Efficient & Scalable Deep Learning and Beyond

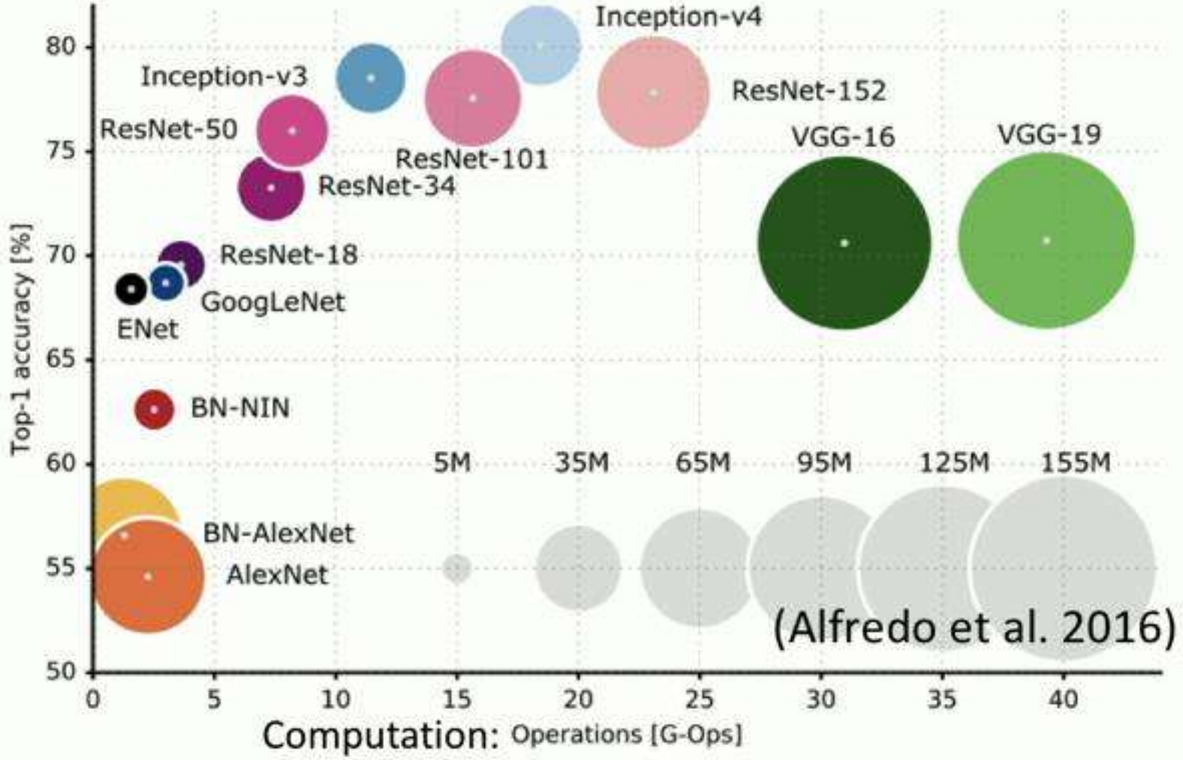
Wei Wen

Duke University

http://www.pittnuts.com/

Deep Learning Model Scaling

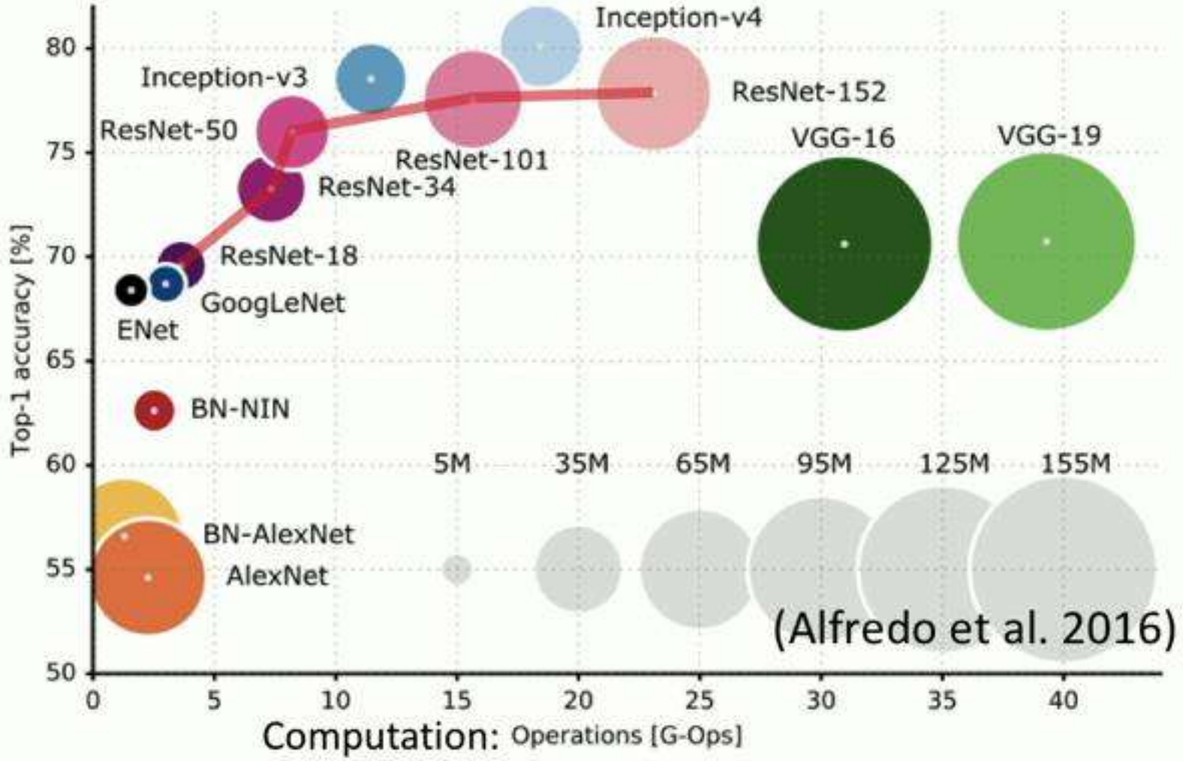
Image Classification on ImageNet



Trend: we can obtain better prediction using larger models

Deep Learning Model Scaling

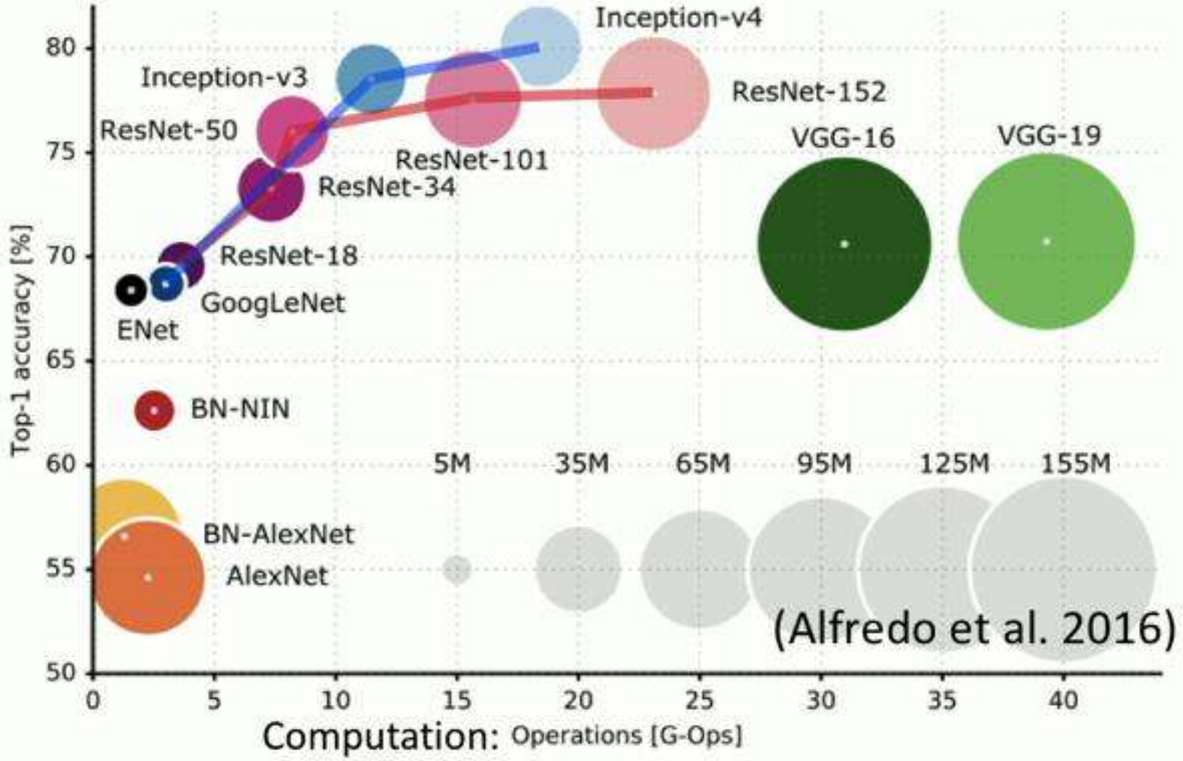
Image Classification on ImageNet



Trend: we can obtain better prediction using larger models

Deep Learning Model Scaling

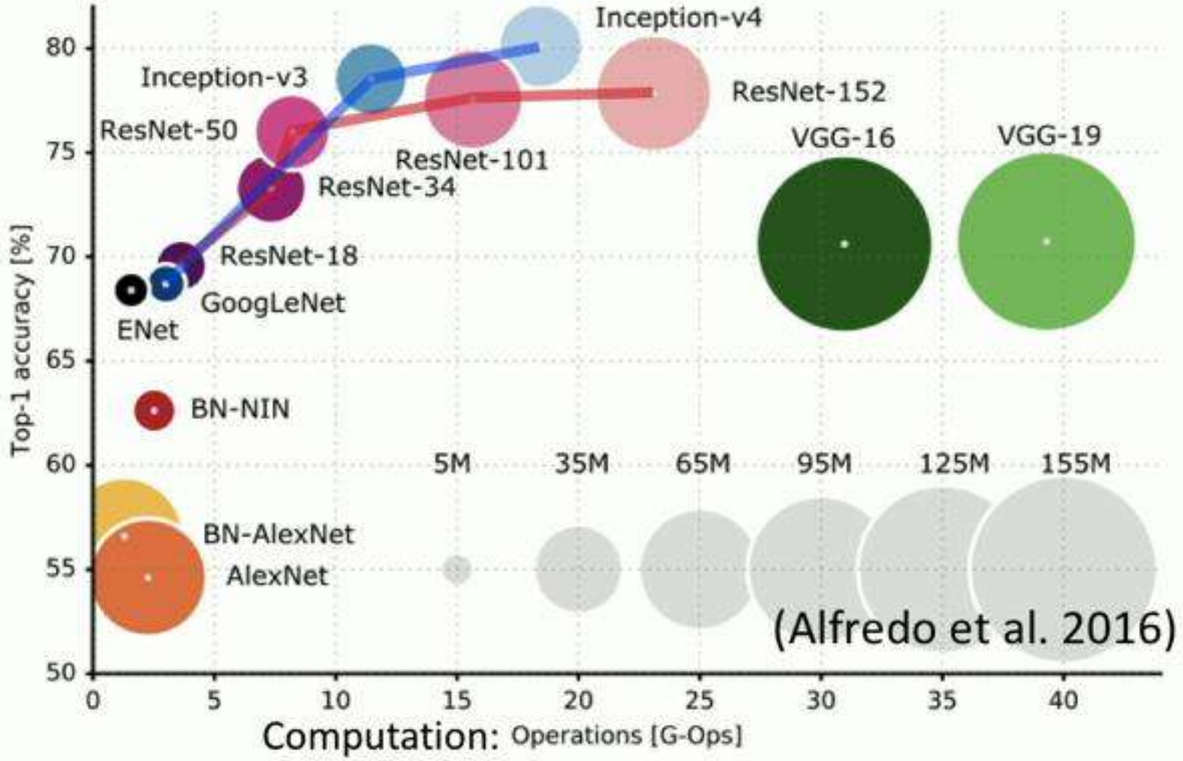
Image Classification on ImageNet



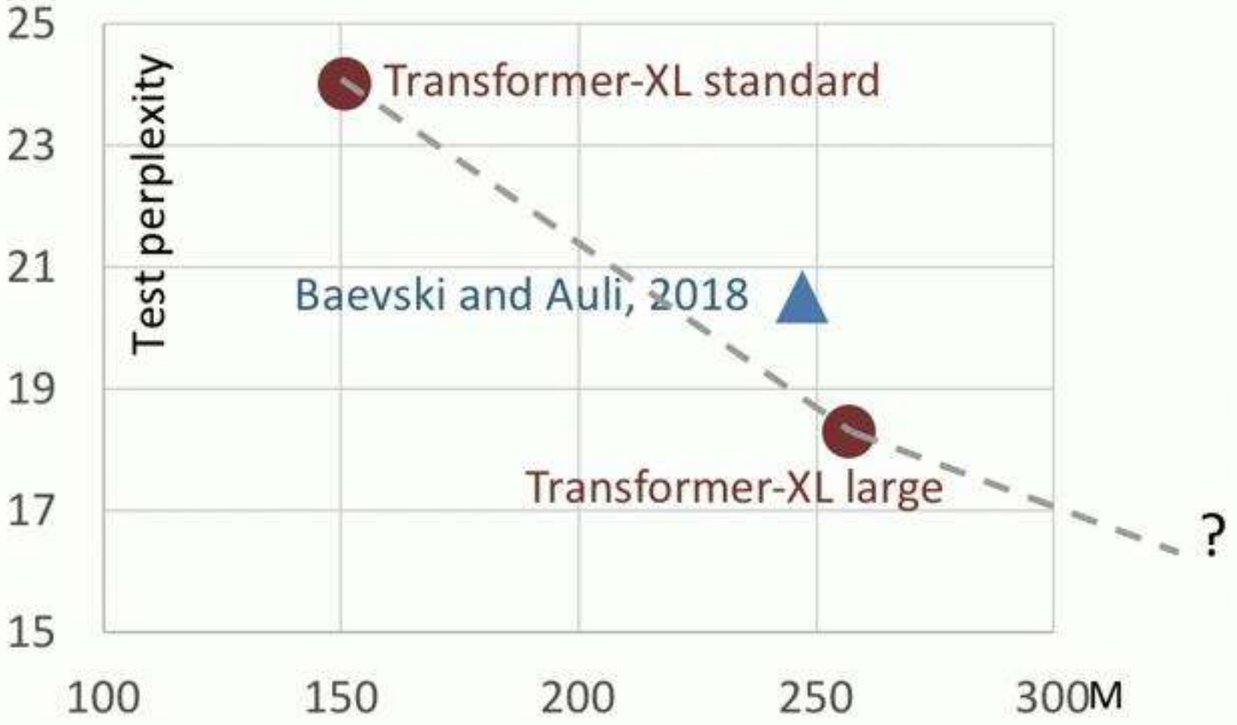
Trend: we can obtain better prediction using larger models

Deep Learning Model Scaling

Image Classification on ImageNet

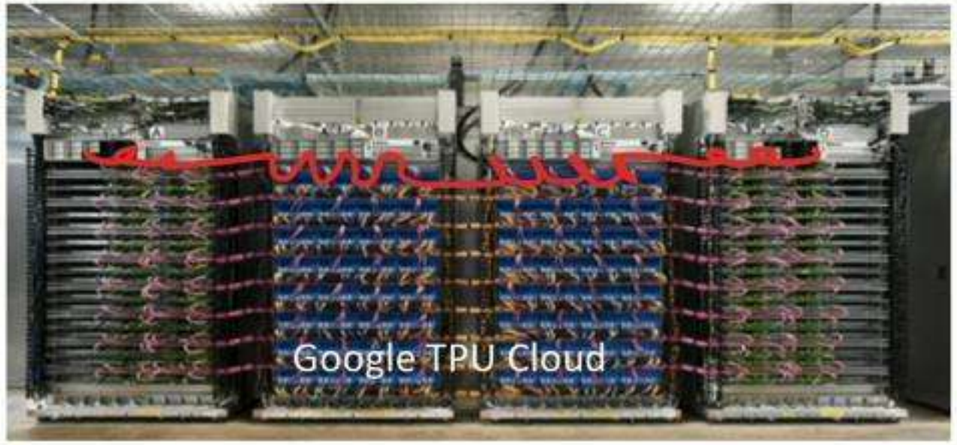


Language Modeling on Wikitext-103



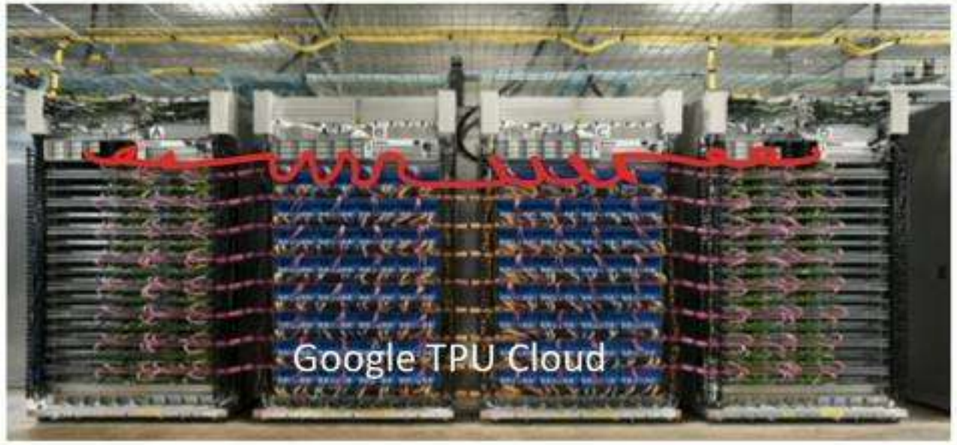
Trend: we can obtain better prediction using larger models

Obstacles of Building Larger Models



- Training larger models is slower
- Longer research & production cycles

