network.

Effect of initialization on pruning: Results



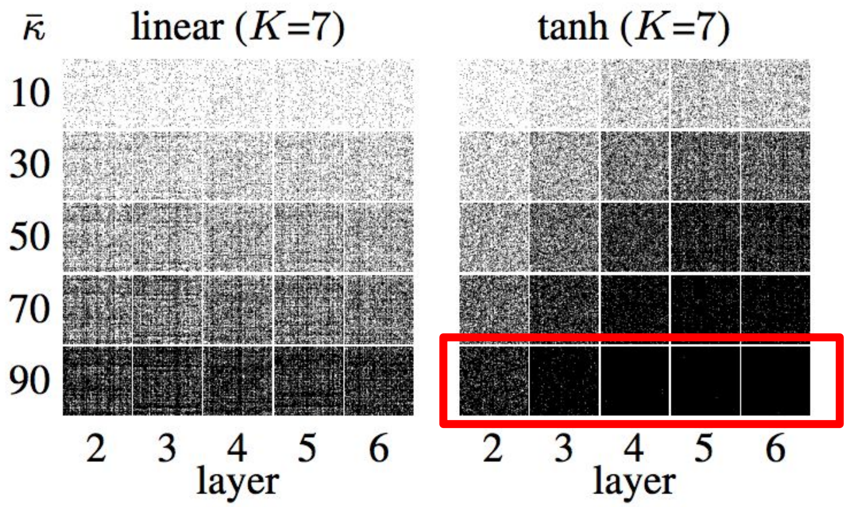
For the tanh case, more parameters tend to be pruned in the later layers.

Effect of initialization on pruning: Results



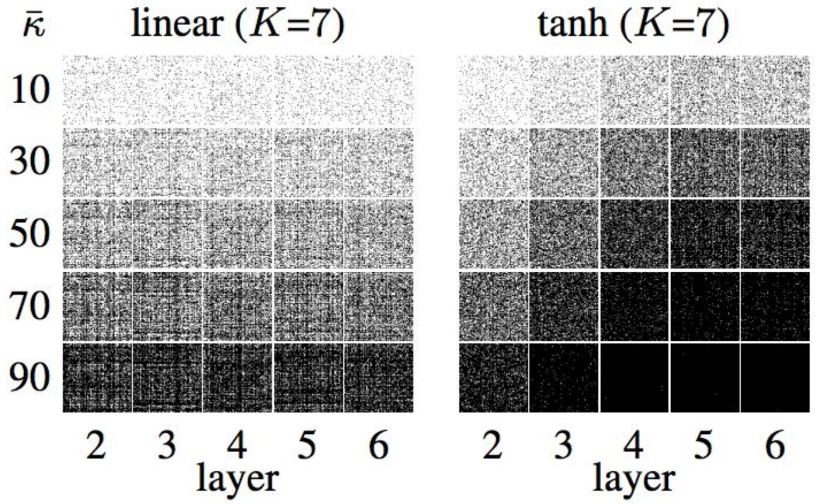
For the tanh case, more parameters tend to be pruned in the later layers.

Effect of initialization on pruning: Results

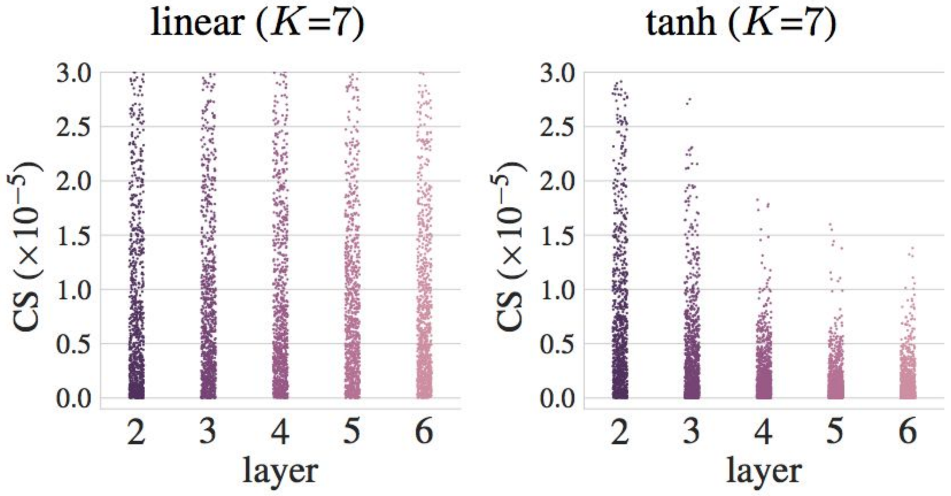


When pruning for a high sparsity level (e.g., 90%), this becomes critical and leads to poor learning capability as there are only a few parameters left in later layers.

Effect of initialization on pruning: Results

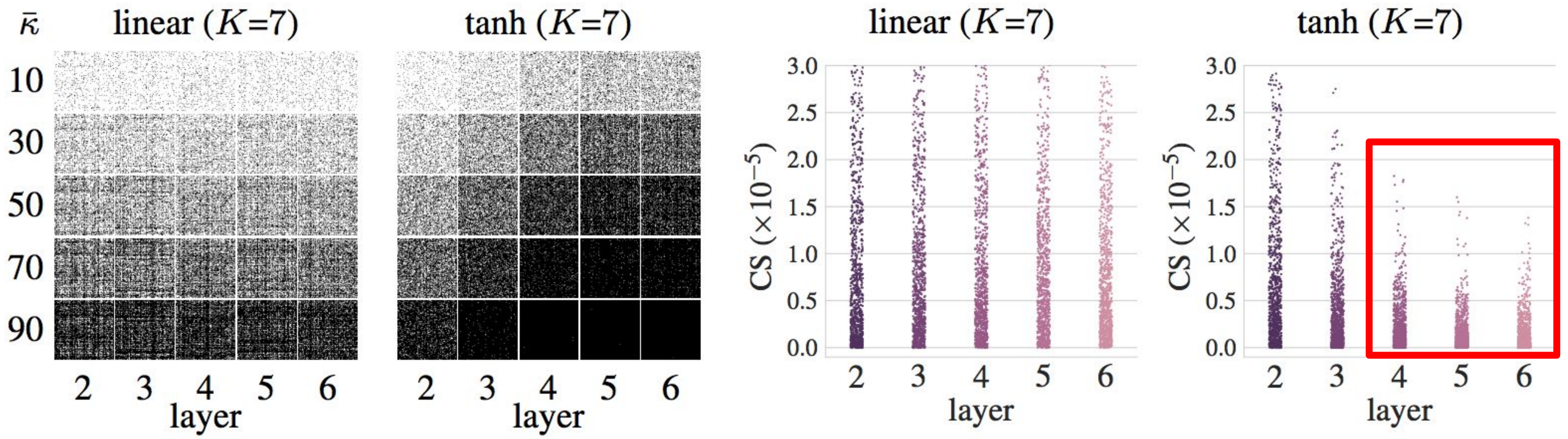


Pruning patterns (c)



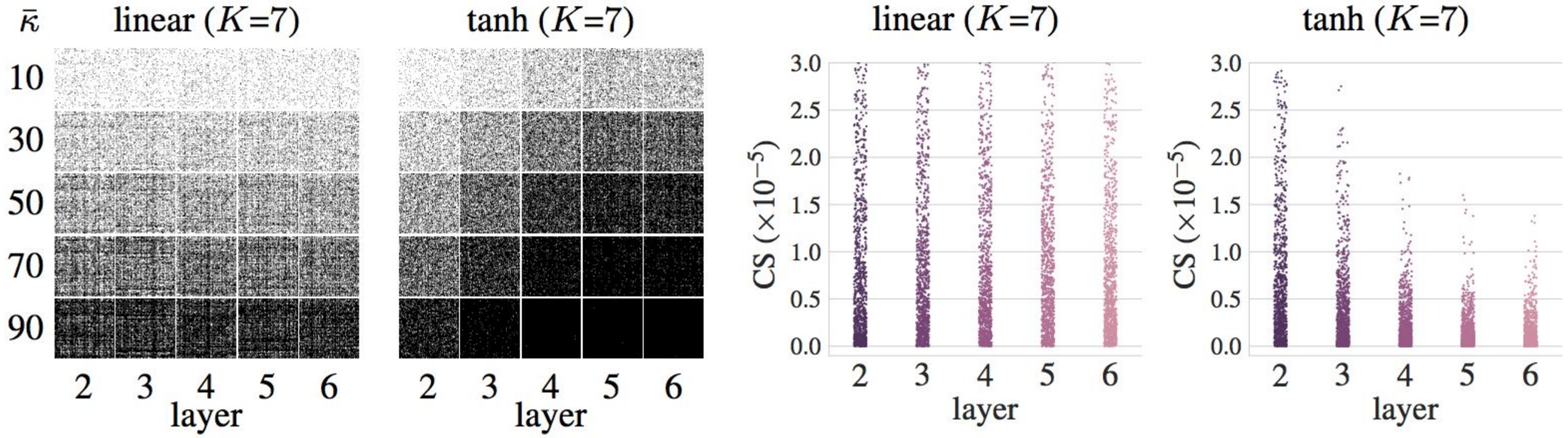
Parameter sensitivity scores (CS)

