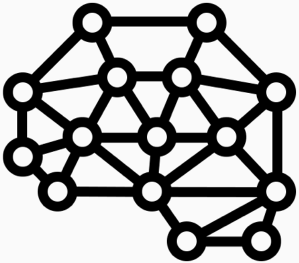
Toward efficient deep learning with sparse neural networks

Namhoon Lee

UNIST

Problem: Neural networks are too large

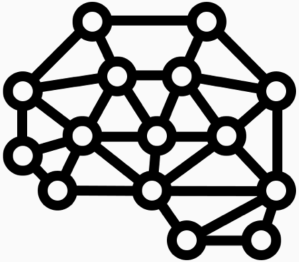
Artificial neural network



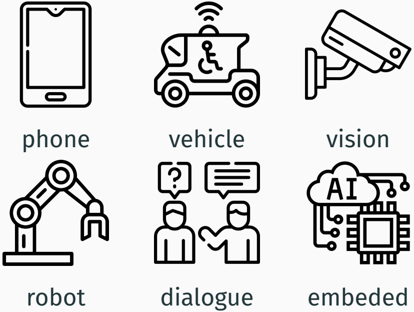
- Number of parameters $>$ M, B, T
- Memory, computation, energy

Problem: Neural networks are too large

Artificial neural network

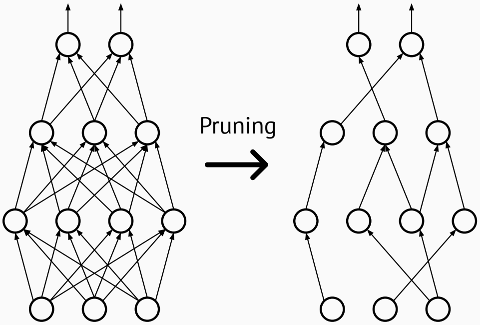


- Number of parameters > M, B, T
- Memory, computation, energy



Resource constrained environments

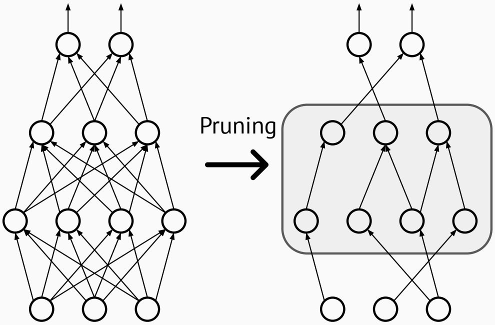
Sparse neural networks



Dense neural network

Sparse neural network

Sparse neural networks



Dense neural network

Sparse neural network

$$\begin{bmatrix} w_{11} & 0 & 0 \\ 0 & w_{22} & 0 \\ 0 & w_{32} & w_{33} \\ 0 & 0 & w_{43} \end{bmatrix}$$

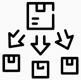
Sparse parameterization

Computations associated with zero values can be skipped!

The focus of this presentation

How to ***find*** a sparse neural network

How to ***initialize*** a sparse neural network



How to ***parallelize*** a sparse neural network training

Outline

"SNIP: Single-shot network pruning based on connection sensitivity" by Lee, Ajanthan, Torr (ICLR 2019)

"A signal propagation perspective for pruning neural networks at initialization" by Lee, Ajanthan, Gould, Torr (ICLR 2020)

"Understanding the effects of data parallelism on neural network training" by Lee, Ajanthan, Torr, Jaggi (ICLR 2021)

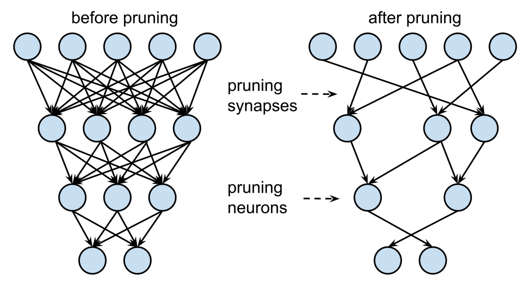
SNIP: Single-shot network pruning based on connection sensitivity

Namhoon Lee[1] Thalaiyasingam Ajanthan[1] Philip H. S. Torr[1]

ICLR 2019

[1]University of Oxford

Neural network pruning



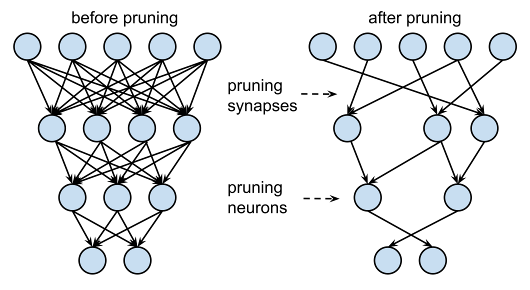
Pruning a densely connected network

Network pruning has a rich history.