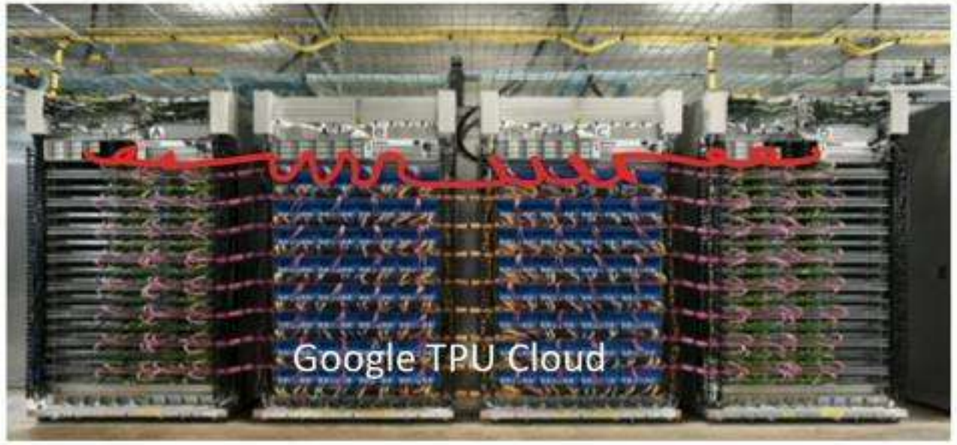
Obstacles of Building Larger Models



- Training larger models is slower
- Longer research & production cycles

- Inference speed is slow
- Challenging to deploy large models to applications with compute and memory constraints

Obstacles of Building Larger Models



- Training larger models is slower
- Longer research & production cycles

- Inference speed is slow
- Challenging to deploy large models to applications with compute and memory constraints

My research: make training and inference faster.

Outline

- Previous Research
  - Distributed deep learning acceleration (15 min)
  - Sparse deep neural networks (20 min)
- Future Research (6 min)

TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning

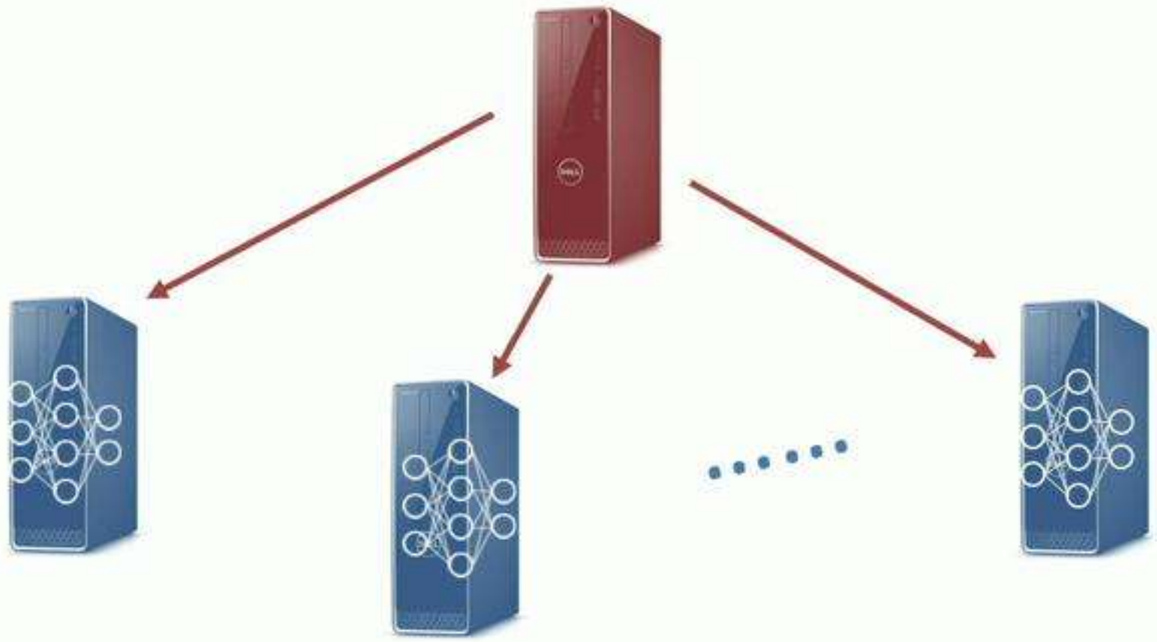
NeurIPS 2017 (Oral)

Wen, Wei, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "Terngrad: Ternary gradients to reduce communication in distributed deep learning." In *Advances in neural information processing systems*, pp. 1509-1519. 2017.