Effect of initialization on pruning: Results



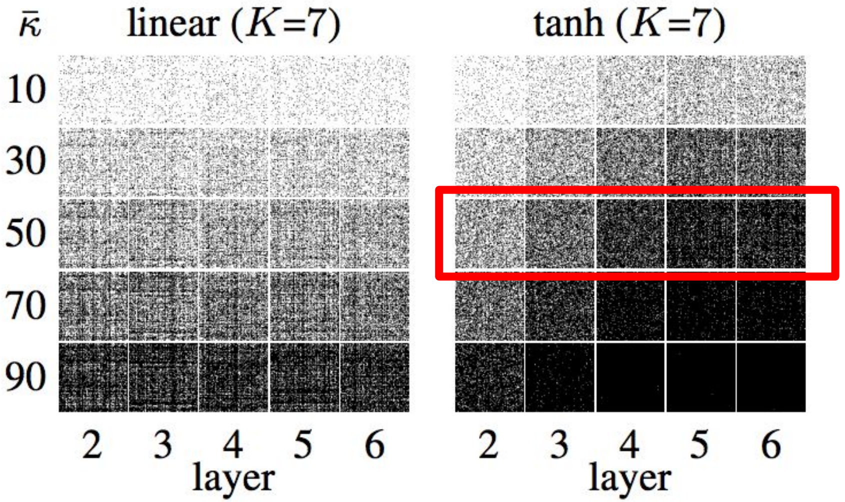
CS tends to decrease towards the later layers

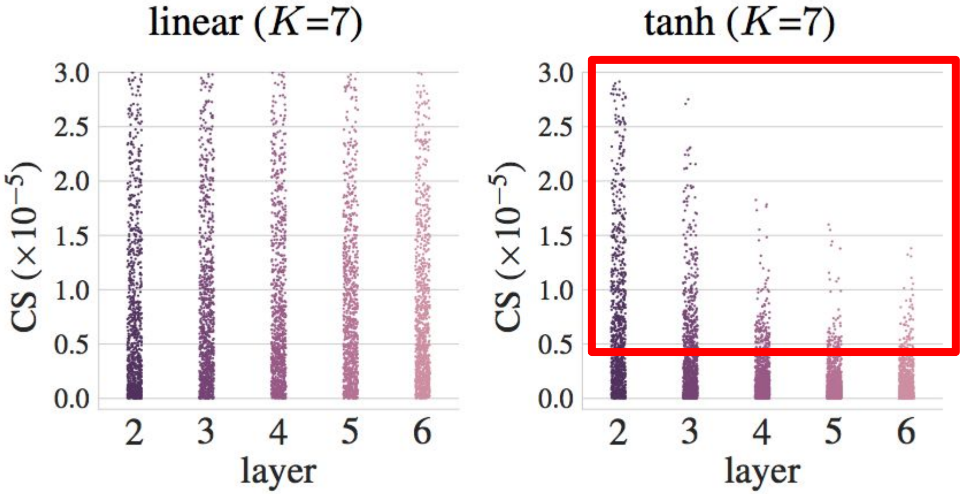
Effect of initialization on pruning: Results



CS tends to decrease towards the later layers → choosing the top-κ globally

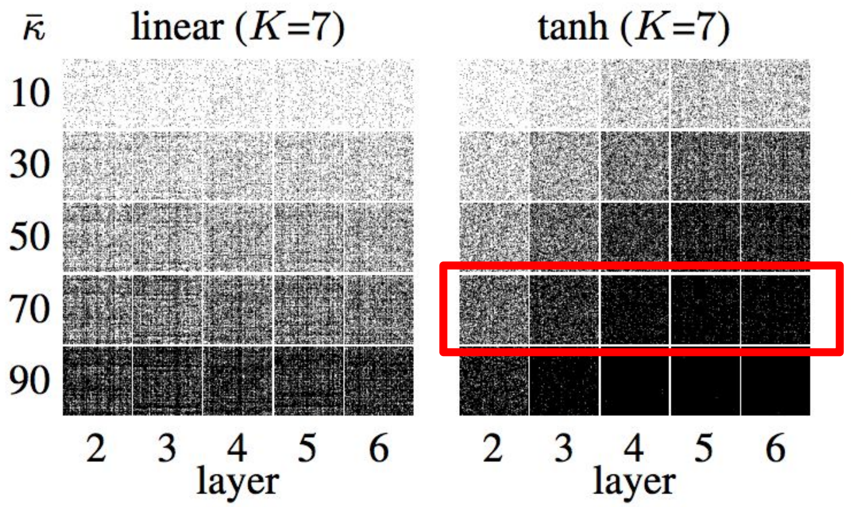
Effect of initialization on pruning: Results

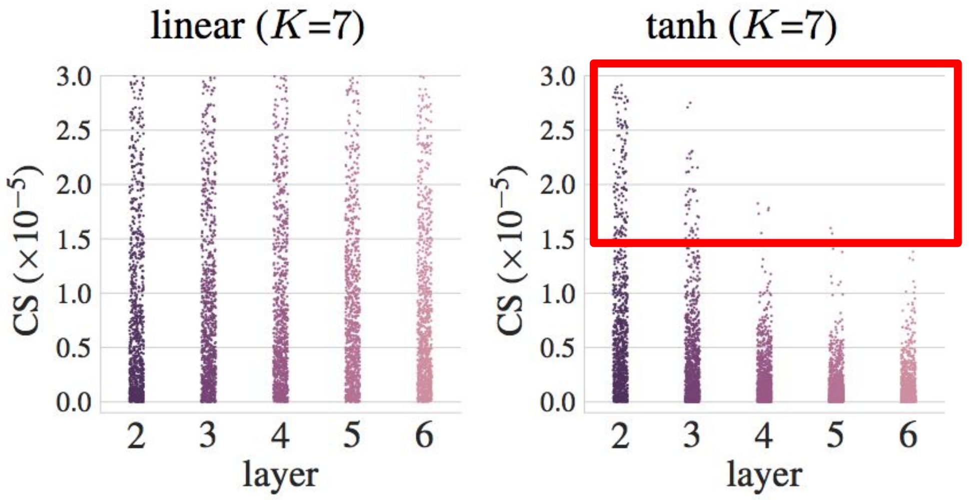




CS tends to decrease towards the later layers → choosing the top-κ globally

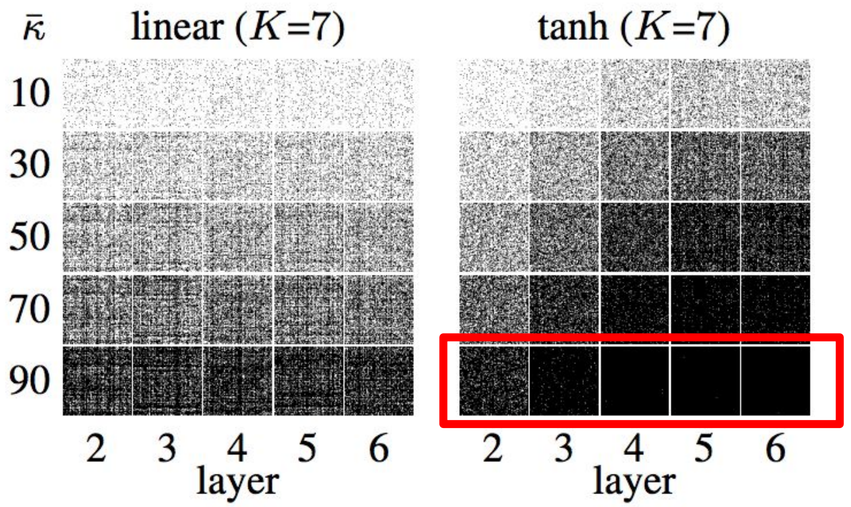
Effect of initialization on pruning: Results