There exists various approaches.

- Elements (parameter, activation)
- Metrics (magnitude, derivative)
- Removal (individually, structured)

Neural network pruning



Pruning a densely connected network

Network pruning has a rich history.

There exists various approaches.

- Elements (parameter, activation)
- Metrics (magnitude, derivative)
- Removal (individually, structured)

It can remove many parameters ($> 90\%$).

Drawbacks in existing methods

Many pruning algorithms involve

- Hyperparameters with heuristics
- Architectural dependency
- Optimization difficulty
- Iterative process
- Pretraining

Drawbacks in existing methods

Many pruning algorithms involve

- Hyperparameters with heuristics
- Architectural dependency
- Optimization difficulty
- Iterative process
- Pretraining

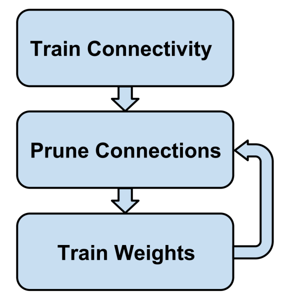
$\Rightarrow$ Complex, non-scalable, expensive

Drawbacks in existing methods

Many pruning algorithms involve

- Hyperparameters with heuristics
- Architectural dependency
- Optimization difficulty
- Iterative process
- Pretraining

$\Rightarrow$ Complex, non-scalable, expensive



A typical pruning algorithm
(Han et al. 2016; Frankle and Carbin 2019)

Desired characteristics

Ideally, we want . .

Desired characteristics

Ideally, we want . .

- No hyperparameters
- No architectural dependency
- No iterative prune–train cycle
- No pretraining
- No large data

Desired characteristics

Ideally, we want . . .

- No hyperparameters
- No architectural dependency
- No iterative prune–train cycle
- No pretraining
- No large data

Single-shot pruning prior to training

Problem formulation

Pruning as constrained optimization:

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

Problem formulation

Pruning as constrained optimization:

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

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \|\mathbf{w}\|_0 \leq \kappa \ .$$

$\Rightarrow$ Difficult to solve

Problem formulation

Pruning as constrained optimization:

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

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \|\mathbf{w}\|_0 \leq \kappa \ .$$

$\Rightarrow$ Difficult to solve

Pruning as identification:

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

i.e., effect of removing parameter j as a saliency measure.

Problem formulation

Pruning as constrained optimization:

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

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \|\mathbf{w}\|_0 \leq \kappa .$$

$\Rightarrow$ Difficult to solve

Pruning as identification:

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

i.e., effect of removing parameter j as a saliency measure.

$\Rightarrow$ Expensive to measure

SNIP

Re-write the objective with auxiliary indicator variable $\mathbf{c}$:

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

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m \ , \quad \mathbf{c} \in \{0, 1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa \ .$$

SNIP

Re-write the objective with auxiliary indicator variable **c** :

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

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m \ , \quad \mathbf{c} \in \{0, 1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa \ .$$

Approximate the effect of removing j :

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

i.e. $\partial L / \partial c_j$ is an infinitesimal version of ΔL_j (Koh and Liang 2017).

SNIP

Define connection sensitivity:

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

Characteristics:

- Alleviate the dependency on weights
- One forward-backward pass for all j at once

SNIP

Algorithm:

SNIP

Algorithm:

1. Initialize the network parameters $\mathbf{w}_0$
2. Sample a mini-batch $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{b} \sim \mathcal{D}$

SNIP

Algorithm:

1. Initialize the network parameters $\mathbf{w}_0$
2. Sample a mini-batch $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{b} \sim \mathcal{D}$
3. Compute the connection sensitivity $s_j \ \forall j \in \{1, .., m\}$
4. Keep top-κ and prune the rest

SNIP

Algorithm:

1. Initialize the network parameters $\mathbf{w}_0$
2. Sample a mini-batch $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{b} \sim \mathcal{D}$
3. Compute the connection sensitivity $s_j \; \forall j \in \{1, .., m\}$
4. Keep top-κ and prune the rest
5. Train the pruned network in the standard way

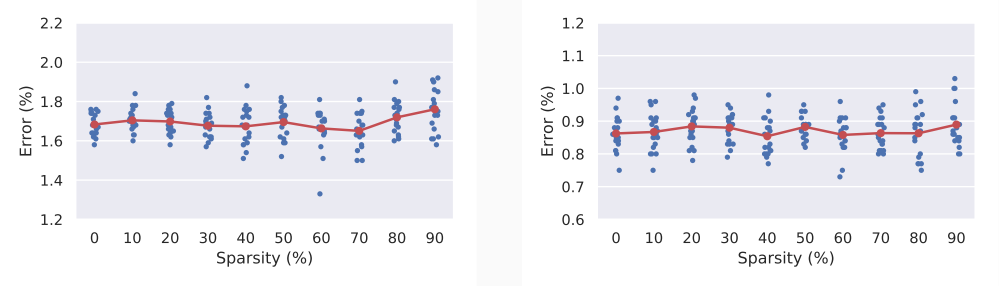
SNIP

Algorithm:

1. Initialize the network parameters $\mathbf{w}_0$
2. Sample a mini-batch $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{b} \sim \mathcal{D}$
3. Compute the connection sensitivity $s_j \; \forall j \in \{1, .., m\}$
4. Keep top-κ and prune the rest
5. Train the pruned network in the standard way

Single-shot **N**etwork at **I**nitialization **P**runing

Results on LeNets



(a) LeNet-300-100 **(b)** LeNet-5

SNIP can prune for a range of sparsity levels without losing much accuracy.

Comparing to state-of-the-arts

Method	Criterion	LeNet-300-100 $\bar{\kappa}$ (%)	LeNet-300-100 err. (%)	LeNet-5-Caffe $\bar{\kappa}$ (%)	LeNet-5-Caffe err. (%)	Pretrain	# Prune	Additional hyperparam.	Augment objective	Arch. constraints
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 98.0	**1.6** 2.4	98.0 99.0	**0.8** 1.1	✗	**1**	✗	✗	✗

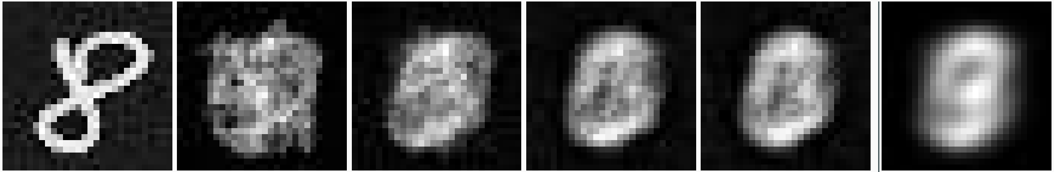
SNIP is capable of pruning for extreme sparsity levels (*e.g.*, 99% for LeNet-5), while being much simpler than other alternatives.

Applying to various architectures

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**

SNIP can be applied to various architectures.

Visualizing sparsity patterns

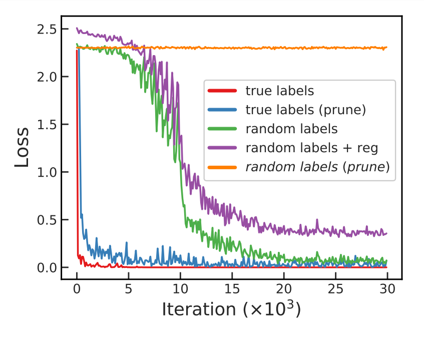


$b = 1$	$b = 10$	$b = 100$	$b = 1000$	$b = 10000$	train set
(1.94%)	(1.72%)	(1.64%)	(1.56%)	(1.40%)	–

Visualizing $\boldsymbol{c}^{(1)}$ of LeNet-300-100 reveals:

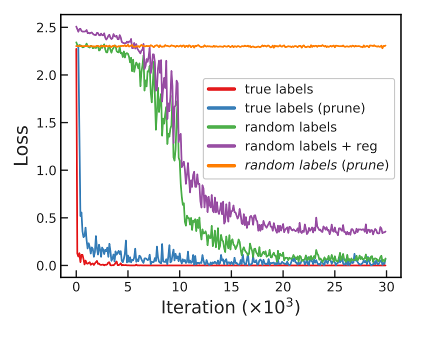
- When $b = 1$, SNIP retains connections relevant to perform classification.
- As b increases, remaining connections get close to the average of train set.