Communication Bottleneck in Distributed Deep Learning

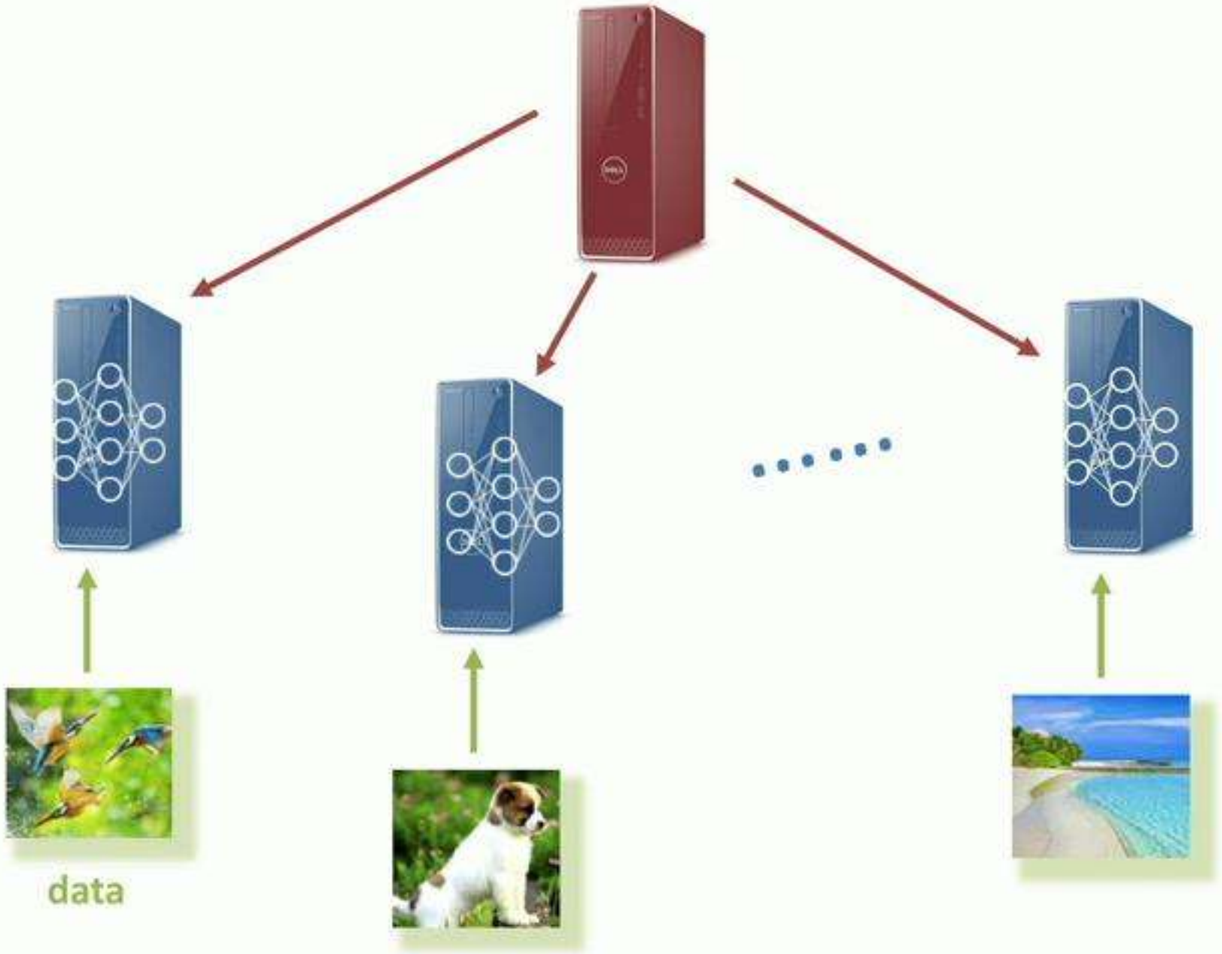
Synchronous SGD

Communication Bottleneck in Distributed Deep Learning



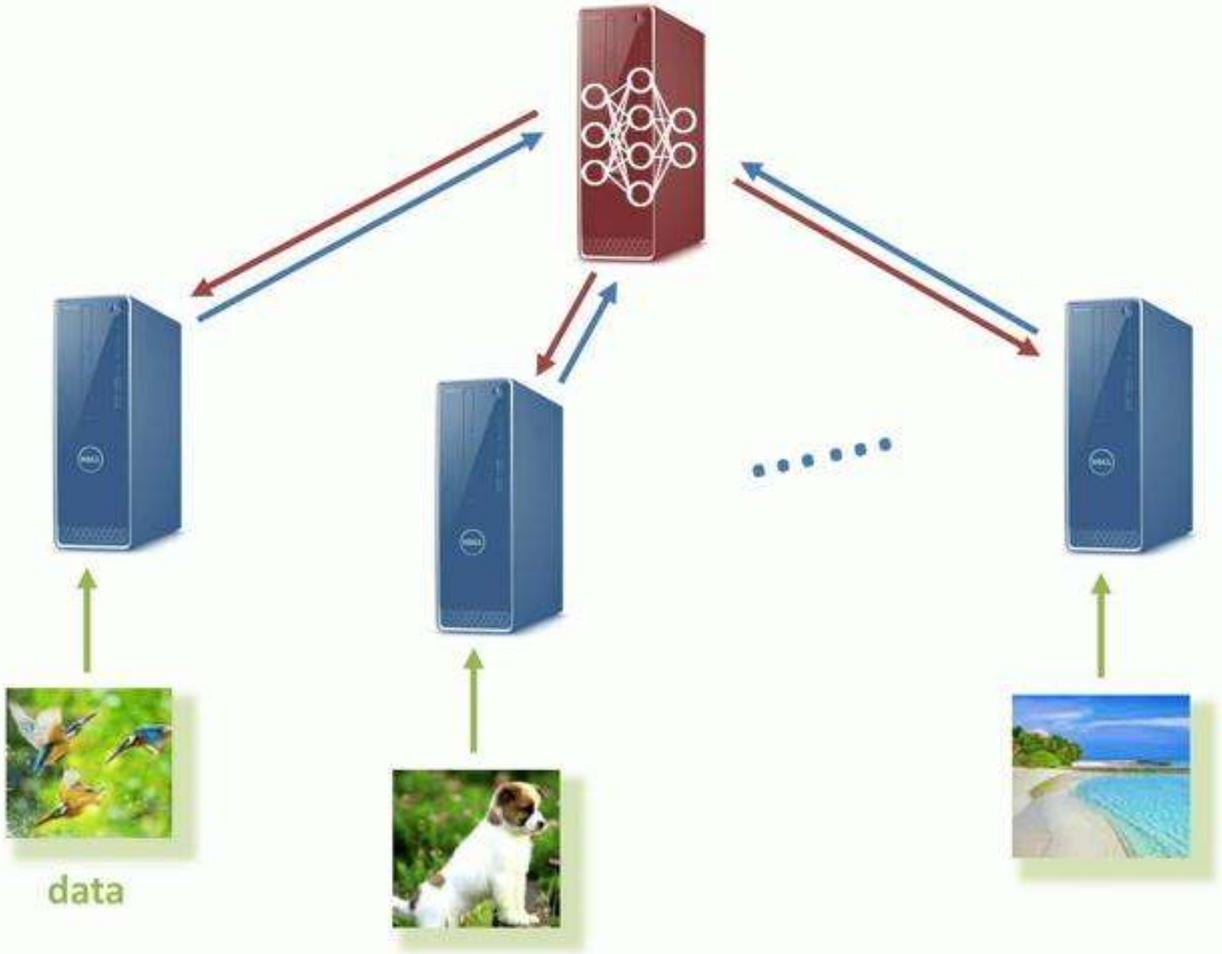
Synchronous SGD

Communication Bottleneck in Distributed Deep Learning