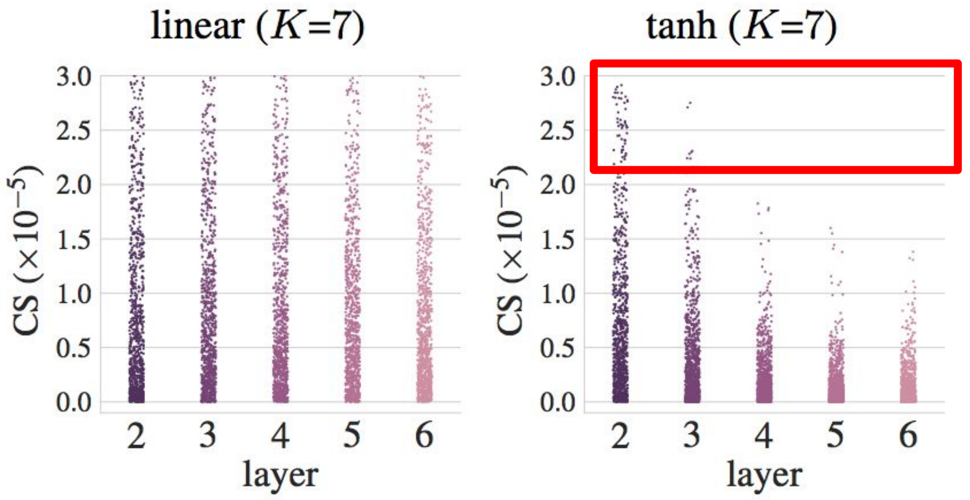




CS tends to decrease towards the later layers → choosing the top-κ globally

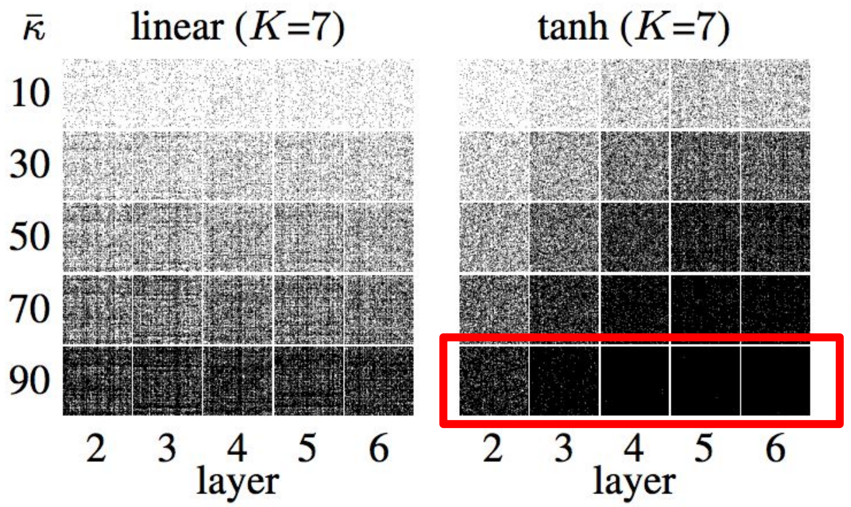
Effect of initialization on pruning: Results

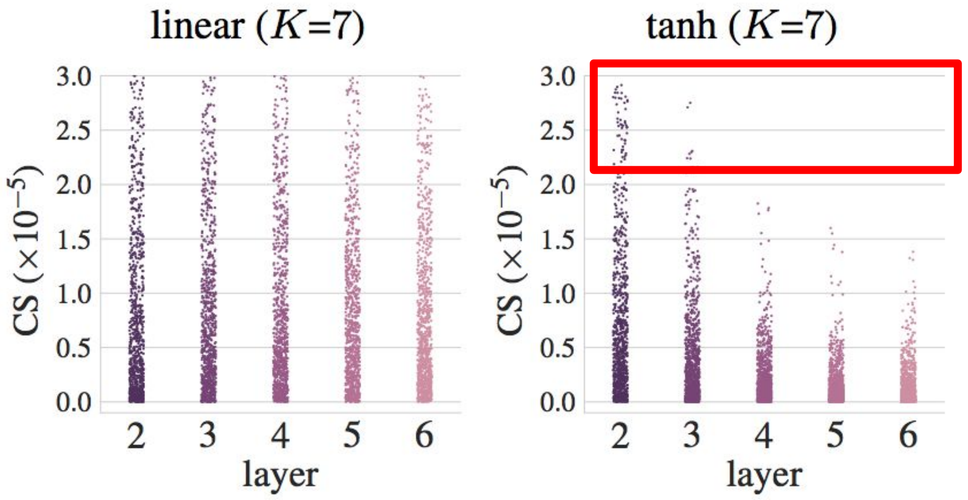




CS tends to decrease towards the later layers → choosing the top-κ globally

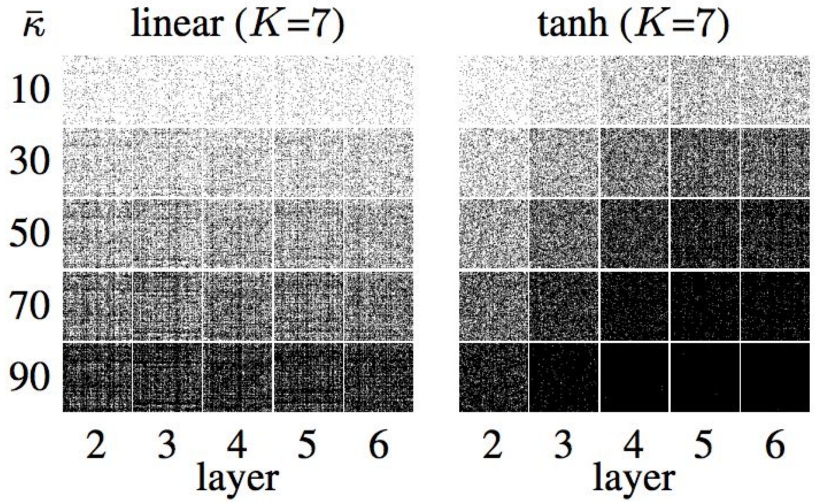
Effect of initialization on pruning: Results

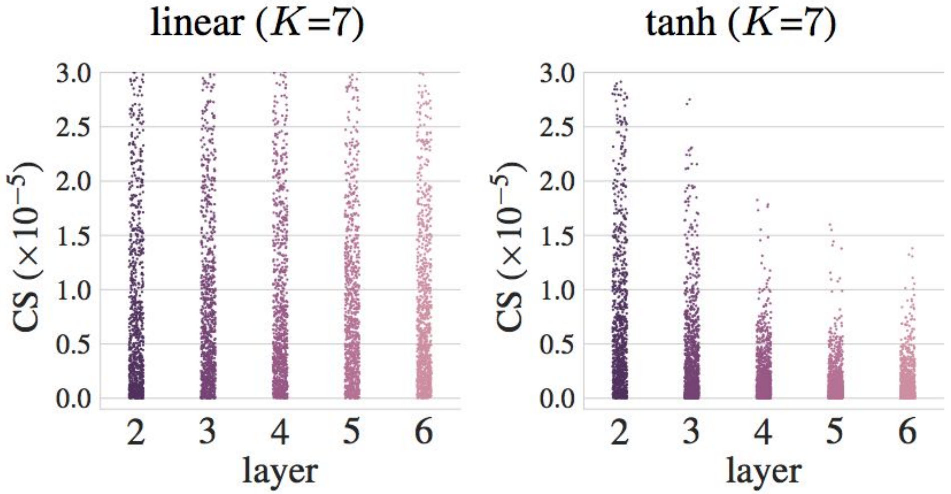




CS tends to decrease towards the later layers → choosing the top-κ globally → parameters distributed **non-uniformly** and **sparsely** towards the end of the network.