Preventing memorization



"Fitting random labels" (Zhang et al. 2017)

Preventing memorization



“Fitting random labels” (Zhang et al. 2017)

The pruned network performs the task well without fitting the random labels.

SNIP-ing can prevent memorization.

A signal propagation perspective for pruning neural networks at initialization

Namhoon Lee[1] Thalaiyasingam Ajanthan[2] Stephen Gould[2] Philip H. S. Torr[1]

ICLR 2020 – spotlight

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

Motivation

Pruning can be done at initialization.

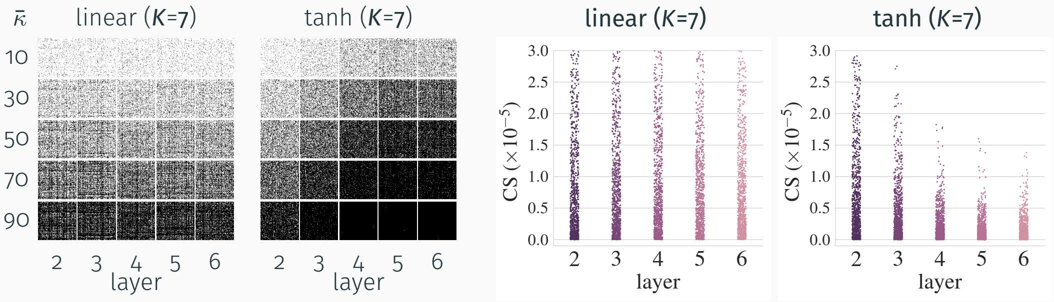
It remains unclear why pruning a randomly initialized neural network can be effective.

We begin by analyzing the effect of initial random weights ($\mathbf{w}_0$) on pruning.

The distribution of each node to be of same variance (LeCun et al. 1998):

$$\mathbf{w}_0^l \sim \mathcal{U}\left[-\sqrt{\frac{3}{n^l}}, \sqrt{\frac{3}{n^l}}\right].$$

Effect of initialization on pruning



CS scores saturate when initialized poorly, leading to a sub-network whose parameters are distributed sparsely toward the end.

- $s_j = \mathrm{norm}(|g_j|) = f(\mathbf{w}; \mathcal{D})$, where $g_j = (\partial L / \partial \mathbf{w}) \odot \mathbf{w}$.
- Necessary to ensure reliable gradient!

Layerwise dynamical isometry for faithful gradients

Gradients in terms of Jacobians

For a feed-forward network, the gradients satisfy:

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

where $\epsilon = \partial L / \partial \mathbf{x}^K$ denote the error signal, $\mathbf{J}^{l,K} = \partial \mathbf{x}^K / \partial \mathbf{x}^l$ is the Jacobian from layer l to the output layer K, $\mathbf{D}^l \in \mathbb{R}^{N \times N}$ refers to the derivative of nonlinearity, and $\otimes$ is the Kronecker product.

Layerwise dynamical isometry for faithful gradients

Gradients in terms of Jacobians

For a feed-forward network, the gradients satisfy:

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

where $\epsilon = \partial L / \partial \mathbf{x}^K$ denote the error signal, $\mathbf{J}^{l,K} = \partial \mathbf{x}^K / \partial \mathbf{x}^l$ is the Jacobian from layer l to the output layer K, $\mathbf{D}^l \in \mathbb{R}^{N \times N}$ refers to the derivative of nonlinearity, and $\otimes$ is the Kronecker product.

When pre-activations fall in the linear region of activation (LeCun et al. 1998; Glorot and Bengio 2010), graidents are solely characterized by Jacobian matrices.

Layerwise dynamical isometry for faithful gradients

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 satisfies *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.

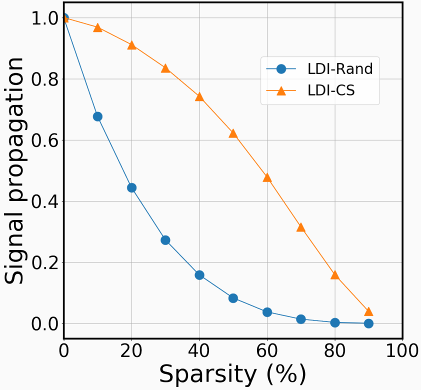
Layerwise dynamical isometry for faithful gradients

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 satisfies *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.