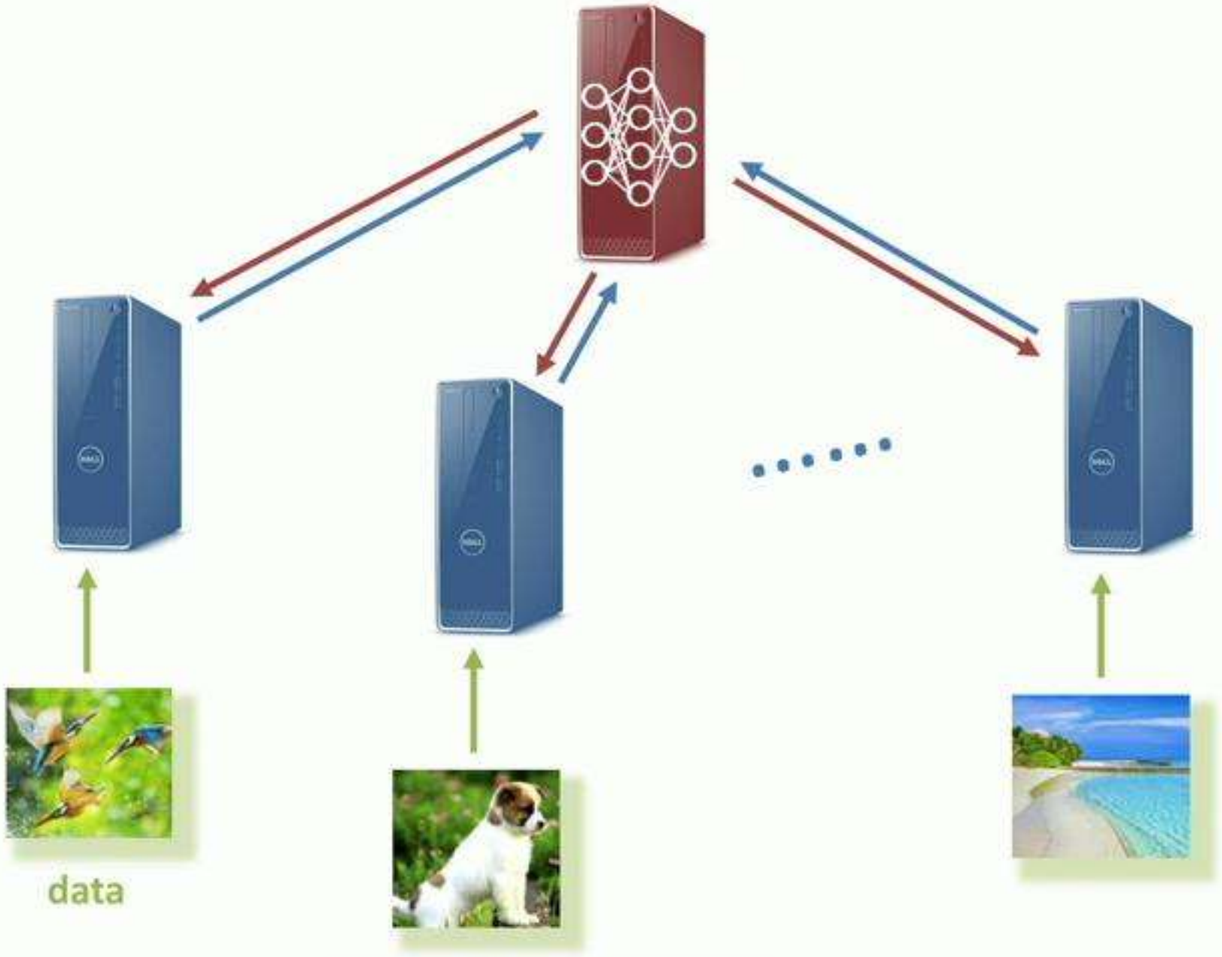
Synchronous SGD

Communication Bottleneck in Distributed Deep Learning

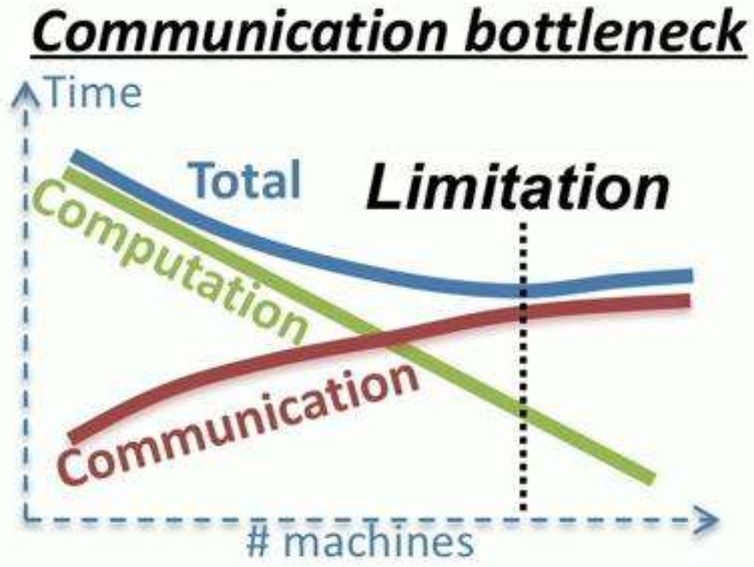


Synchronous SGD

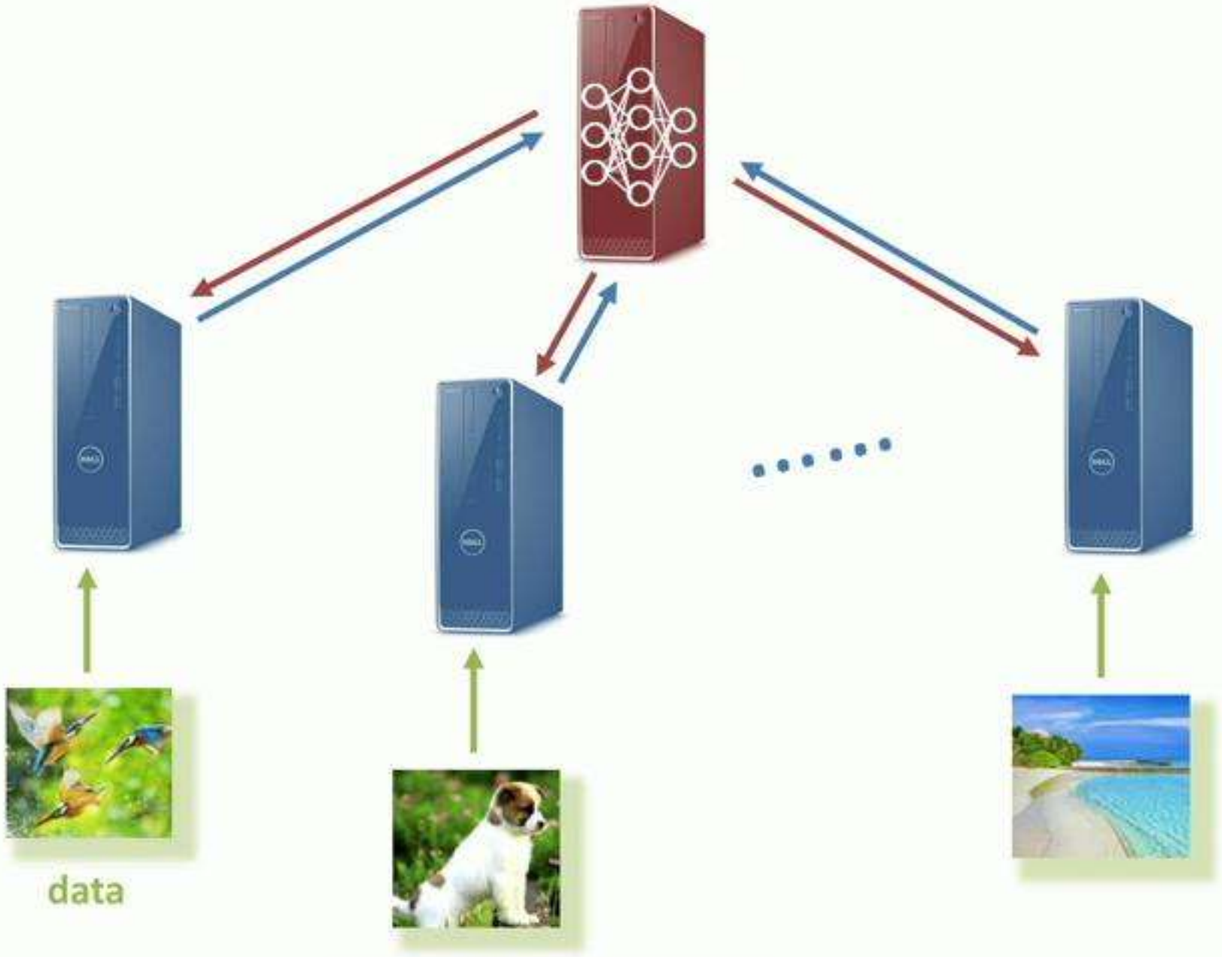
Communication Bottleneck in Distributed Deep Learning