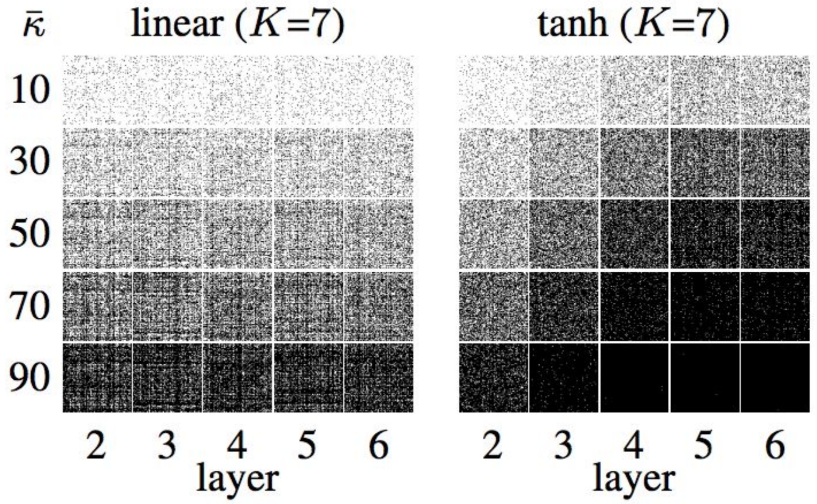
Effect of initialization on pruning: Results

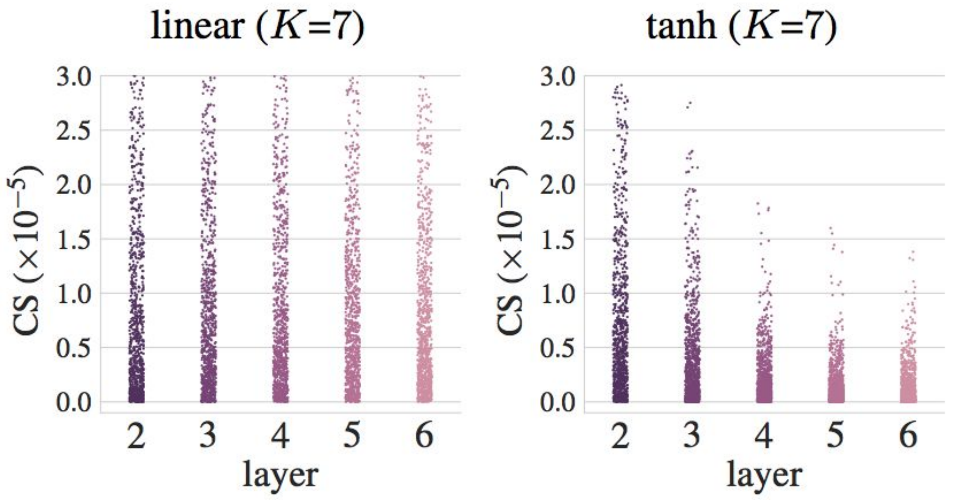




The initial weights have a crucial effect on CS and pruning results.

Effect of initialization on pruning: Results





The initial weights have a crucial effect on CS and pruning results. ***But why?***

Poor signal propagation → unreliable CS

We posit that the unreliability of CS is due to the poor signal propagation:
an initialization that projects the input signal to be strongly amplified or attenuated in the forward pass will saturate the error signal under backpropagation (i.e., gradients), and hence will result in poorly calibrated sensitivity scores across layers, which will eventually lead to poor pruning results.

Poor signal propagation → unreliable CS

We posit that the unreliability of CS is due to the poor signal propagation:
an initialization that projects the input signal to be strongly amplified or attenuated in the forward pass will saturate the error signal under backpropagation (i.e., gradients), and hence will result in poorly calibrated sensitivity scores across layers, which will eventually lead to poor pruning results.

Poor signal propagation → unreliable CS

We posit that the unreliability of CS is due to the poor signal propagation: an initialization that projects the input signal to be strongly amplified or attenuated in the forward pass will saturate the error signal under backpropagation (i.e., gradients), and hence will result in poorly calibrated sensitivity scores across layers, which will eventually lead to poor pruning results.

Poor signal propagation → unreliable CS

We posit that the unreliability of CS is due to the poor signal propagation:
an initialization that projects the input signal to be strongly amplified or attenuated in the forward pass will saturate the error signal under backpropagation (i.e., gradients), and hence will result in poorly calibrated sensitivity scores across layers, which will eventually lead to poor pruning results.

Gradient signal in connection sensitivity

Decompose the connection sensitivity metric:

$$\left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial \mathbf{c}}\right|_{\mathbf{c}=\mathbf{1}} = \left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial (\mathbf{c} \odot \mathbf{w})}\right|_{\mathbf{c}=\mathbf{1}} \odot \mathbf{w} = \frac{\partial L(\mathbf{w}; \mathcal{D})}{\partial \mathbf{w}} \odot \mathbf{w}\ .$$

Gradient signal in connection sensitivity

Decompose the connection sensitivity metric:

$$\left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial \mathbf{c}}\right|_{\mathbf{c}=\mathbf{1}} = \left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial (\mathbf{c} \odot \mathbf{w})}\right|_{\mathbf{c}=\mathbf{1}} \odot \mathbf{w} = \frac{\partial L(\mathbf{w}; \mathcal{D})}{\partial \mathbf{w}} \odot \mathbf{w} \,.$$

Considering ∂L/∂cj for a given j, since w_j does not depend on any other layers or signal propagation, the only term that depends on signal propagation in the network is the gradient term $\partial L/\partial w_j$.

Gradient signal in connection sensitivity

Decompose the connection sensitivity metric:

$$\left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial \mathbf{c}}\right|_{\mathbf{c}=\mathbf{1}} = \left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial (\mathbf{c} \odot \mathbf{w})}\right|_{\mathbf{c}=\mathbf{1}} \odot \mathbf{w} = \frac{\partial L(\mathbf{w}; \mathcal{D})}{\partial \mathbf{w}} \odot \mathbf{w} \,.$$

Considering ∂L/∂cj for a given j, since w_j does not depend on any other layers or signal propagation, the only term that depends on signal propagation in the network is the gradient term $\partial L/\partial w_j$.

Hence, a necessary condition to ensure reliable ∂L/∂c (and connection sensitivity) is that the gradients ∂L/∂w need to be faithful.

Layerwise dynamical isometry

Dynamical isometry & mean field theory

Dynamical isometry is a condition by having as many singular values (of the network’s input-output Jacobian) as possible near 1 (Saxe et al., 2014). Under this condition, error signals backpropagate faithfully and isometrically through the network, approximately preserving its norm and all angles between error vectors.

Dynamical isometry & mean field theory

Dynamical isometry is a condition by having as many singular values (of the network's input-output Jacobian) as possible near 1 (Saxe et al., 2014). Under this condition, error signals backpropagate faithfully and isometrically through the network, approximately preserving its norm and all angles between error vectors.

Dynamical isometry & mean field theory

Dynamical isometry is a condition by having as many singular values (of the network's input-output Jacobian) as possible near 1 (Saxe et al., 2014). Under this condition, error signals backpropagate faithfully and isometrically through the network, approximately preserving its norm and all angles between error vectors.

Mean field theory is used to develop a theoretical understanding of signal propagation in neural networks with random parameters (Poole et al., 2016). Precisely, the mean field approximation states that preactivations of wide, untrained neural networks can be captured as a Gaussian distribution;
E.g., a maximum depth through which signals can propagate at initialization; networks are trainable when signals can travel all the way through them (Schoenholz et al., 2017; Yang & Schoenholz, 2017; Xiao et al., 2018).

Dynamical isometry & mean field theory

Dynamical isometry is a condition by having as many singular values (of the network's input-output Jacobian) as possible near 1 (Saxe et al., 2014). Under this condition, error signals backpropagate faithfully and isometrically through the network, approximately preserving its norm and all angles between error vectors.