- Assuming mean-field approximation of pre-activations (Poole et al. 2016)
- Stronger condition than dynamical isometry (Saxe et al. 2014)

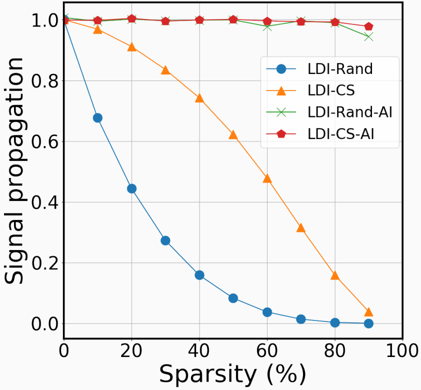
Sparsity degrades signal propagation



Jacobian singular values (JSV) decrease as per increasing sparsity.

⇒ Sparsity degrades signal propagation.

Enforcing approximate dynamical isometry

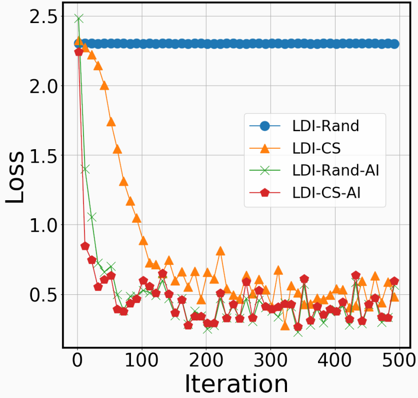


Enforce approximate isometry:

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

$\Rightarrow$ Restore signal propagation!

Enforcing approximate dynamical isometry



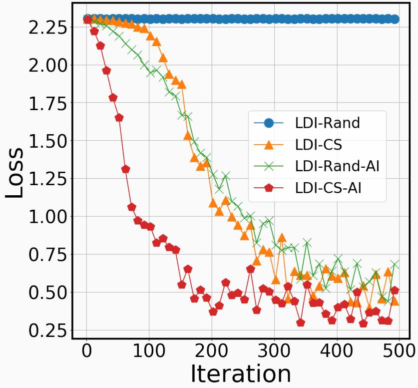
Enforce approximate isometry:

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

$\Rightarrow$ Restore signal propagation!

$\Rightarrow$ Improve training performance!

Enforcing approximate dynamical isometry



Enforce approximate isometry:

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

$\Rightarrow$ Restore signal propagation!

$\Rightarrow$ Improve training performance!

Validations and extensions

Experiments:

- Modern neural networks
- Non-linearity functions
- Pruning without supervision
- Transfer of sparsity
- Architecture sculpting

Please check the paper for more.

Summary

Observations:

- The initial random weights have critical impact on pruning.
- Sparsity breaks dynamical isometry and degrades signal propagation.

Suggestion:

Approximate isometry to secure signal propagation and enhance training!

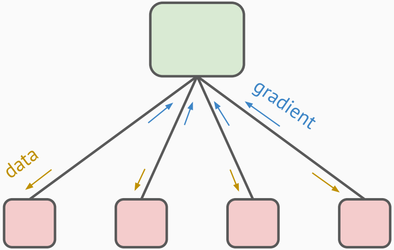
Understanding the effects of data parallelism and sparsity on neural network training

Namhoon Lee[1] Thalaiyasingam Ajanthan[2] Philip H. S. Torr[1] Martin Jaggi[3]

ICLR 2021

[1]University of Oxford [2]Australian National University [3]EPFL

Data parallelism

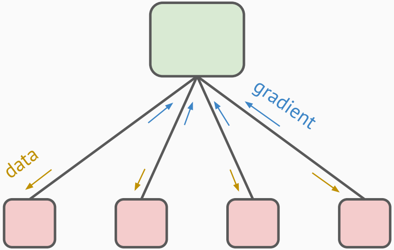


A parallel computing system

Processing training data in parallel

Accelerate training and model-agnostic

Data parallelism



A parallel computing system

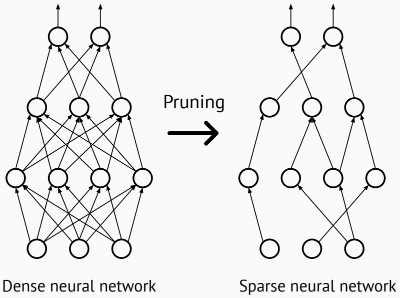
Processing training data in parallel

Accelerate training and model-agnostic

Degree of parallelism $\equiv$ Batch size (single node)