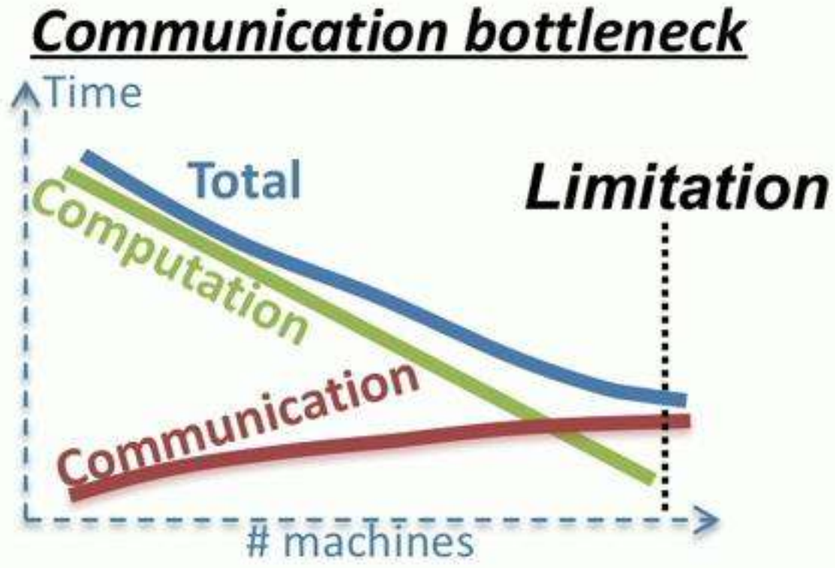
Synchronous SGD



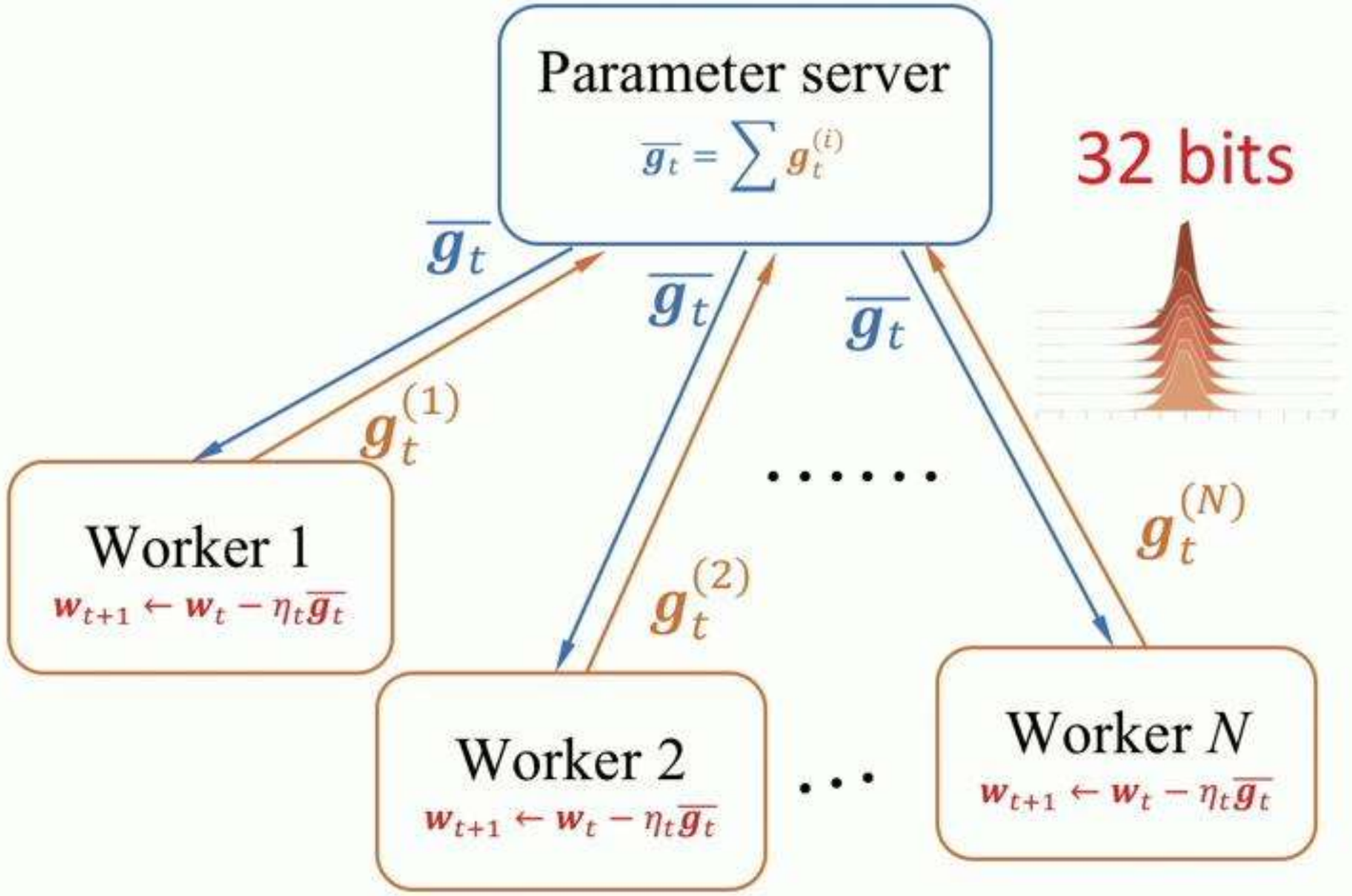
Communication Bottleneck in Distributed Deep Learning



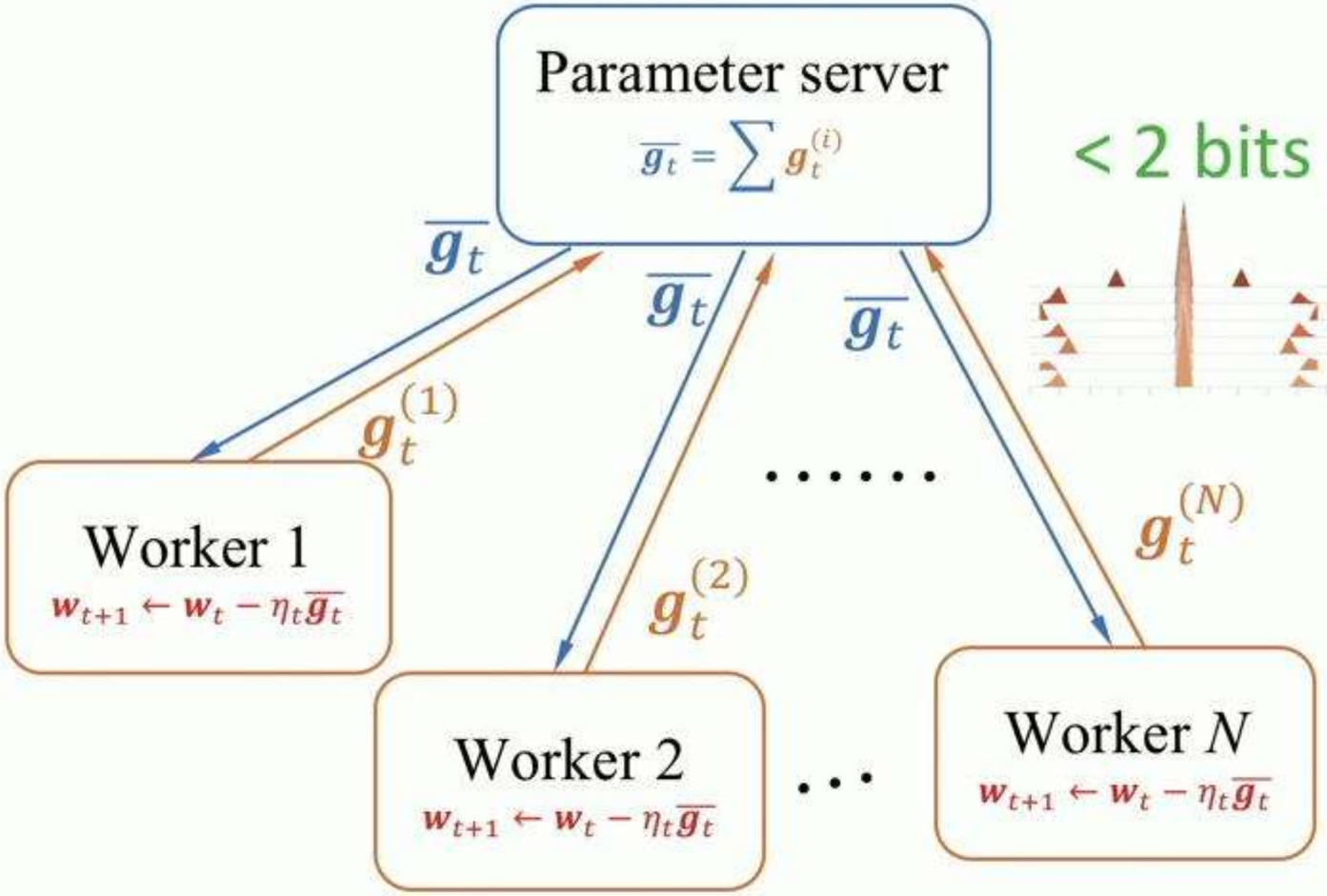
Synchronous SGD



Gradient Quantization for Communication Reduction



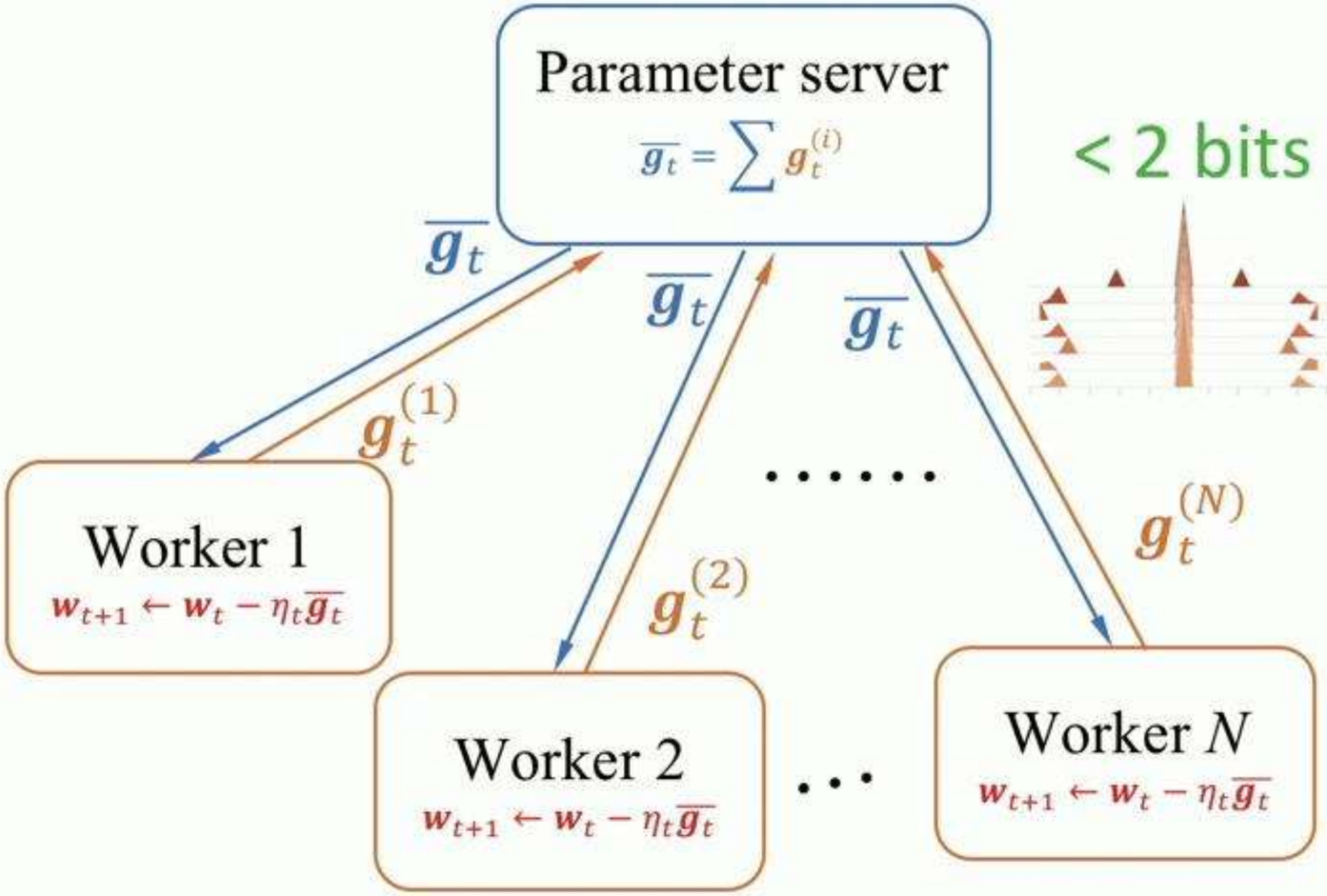
Gradient Quantization for Communication Reduction



< 2 bits (>16x reduction)

ternary gradients (TernGrad)

Gradient Quantization for Communication Reduction



< 2 bits (>16x reduction)

ternary gradients (TernGrad)

It's challenging because it loses precision in the learning.