Mean field theory is used to develop a theoretical understanding of signal propagation in neural networks with random parameters (Poole et al., 2016). Precisely, the mean field approximation states that preactivations of wide, untrained neural networks can be captured as a Gaussian distribution;
E.g., a maximum depth through which signals can propagate at initialization; networks are trainable when signals can travel all the way through them (Schoenholz et al., 2017; Yang & Schoenholz, 2017; Xiao et al., 2018).

Dynamical isometry & mean field theory

Dynamical isometry is a condition by having as many singular values (of the network's input-output Jacobian) as possible near 1 (Saxe et al., 2014). Under this condition, error signals backpropagate faithfully and isometrically through the network, approximately preserving its norm and all angles between error vectors.

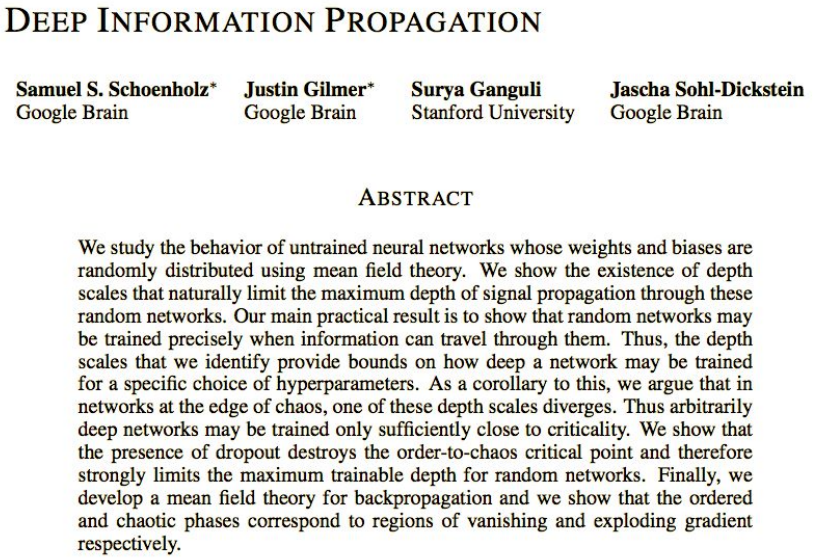
Mean field theory is used to develop a theoretical understanding of signal propagation in neural networks with random parameters (Poole et al., 2016). Precisely, the mean field approximation states that preactivations of wide, untrained neural networks can be captured as a Gaussian distribution;
E.g., a maximum depth through which signals can propagate at initialization; networks are trainable when signals can travel all the way through them (Schoenholz et al., 2017; Yang & Schoenholz, 2017; Xiao et al., 2018).

Dynamical isometry & mean field theory

Dynamical isometry is a condition by having as many singular values (of the network's input-output Jacobian) as possible near 1 (Saxe et al., 2014). Under this condition, error signals backpropagate faithfully and isometrically through the network, approximately preserving its norm and all angles between error vectors.

Mean field theory is used to develop a theoretical understanding of signal propagation in neural networks with random parameters (Poole et al., 2016). Precisely, the mean field approximation states that preactivations of wide, untrained neural networks can be captured as a Gaussian distribution;
E.g., a maximum depth through which signals can propagate at initialization; networks are trainable when signals can travel all the way through them (Schoenholz et al., 2017; Yang & Schoenholz, 2017; Xiao et al., 2018).

Depth scale ξ

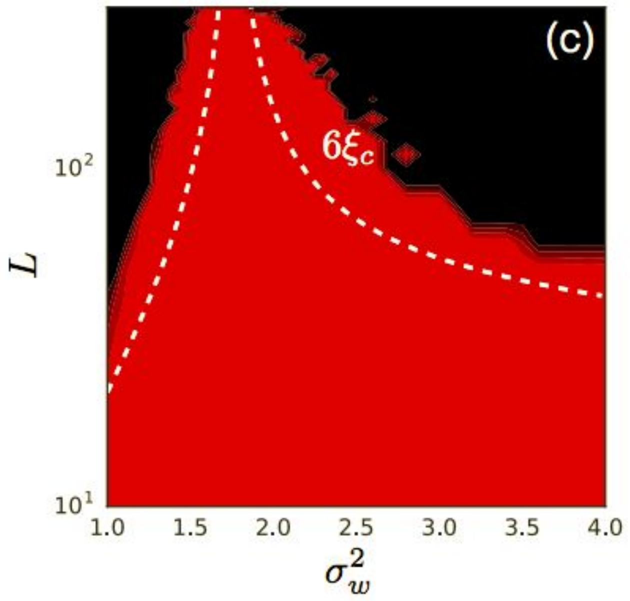


DEEP INFORMATION PROPAGATION

Samuel S. Schoenholz* Google Brain
Justin Gilmer* Google Brain
Surya Ganguli Stanford University
Jascha Sohl-Dickstein Google Brain

ABSTRACT

We study the behavior of untrained neural networks whose weights and biases are randomly distributed using mean field theory. We show the existence of depth scales that naturally limit the maximum depth of signal propagation through these random networks. Our main practical result is to show that random networks may be trained precisely when information can travel through them. Thus, the depth scales that we identify provide bounds on how deep a network may be trained for a specific choice of hyperparameters. As a corollary to this, we argue that in networks at the edge of chaos, one of these depth scales diverges. Thus arbitrarily deep networks may be trained only sufficiently close to criticality. We show that the presence of dropout destroys the order-to-chaos critical point and therefore strongly limits the maximum trainable depth for random networks. Finally, we develop a mean field theory for backpropagation and we show that the ordered and chaotic phases correspond to regions of vanishing and exploding gradient respectively.



Gradients in terms of Jacobians

From the feed-forward dynamics of a network in Equation 2, the network's input-output Jacobian corresponding to a given input $\mathbf{x}^0$ can be written, by the chain rule of differentiation, as:

$$\mathbf{J}^{0,K} = \frac{\partial \mathbf{x}^K}{\partial \mathbf{x}^0} = \prod_{l=1}^{K} \mathbf{D}^l \mathbf{W}^l \,, \tag{4}$$

where $\mathbf{D}^l \in \mathbb{R}^{N \times N}$ is a diagonal matrix with entries $\mathbf{D}^l_{ij} = \phi'(h^l_i)\delta_{ij}$, with ϕ' denoting the derivative of nonlinearity ϕ, and $\delta_{ij} = \mathbb{1}[i = j]$ is the Kronecker delta. Here, we will use $\mathbf{J}^{k,l}$ to denote the Jacobian from layer k to layer l. Now, we give the relationship between gradients and Jacobians:

Gradients in terms of Jacobians

Proposition 1. Let $\epsilon = \partial L / \partial \mathbf{x}^K$ denote the error signal and $\mathbf{x}^0$ denote the input signal. Then,

1. the gradients satisfy:

$$\mathbf{g}_{\mathbf{w}^l}^T = \epsilon \, \mathbf{J}^{l,K} \mathbf{D}^l \otimes \mathbf{x}^{l-1} \;, \tag{9}$$

   where $\mathbf{J}^{l,K} = \partial \mathbf{x}^K / \partial \mathbf{x}^l$ is the Jacobian from layer l to the output and $\otimes$ is the Kronecker product.

2. additionally, for linear networks, *i.e.*, when ϕ is the identity:

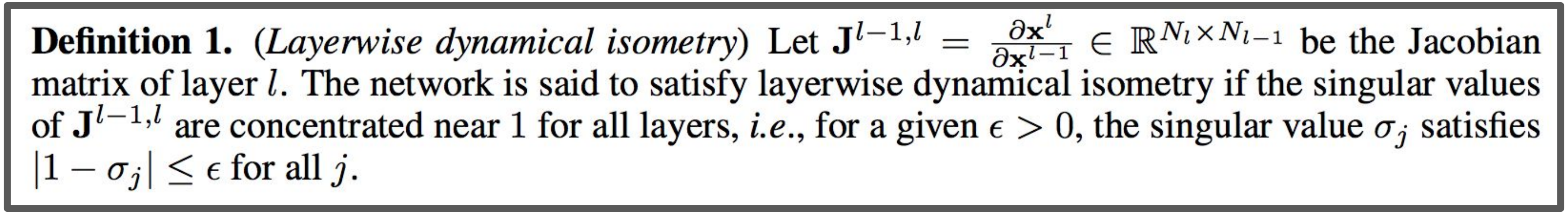
$$\mathbf{g}_{\mathbf{w}^l}^T = \epsilon \, \mathbf{J}^{l,K} \otimes \left(\mathbf{J}^{0,l-1} \mathbf{x}^0 + \mathbf{a} \right) \;, \tag{10}$$

   where $\mathbf{J}^{0,l-1} = \partial \mathbf{x}^{l-1} / \partial \mathbf{x}^0$ is the Jacobian from the input to layer $l-1$ and $\mathbf{a} \in \mathbb{R}^N$ is the constant term that does not depend on $\mathbf{x}^0$.