Active research for the effect of batch size (Dean et al. 2012; Goyal et al. 2017; Hoffer et al. 2017; Shallue et al. 2019; Lin et al. 2020)

Sparsity

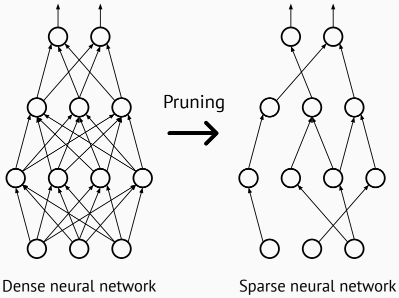


Introducing sparsity by pruning

Sparse neural networks

Save computations and memory

Sparsity



Introducing sparsity by pruning

Sparse neural networks

Save computations and memory

Pruning at initialization prior to training (Lee et al. 2019; Wang et al. 2020)

Subsequent training remains unknown.

Proposal

Data parallelism & Sparsity

- Efficient deep learning
- Complimentary benefits

Proposal

Data parallelism & Sparsity

- Efficient deep learning
- Complimentary benefits

What we do:

1. Measure their effects on training time
2. Develop theoretical analysis to explain the effects

Setup

For a given workload

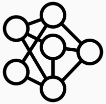
Network Data set Algorithm

Train for batch sizes and sparsity levels

Measure steps-to-result ($K^{\star}$)

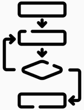
Setup

For a given workload



Network

Data set



Algorithm

Train for batch sizes and sparsity levels

Measure steps-to-result ($K^\star$)

Metaparameter search

- Parameters set before training (*e.g.* learning rate)
- To avoid any assumption on optimal metaparameters
- Search space: preliminary results
- Budget: 100 training trials

Setup

For a given workload

Network Data set Algorithm

Train for batch sizes and sparsity levels

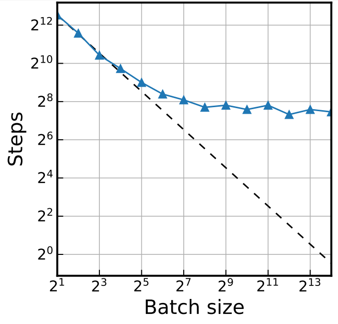
Measure steps-to-result ($K^\star$)

Metaparameter search

- Parameters set before training (*e.g.* learning rate)
- To avoid any assumption on optimal metaparameters
- Search space: preliminary results
- Budget: 100 training trials

Steps-to-result ($K^\star$) vs. Batch size (B)

Measuring the effects

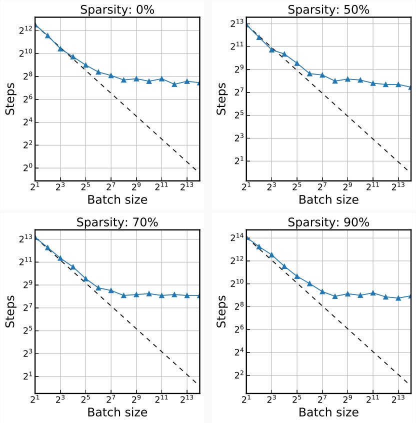


General scaling trend
($K^\star$ vs. B)

General scaling trend across various workloads

- Linear scaling
- Diminishing returns
- Maximal data parallelism

Measuring the effects



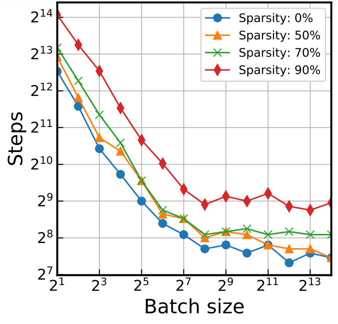
Various sparsity levels

General scaling trend across various workloads

- Linear scaling
- Diminishing returns
- Maximal data parallelism

Sparsity levels (0 – 90%)