Basic Idea: Stochastic Gradients without Bias

Batch Gradient Descent

$$C(\boldsymbol{w}) \triangleq \frac{1}{n}\sum_{i=1}^{n} Q(\boldsymbol{z}_i, \boldsymbol{w})$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta_t}{n}\sum_{i=1}^{n} g_t^{(i)}$$

SGD

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \cdot g_t^{(I)}$$

I is randomly drawn from [1,n]

$$\mathrm{E}\left\{g_t^{(I)}\right\} = \nabla C(\boldsymbol{w})$$

No bias

TernGrad

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \cdot ternarize\left(g_t^{(I)}\right)$$

$$\mathrm{E}\left\{ternarize\left(g_t^{(I)}\right)\right\} = \nabla C(\boldsymbol{w})$$

No bias

TernGrad is Simple

$$\tilde{\boldsymbol{g}}_t = ternarize(\boldsymbol{g}_t) = s_t \cdot sign\left(\boldsymbol{g}_t\right) \circ \boldsymbol{b}_t$$

$$s_t \triangleq ||\boldsymbol{g}_t||_\infty \triangleq max\left(abs\left(\boldsymbol{g}_t\right)\right)$$

$$\begin{cases} P(b_{tk} = 1 \mid \boldsymbol{g}_t) = |g_{tk}|/s_t \\ P(b_{tk} = 0 \mid \boldsymbol{g}_t) = 1 - |g_{tk}|/s_t \end{cases}$$

Example:

$\boldsymbol{g}_t^{(i)}$: [0.30, -1.20, ..., 0.9]

s_t: 1.20

Signs: [1, -1, ..., 1]

$P(b_{tk} = 1|g_t)$: $[\frac{0.3}{1.2}, \frac{1.2}{1.2}, \ldots, \frac{0.9}{1.2}]$

$\boldsymbol{b}_t$: [0,1,...,1]

$\widetilde{\boldsymbol{g}_t^{(i)}}$: [0, -1, ..., 1]*1.20

TernGrad is Simple

$$\tilde{\boldsymbol{g}}_t = ternarize(\boldsymbol{g}_t) = s_t \cdot sign\left(\boldsymbol{g}_t\right) \circ \boldsymbol{b}_t$$

$$s_t \triangleq ||\boldsymbol{g}_t||_\infty \triangleq max\left(abs\left(\boldsymbol{g}_t\right)\right)$$

$$\begin{cases} P(b_{tk} = 1 \mid \boldsymbol{g}_t) = |g_{tk}|/s_t \\ P(b_{tk} = 0 \mid \boldsymbol{g}_t) = 1 - |g_{tk}|/s_t \end{cases}$$

Example:

$\boldsymbol{g}_t^{(i)}$: [0.30, -1.20, ..., 0.9]

s_t: 1.20

Signs: [1, -1, ..., 1]

$P(b_{tk} = 1|g_t)$: $[\frac{0.3}{1.2}, \frac{1.2}{1.2}, \ldots, \frac{0.9}{1.2}]$

$\boldsymbol{b}_t$: [0,1,...,1]

$\widetilde{\boldsymbol{g}_t^{(i)}}$: [0, -1, ..., 1]*1.20

$$\mathbf{E}_{\boldsymbol{z},\boldsymbol{b}}\left\{\tilde{\boldsymbol{g}}_t\right\} = \mathbf{E}_{\boldsymbol{z},\boldsymbol{b}}\left\{s_t \cdot sign\left(\boldsymbol{g}_t\right) \circ \boldsymbol{b}_t\right\}$$

$$= \mathbf{E}_{\boldsymbol{z}}\left\{s_t \cdot sign\left(\boldsymbol{g}_t\right) \circ \mathbf{E}_{\boldsymbol{b}}\left\{\boldsymbol{b}_t|\boldsymbol{z}_t\right\}\right\} = \mathbf{E}_{\boldsymbol{z}}\left\{\boldsymbol{g}_t\right\} = \nabla_{\boldsymbol{w}} C(\boldsymbol{w}_t)$$

No bias

Convergence

Standard SGD almost truly converges under assumptions (Fisk 1965, Metivier 1981&1983, Bottou 1998)

Assumption 1:

$C(\boldsymbol{w})$ has a single minimum $\boldsymbol{w}^*$ and $\forall \epsilon > 0, \inf_{||\boldsymbol{w}-\boldsymbol{w}^*||^2>\epsilon} (\boldsymbol{w}-\boldsymbol{w}^*)^T \nabla_{\boldsymbol{w}} C(\boldsymbol{w}) > 0$

Assumption 2:

Learning rate γ_t decreases neither very fast nor very slow $\begin{cases} \sum_{t=0}^{+\infty} \gamma_t^2 < +\infty \\ \sum_{t=0}^{+\infty} \gamma_t = +\infty \end{cases}$

Assumption 3 (gradient bound):

$\mathbf{E}\left\{||\boldsymbol{g}||^2\right\} \leq A + B\,||\boldsymbol{w}-\boldsymbol{w}^*||^2$

Standard SGD *almost-truly* converges

Convergence

Standard SGD almost truly converges under assumptions (Fisk 1965, Metivier 1981&1983, Bottou 1998)

Assumption 1:

$C(\boldsymbol{w})$ has a single minimum $\boldsymbol{w}^*$ and $\forall \epsilon > 0, \inf_{||\boldsymbol{w}-\boldsymbol{w}^*||^2>\epsilon} (\boldsymbol{w}-\boldsymbol{w}^*)^T \nabla_{\boldsymbol{w}} C(\boldsymbol{w}) > 0$

Assumption 2:

Learning rate γ_t decreases neither very fast nor very slow $\begin{cases} \sum_{t=0}^{+\infty} \gamma_t^2 < +\infty \\ \sum_{t=0}^{+\infty} \gamma_t = +\infty \end{cases}$

Assumption 3 (gradient bound):

$\mathbf{E}\left\{||\boldsymbol{g}||^2\right\} \leq A + B\,||\boldsymbol{w}-\boldsymbol{w}^*||^2$

Standard SGD *almost-truly* converges

Assumption 3 (gradient bound):

$\mathbf{E}\left\{||\boldsymbol{g}||_\infty \cdot ||\boldsymbol{g}||_1\right\} \leq A + B\,||\boldsymbol{w}-\boldsymbol{w}^*||^2$

TernGrad almost-truly converges

Convergence

Standard SGD almost truly converges under assumptions (Fisk 1965, Metivier 1981&1983, Bottou 1998)

Assumption 1:

$C(\boldsymbol{w})$ has a single minimum $\boldsymbol{w}^*$ and $\forall \epsilon > 0, \inf_{||\boldsymbol{w}-\boldsymbol{w}^*||^2>\epsilon} (\boldsymbol{w}-\boldsymbol{w}^*)^T \nabla_{\boldsymbol{w}} C(\boldsymbol{w}) > 0$

Assumption 2:

Learning rate γ_t decreases neither very fast nor very slow $\begin{cases} \sum_{t=0}^{+\infty} \gamma_t^2 < +\infty \\ \sum_{t=0}^{+\infty} \gamma_t = +\infty \end{cases}$

Assumption 3 (gradient bound):

$\mathbf{E}\left\{||\boldsymbol{g}||^2\right\} \leq A + B\,||\boldsymbol{w}-\boldsymbol{w}^*||^2$

Standard SGD *almost-truly* converges

Assumption 3 (gradient bound):

$\mathbf{E}\left\{||\boldsymbol{g}||_\infty \cdot ||\boldsymbol{g}||_1\right\} \leq A + B\,||\boldsymbol{w}-\boldsymbol{w}^*||^2$

TernGrad almost-truly converges

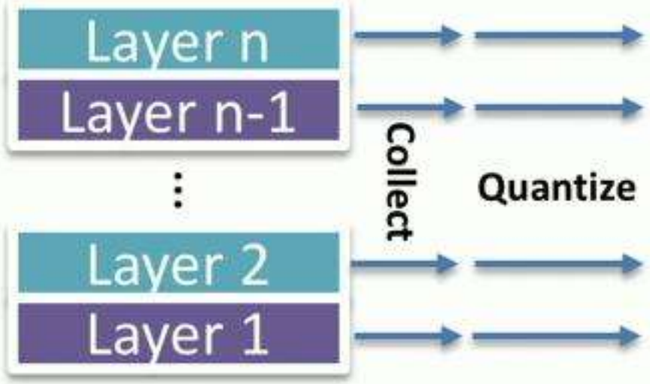
$$\mathbf{E}\left\{||\boldsymbol{g}||^2\right\} \leq \mathbf{E}\left\{||\boldsymbol{g}||_\infty \cdot ||\boldsymbol{g}||_1\right\} \leq A + B\,||\boldsymbol{w}-\boldsymbol{w}^*||^2$$

Stronger gradient bound in *TernGrad*

Closing Bound Gap

Two methods to push the gradient bound of *TernGrad* closer to the bound of standard SGD

Layer-wise ternarizing



Gradient clipping

Histogram of gradients

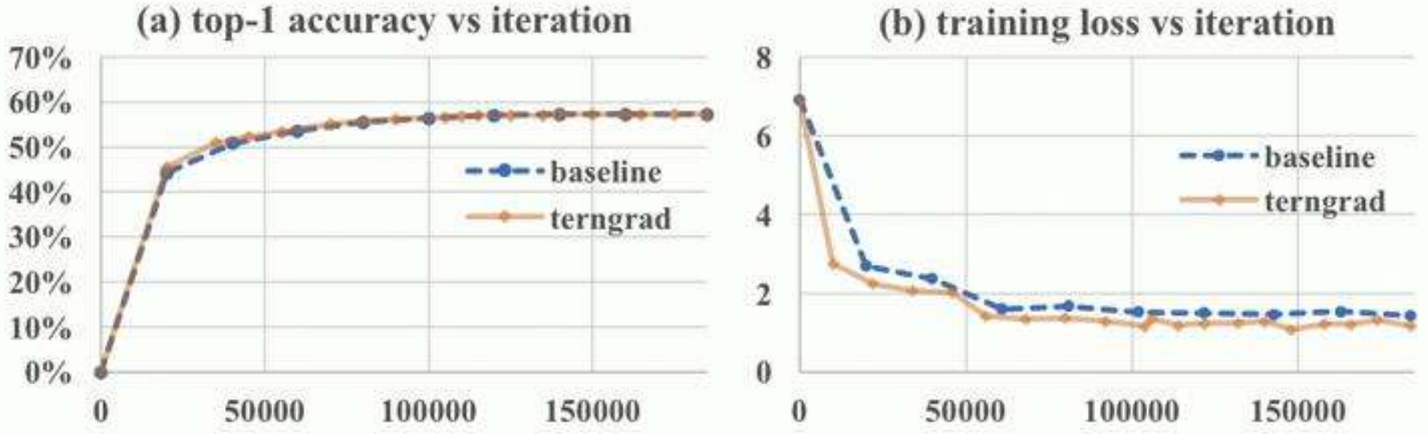
Training ImageNet with Ternary Gradients