Layerwise dynamical isometry: Ensuring faithful gradients

Consider layerwise Jacobian: $$\mathbf{J}^{l-1,l} = \frac{\partial \mathbf{x}^l}{\partial \mathbf{x}^{l-1}} = \mathbf{D}^l \mathbf{W}^l \,.$$

Then, it is sufficient to have *layerwise dynamical isometry* in order to ensure faithful signal propagation in the network.



Definition 1. (*Layerwise dynamical isometry*) Let $\mathbf{J}^{l-1,l} = \frac{\partial \mathbf{x}^l}{\partial \mathbf{x}^{l-1}} \in \mathbb{R}^{N_l \times N_{l-1}}$ be the Jacobian matrix of layer l. The network is said to satisfy layerwise dynamical isometry if the singular values of $\mathbf{J}^{l-1,l}$ are concentrated near 1 for all layers, *i.e.*, for a given $\epsilon > 0$, the singular value σ_j satisfies $|1 - \sigma_j| \leq \epsilon$ for all j.

This would guarantee that the signal from layer l to l − 1 (or vice versa) is propagated without amplification or attenuation in any of its dimension.

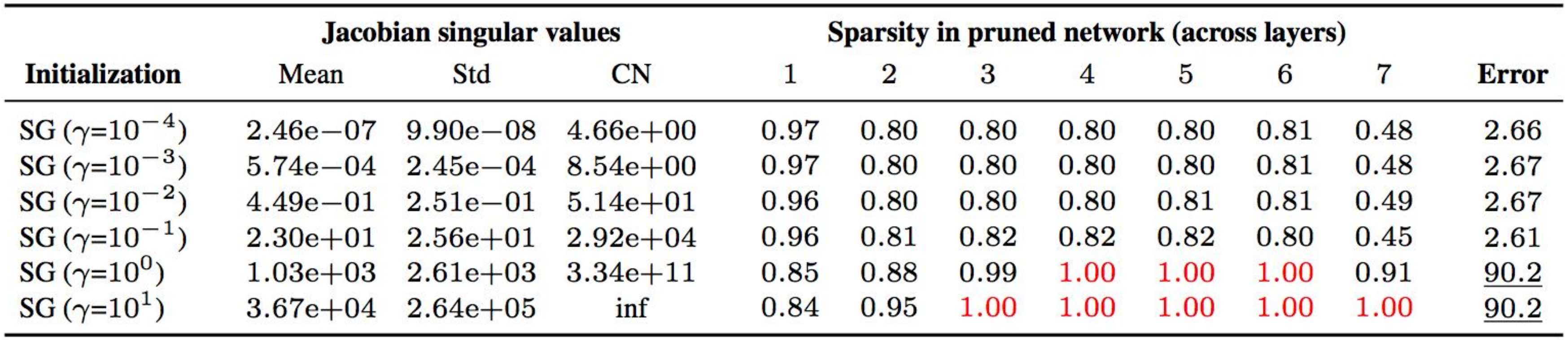
Layerwise dynamical isometry: Ensuring faithful gradients

Recall,
1. a network is pruned with a global threshold based on connection sensitivity.
2. the connection sensitivity is the gradients scaled by the weights.

$\Longleftrightarrow$ CS scores across layers need to be of the same scale
$\Longleftrightarrow$ Require the gradients to be faithful and the weights to be of the same scale for all layers.

This condition is trivially satisfied with the layerwise dynamical isometry, as each layer is initialized identically (i.e., orthogonal initialization).

Interpreting failure cases from signal propagation perspective



Initialization	Jacobian singular values Mean	Std	CN	Sparsity in pruned network (across layers) 1	2	3	4	5	6	7	Error
SG (γ=10^{-4})	2.46e−07	9.90e−08	4.66e+00	0.97	0.80	0.80	0.80	0.80	0.81	0.48	2.66
SG (γ=10^{-3})	5.74e−04	2.45e−04	8.54e+00	0.97	0.80	0.80	0.80	0.80	0.81	0.48	2.67
SG (γ=10^{-2})	4.49e−01	2.51e−01	5.14e+01	0.96	0.80	0.80	0.80	0.81	0.81	0.49	2.67
SG (γ=10^{-1})	2.30e+01	2.56e+01	2.92e+04	0.96	0.81	0.82	0.82	0.82	0.80	0.45	2.61
SG (γ=10^{0})	1.03e+03	2.61e+03	3.34e+11	0.85	0.88	0.99	1.00	1.00	1.00	0.91	90.2
SG (γ=10^{1})	3.67e+04	2.64e+05	inf	0.84	0.95	1.00	1.00	1.00	1.00	1.00	90.2

While CS pruning is robust to moderate changes in JSV, it fails catastrophically when the condition number is very large (> 1e+11).

Signal propagation on sparse networks and their training behaviors

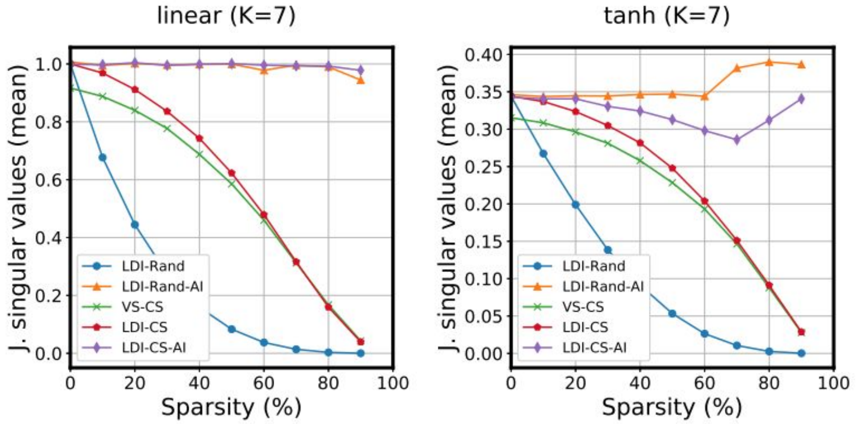
How do signals propagate in the sparse network?

Experiments:

Step 1: Initialize a network (VS or LDI).
Step 2: Prune for a sparsity level κ based on connection sensitivity.
Step 3: (optional) Enforce approximate dynamical isometry, if specified.
Step 4: Train the pruned sparse network using SGD.

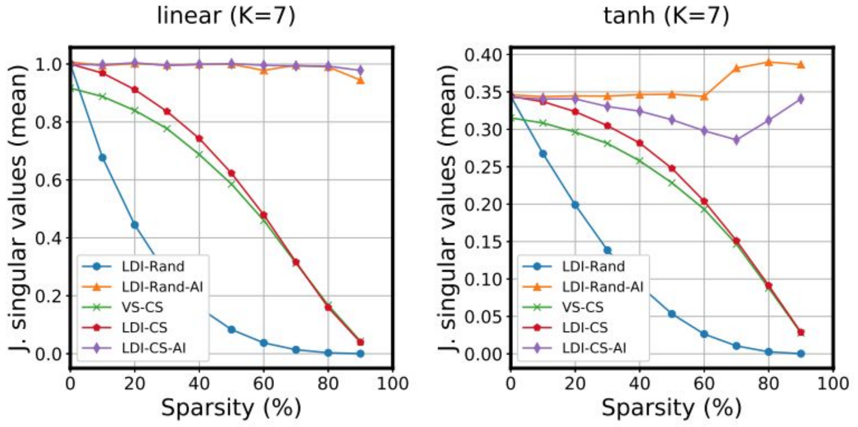
- We measure signal propagation on the sparse network right before *Step 4*, and observe training behavior during *Step 4*.
- {A}-{B}-{C} for {initialization}-{pruning}-{approximate isometry}.