Measuring the effects



All sparsity levels

General scaling trend across various workloads

- Linear scaling
- Diminishing returns
- Maximal data parallelism

Sparsity levels ($0 - 90\%$)

Difficulty of training under sparsity

Understanding the effects

Based on convergence properties of stochastic gradient methods:

The relationship between steps-to-result ($K^\star$) and batch size (B)

$$K^\star \approx \frac{c_1}{B} + c_2\,, \qquad \text{where } \; c_1 = \frac{\Delta L \beta}{\mu^2 \varepsilon^2} \; \text{ and } \; c_2 = \frac{\Delta}{\bar{\eta}^\star \mu \varepsilon}\,.$$

Understanding the effects

Based on convergence properties of stochastic gradient methods:

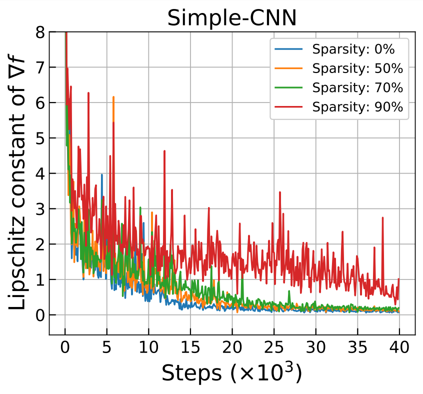
The relationship between steps-to-result ($K^\star$) and batch size (B)

$$K^\star \approx \frac{c_1}{B} + c_2\,, \qquad \text{where } c_1 = \frac{\Delta L \beta}{\mu^2 \varepsilon^2} \text{ and } c_2 = \frac{\Delta}{\bar{\eta}^\star \mu \varepsilon}\,.$$

This result precisely illustrates the observed scaling trends.

1. Linear scaling, diminishing returns, maximal data parallelism
2. Lipschitz smoothness (L) is what can shift the curve vertically

Lipschitz smoothness under sparsity



Local L throughout training

Local Lipschitz smoothness ($\boldsymbol{L}$)

The higher sparsity, the higher $\boldsymbol{L}$

Gradient changes relatively too quickly

The difficulty of training sparse networks

Summary

Main points:

1. General scaling trend for the effects of data parallelism and sparsity
2. Theoretical analysis to verify the effects
3. Lipschitz smoothness to explain the difficulty of training sparse networks

Code: https://github.com/namhoonlee/effect-dps-public

Contact: namhoon@robots.ox.ac.uk

References i

- Dean, Jeffrey et al. (2012). "Large scale distributed deep networks". In: *NeurIPS*.
- Frankle, Jonathan and Michael Carbin (2019). "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *ICLR*.
- Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *AISTATS*.
- Goyal, Priya et al. (2017). "Accurate, large minibatch sgd: Training imagenet in 1 hour". In: *arXiv preprint arXiv:1706.02677*.
- Han, Song, Huizi Mao, and William J Dally (2016). "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *ICLR*.
- Hoffer, Elad, Itay Hubara, and Daniel Soudry (2017). "Train longer, generalize better: closing the generalization gap in large batch training of neural networks". In: *NeurIPS*.

References ii

- Koh, Pang Wei and Percy Liang (2017). "Understanding black-box predictions via influence functions". In: *ICML*.
- LeCun, Yann A et al. (1998). "Efficient backprop". In: *Neural networks: Tricks of the trade*.
- Lee, Namhoon, Thalaiyasingam Ajanthan, and Philip HS Torr (2019). "SNIP: Single-shot network pruning based on connection sensitivity". In: *ICLR*.
- Lin, Tao et al. (2020). "Don't Use Large Mini-Batches, Use Local SGD". In: *ICLR*.
- Poole, Ben et al. (2016). "Exponential expressivity in deep neural networks through transient chaos". In: *NeurIPS*.
- Saxe, Andrew M, James L McClelland, and Surya Ganguli (2014). "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *ICLR*.
- Shallue, Christopher J et al. (2019). "Measuring the effects of data parallelism on neural network training". In: *JMLR*.

References iii

Wang, Chaoqi, Guodong Zhang, and Roger Grosse (2020). "Picking Winning Tickets Before Training by Preserving Gradient Flow". In: *ICLR*.

Zhang, Chiyuan et al. (2017). "Understanding deep learning requires rethinking generalization". In: *ICLR*.