AlexNet

base LR	mini-batch size	workers	gradients	top-1
0.01	256	2	floating	57.33%
			TernGrad	57.61%
			TernGrad-noclip	54.63%
0.02	512	4	floating	57.32%
			TernGrad	57.28%
0.04	1024	8	floating	**56.62%**
			TernGrad	**57.54%**

Keskar, et al., 2017

4 workers



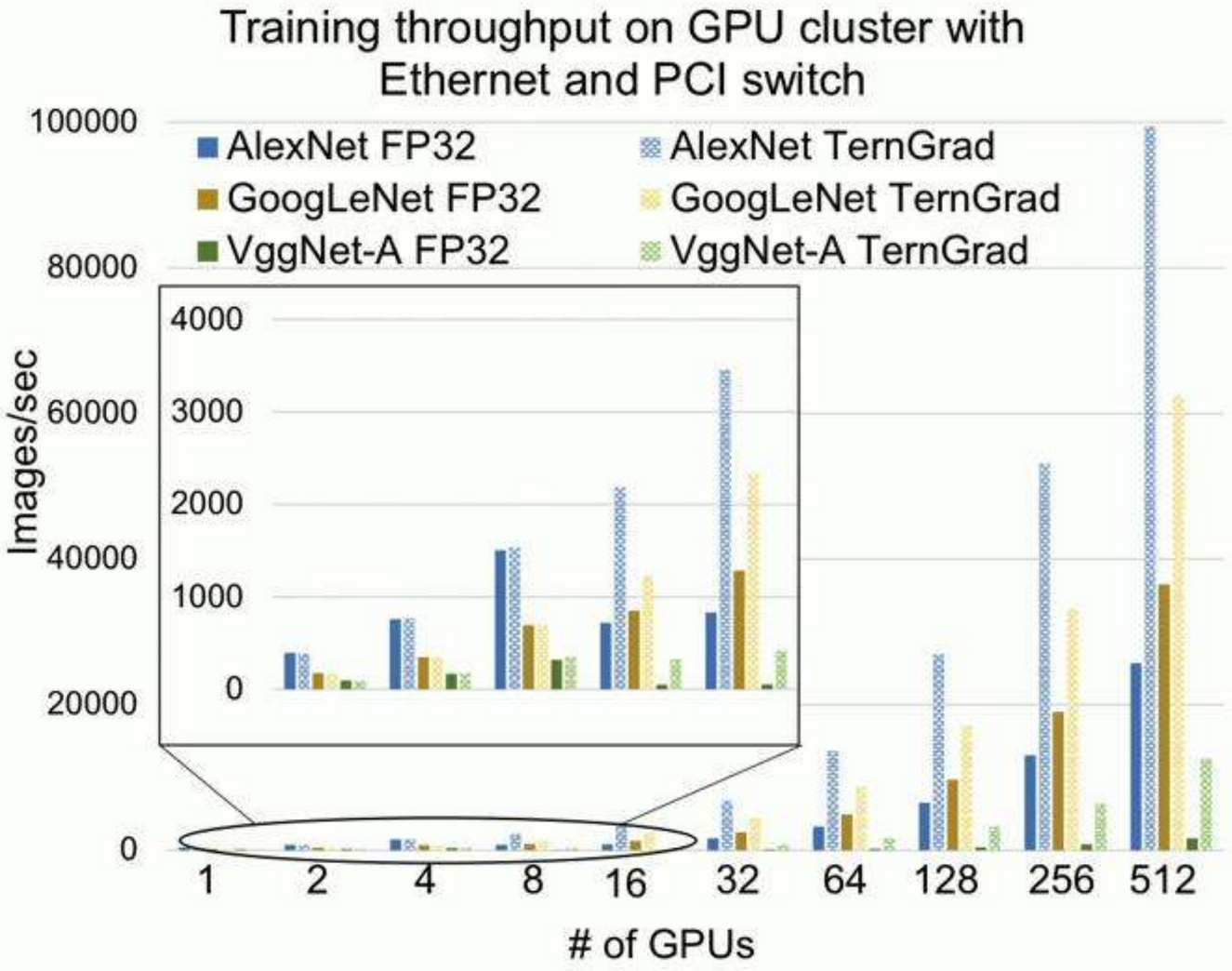
Training ImageNet with Ternary Gradients

GoogLeNet accuracy loss is <2% on avgerage.

base LR	mini-batch size	workers	gradients	top-5
0.04	128	2	floating	88.30%
			TernGrad	86.77%
0.08	256	4	floating	87.82%
			TernGrad	85.96%
0.10	512	8	floating	89.00%
			TernGrad	86.47%

All hyper-parameters in standard SGD are directed used in *TernGrad* (such as learning rate, batch size, total epochs, etc.)

Speedup



TernGrad gives higher speedup when communication time occupies a higher ratio, such as:

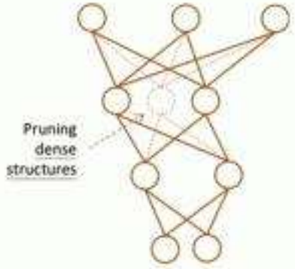
1. using more machines;
2. having smaller communication bandwidth, e.g., Ethernet vs. InfiniBand;
3. training a DNN with a higher "parameter #: computation #"
4. using GPU-based distributed systems vs. CPU-based

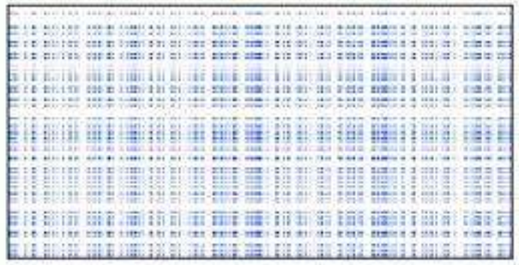
TernGrad in Production

- Facebook is using *TernGrad* to overcome communication bottleneck in AI Infra
- Effectiveness is evaluated by ad ranking models, which have zero toleration on prediction loss
- Available in PyTorch/Caffe2

Inference Acceleration

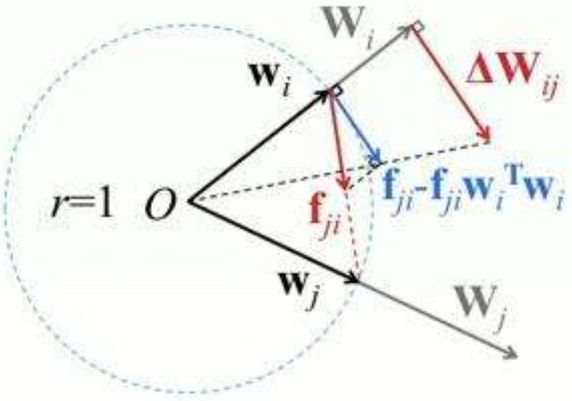
Wen et al. NeurIPS 2016



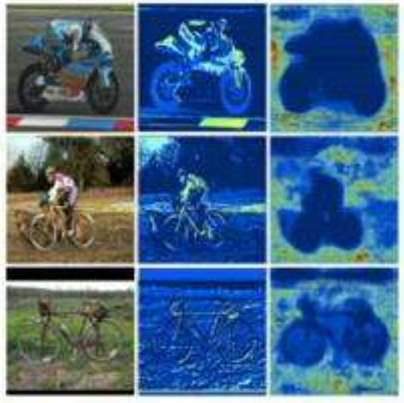


Wen et al. ICLR 2018

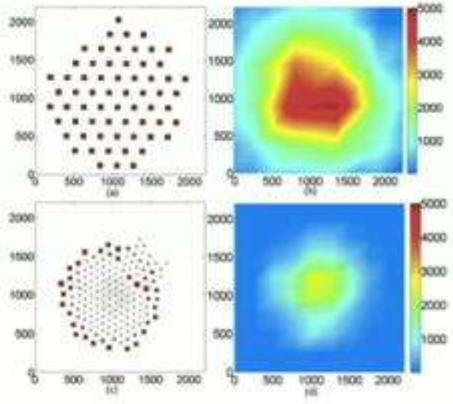
Wang et al. ASP-DAC 2017
(Best Paper Award)



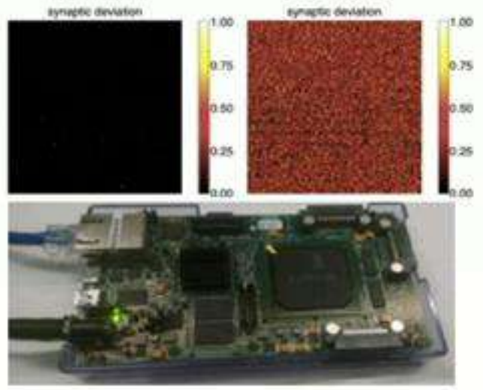
Wen et al. ICCV 2017



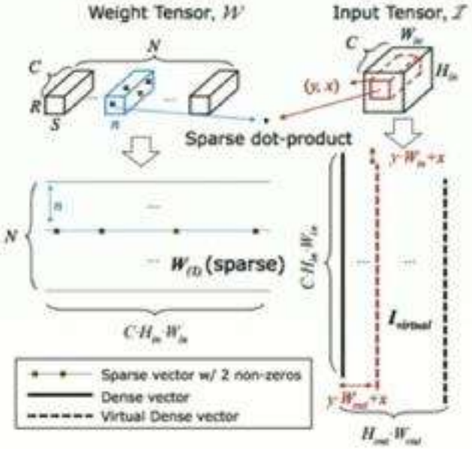
Wu et al. CVPR 2017



Wen et al. DAC 2015
(Best Paper Nomination)



Wen et al. DAC 2016
(Best Paper Nomination)



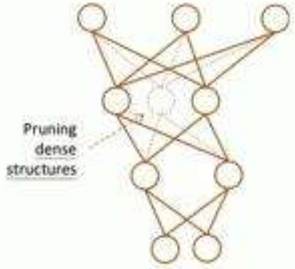
Park et al. ICLR 2017

TernGrad in Production

- Facebook is using *TernGrad* to overcome communication bottleneck in AI Infra
- Effectiveness is evaluated by ad ranking models, which have zero toleration on prediction loss
- Available in PyTorch/Caffe2