Signal propagation and trainability

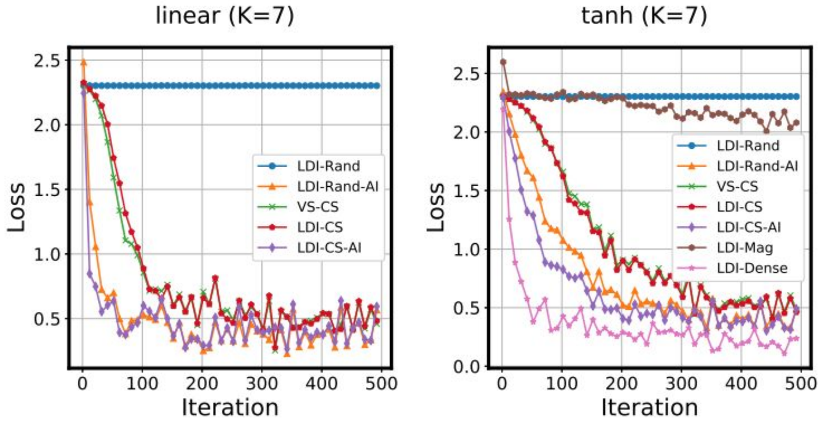


(a) Signal propagation

Signal propagation and trainability



(a) Signal propagation



(b) Training behavior

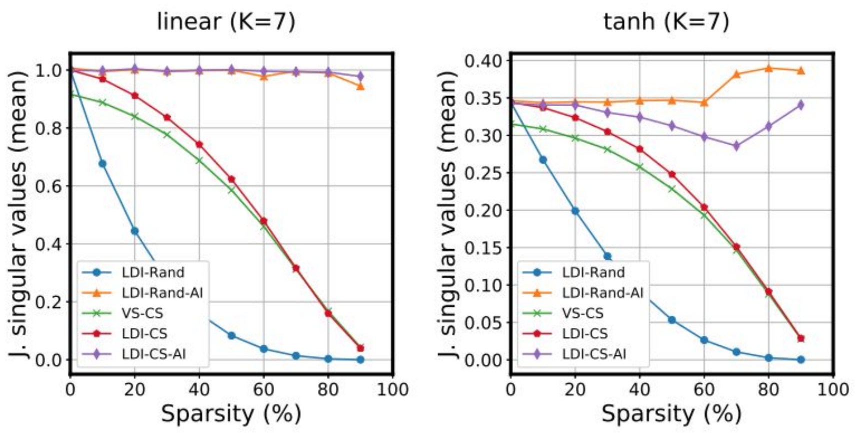
Approximate dynamical isometry

Q: What if we can *repair the broken isometry*, before training the pruned network, such that we can achieve trainability comparable to that of the dense network? Precisely, we consider the following:

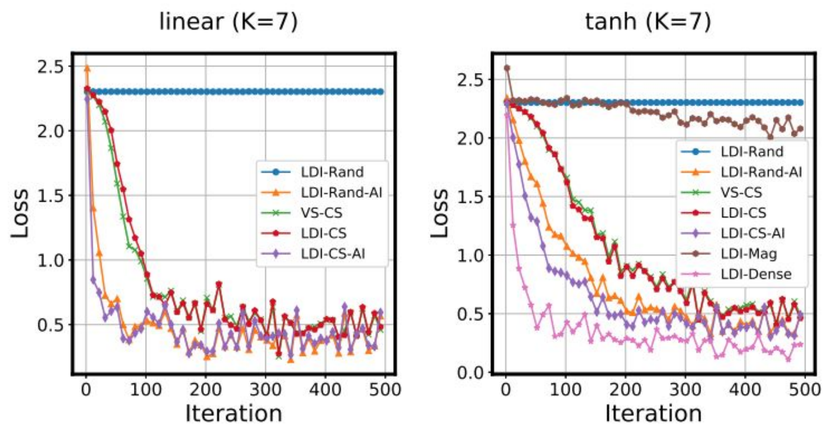
$$\min_{\mathbf{W}^l} \|(\mathbf{C}^l \odot \mathbf{W}^l)^T(\mathbf{C}^l \odot \mathbf{W}^l) - \mathbf{I}^l\|_F$$

Given the sparsity topology C and initial weights W, this data-free spectral method attempts to find an optimal W*, such that the combination of the sparse topology and the weights to be layerwise orthogonal, potentially to the full rank capacity.

Signal propagation and trainability



(a) Signal propagation

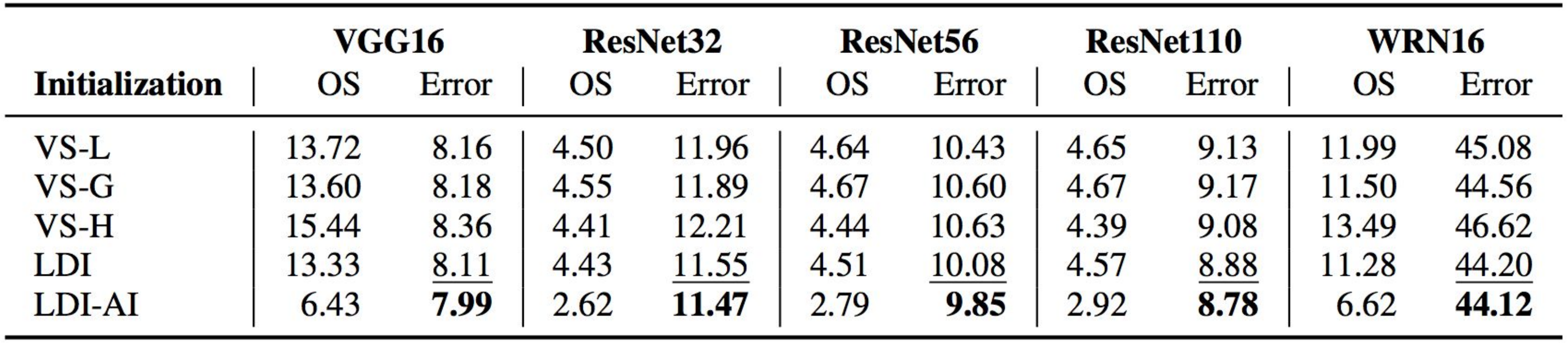


(b) Training behavior

Key points

- Pruning breaks dynamical isometry, and the more a network is pruned, the weaker signal propagation becomes on the pruned sparse network.
- The better a network propagates signals, the faster it converges during training.
- Enforcing approximate isometry recovers signal propagation on sparse networks, which in turn improves the training performance of sparse networks quite significantly.
- In addition to signal propagation, the structure (the choice of pruning method) and the number of parameters (sparsity level) also affect trainability of sparse neural networks.

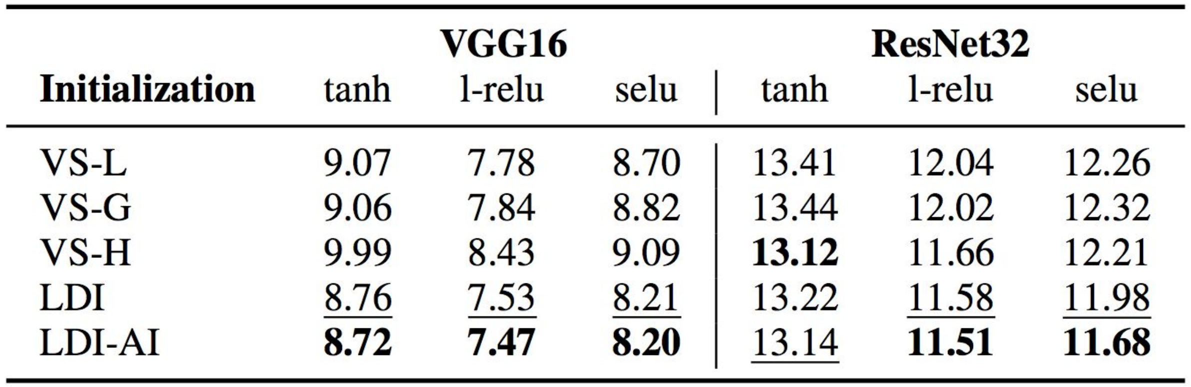
Validation and extensions: Modern networks



	VGG16		ResNet32		ResNet56		ResNet110		WRN16	
Initialization	OS	Error	OS	Error	OS	Error	OS	Error	OS	Error
VS-L	13.72	8.16	4.50	11.96	4.64	10.43	4.65	9.13	11.99	45.08
VS-G	13.60	8.18	4.55	11.89	4.67	10.60	4.67	9.17	11.50	44.56
VS-H	15.44	8.36	4.41	12.21	4.44	10.63	4.39	9.08	13.49	46.62
LDI	13.33	8.11	4.43	11.55	4.51	10.08	4.57	8.88	11.28	44.20
LDI-AI	6.43	**7.99**	2.62	**11.47**	2.79	**9.85**	2.92	**8.78**	6.62	**44.12**

- The **first** and second best results are highlighted in each column of errors.
- The orthogonal initialization with enforced approximate isometry method achieves the best results across all tested architectures.

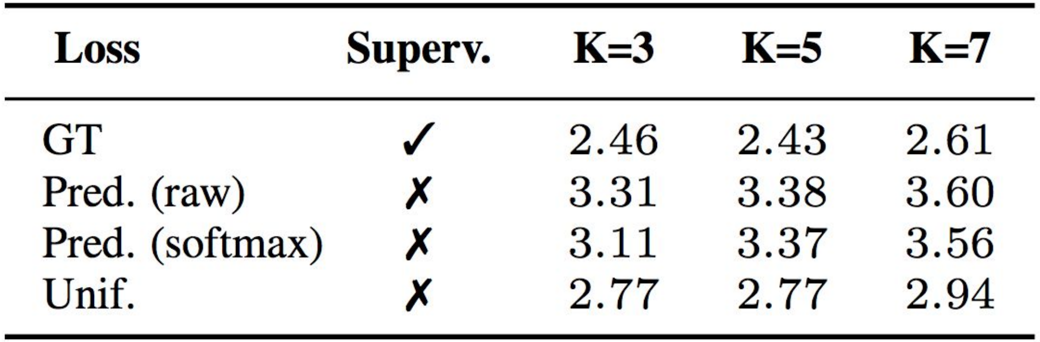
Validation and extensions: Nonlinearities



	VGG16			**ResNet32**		
Initialization	tanh	l-relu	selu	tanh	l-relu	selu
VS-L	9.07	7.78	8.70	13.41	12.04	12.26
VS-G	9.06	7.84	8.82	13.44	12.02	12.32
VS-H	9.99	8.43	9.09	**13.12**	11.66	12.21
LDI	8.76	7.53	8.21	13.22	11.58	11.98
LDI-AI	**8.72**	**7.47**	**8.20**	13.14	**11.51**	**11.68**

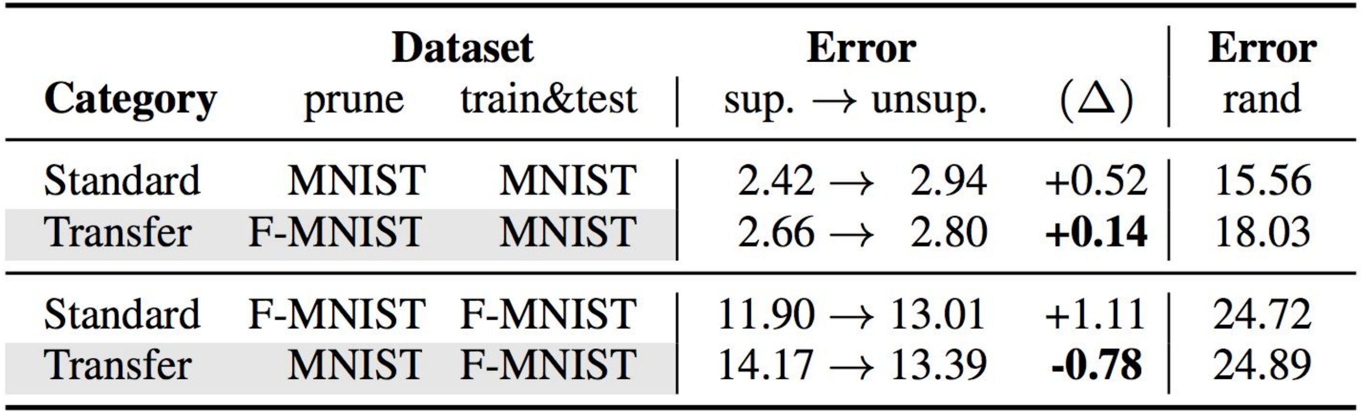
- The **first** and second best results are highlighted in each column of errors.
- The orthogonal initialization consistently outperforms variance scaling methods across different nonlinear activation functions.

Validation and extensions: Unsupervised pruning



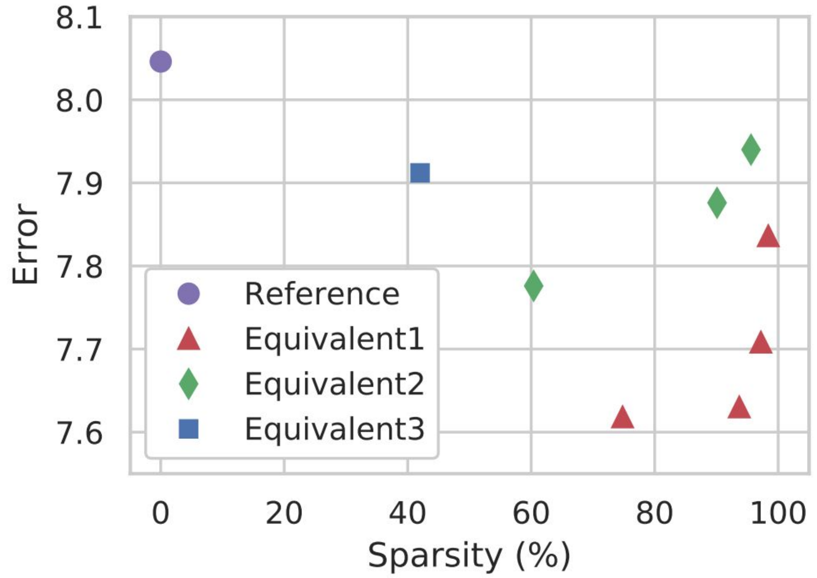
Loss	Superv.	K=3	K=5	K=7
GT	✓	2.46	2.43	2.61
Pred. (raw)	✗	3.31	3.38	3.60
Pred. (softmax)	✗	3.11	3.37	3.56
Unif.	✗	2.77	2.77	2.94

Validation and extensions: Transfer of sparsity



	Dataset		Error		Error
Category	prune	train&test	sup. $\to$ unsup.	(Δ)	rand
Standard	MNIST	MNIST	2.42 $\to$ 2.94	+0.52	15.56
Transfer	F-MNIST	MNIST	2.66 $\to$ 2.80	**+0.14**	18.03
Standard	F-MNIST	F-MNIST	11.90 $\to$ 13.01	+1.11	24.72
Transfer	MNIST	F-MNIST	14.17 $\to$ 13.39	**-0.78**	24.89

Validation and extensions: Architecture sculpting



Summary

- (Observation) The initial weights have a critical impact on connection sensitivity, and hence pruning results.
- (Layerwise dynamical isometry) A formal characterization of a sufficient condition to ensure faithful signal propagation in a network.
- (Signal propagation and trainability of sparse networks) pruning breaks dynamical isometry and degrades trainability; a simple method to recover signal propagation and enhance trainability of the compressed network.
- (Validation and extensions) A range of experiments to verify that the effectiveness of signal propagation perspective for pruning at initialization.

Towards “winning lottery ticket”

The results on the increased trainability of compressed neural networks can take us one step towards finding “winning lottery ticket” (i.e., a set of initial weights that given a sparse topology can quickly reach to a generalization performance that is comparable to the uncompressed network, once trained) suggested in Frankle & Carbin (2019).

Limitations & future work

- While we produce (quickly) trainable sparse networks, the two-stage orthogonalization process can be suboptimal, especially at high sparsity.

- Weights change during training affecting signal propagation, and hence, dynamical isometry may not continue to hold over the course of training.

- A potential key to network compression is to address the complex interplay between optimization and signal propagation, and it might be immensely beneficial if an optimization naturally takes place in the space of isometry.