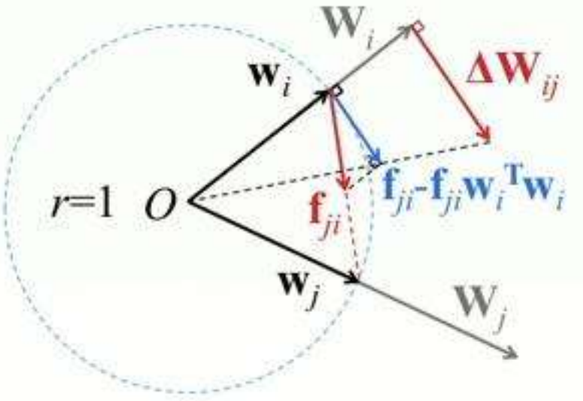
Inference Acceleration

Wen et al. NeurIPS 2016

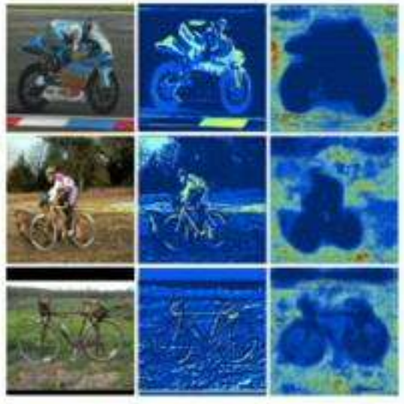


Wen et al. ICLR 2018

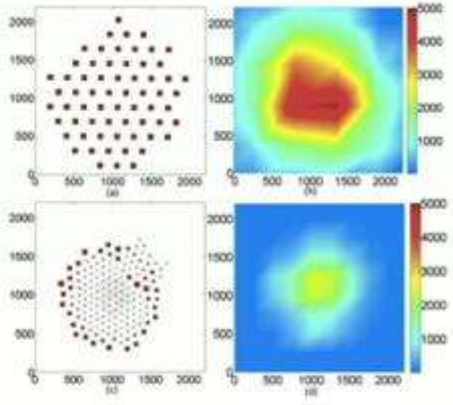
Wang et al. ASP-DAC 2017
(Best Paper Award)



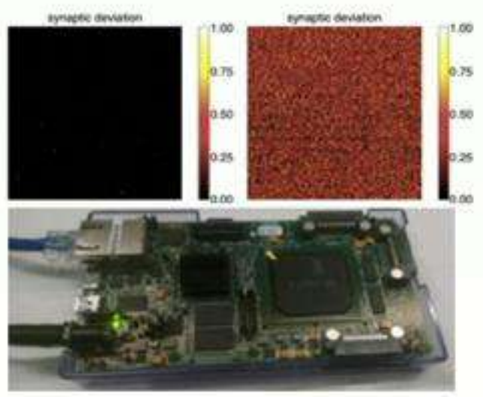
Wen et al. ICCV 2017



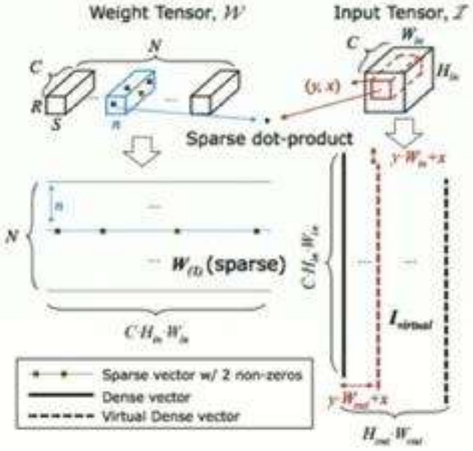
Wu et al. CVPR 2017



Wen et al. DAC 2015
(Best Paper Nomination)

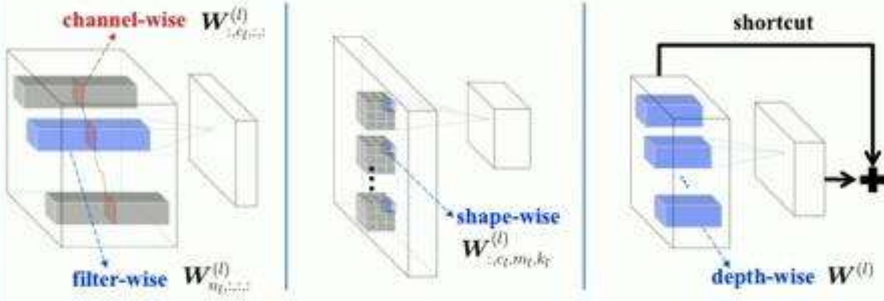


Wen et al. DAC 2016
(Best Paper Nomination)



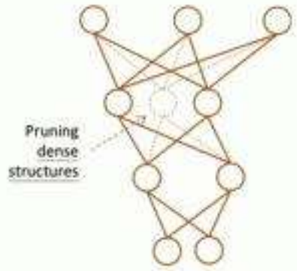
Park et al. ICLR 2017

Inference Acceleration



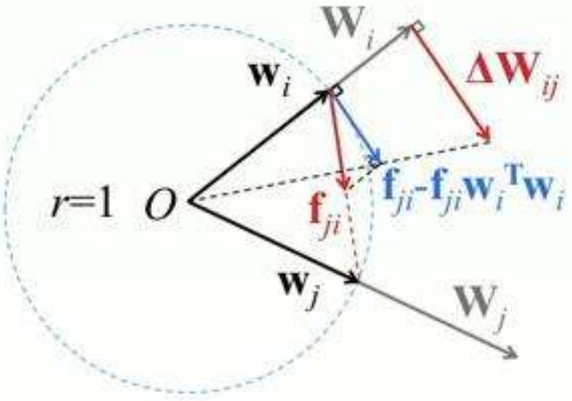
Wen et al. NeurIPS 2016

This presentation

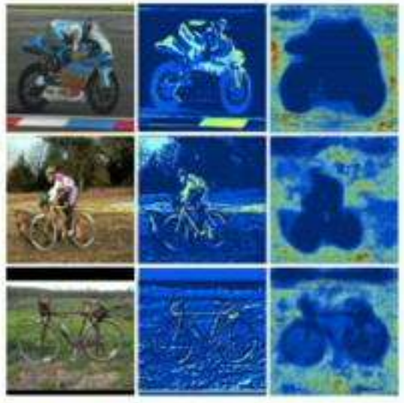


Wen et al. ICLR 2018

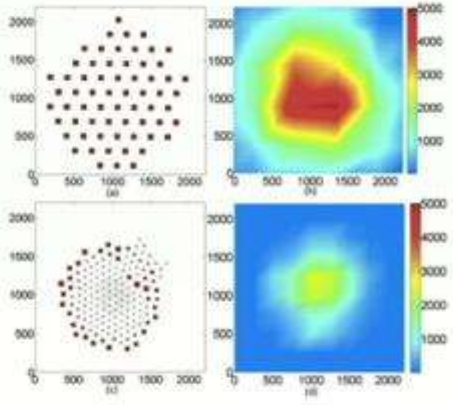
Wang et al. ASP-DAC 2017
(Best Paper Award)



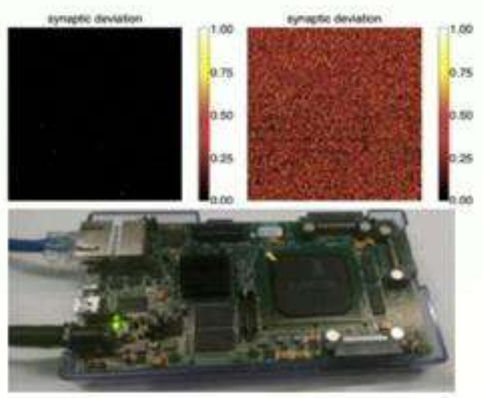
Wen et al. ICCV 2017



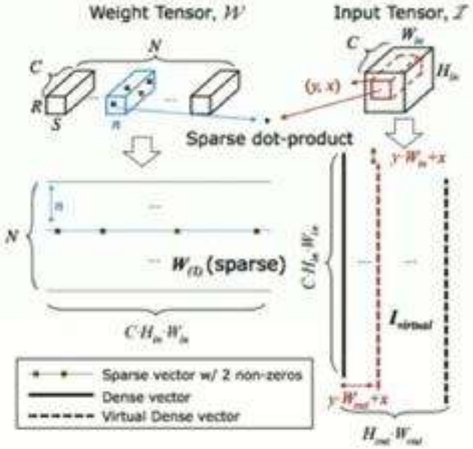
Wu et al. CVPR 2017



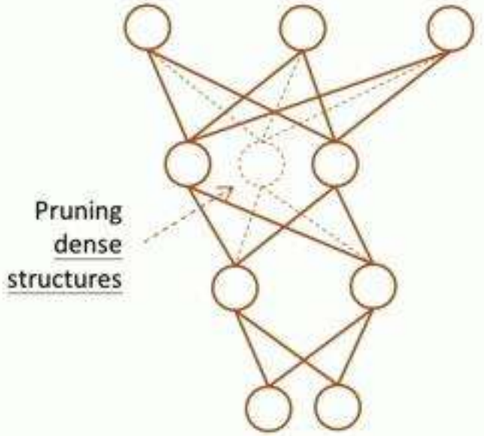
Wen et al. DAC 2015
(Best Paper Nomination)



Wen et al. DAC 2016
(Best Paper Nomination)



Park et al. ICLR 2017

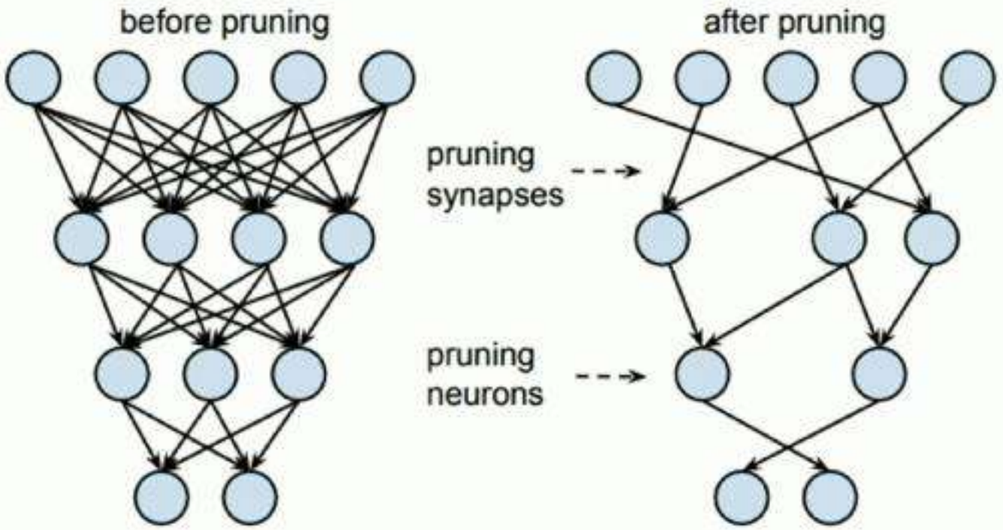


Sparse Deep Neural Networks

Wen, Wei, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "Learning structured sparsity in deep neural networks." In *Advances in neural information processing systems*, pp. 2074-2082. 2016.

Wen, Wei, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. "Learning intrinsic sparse structures within long short-term memory." *ICLR* 2018.

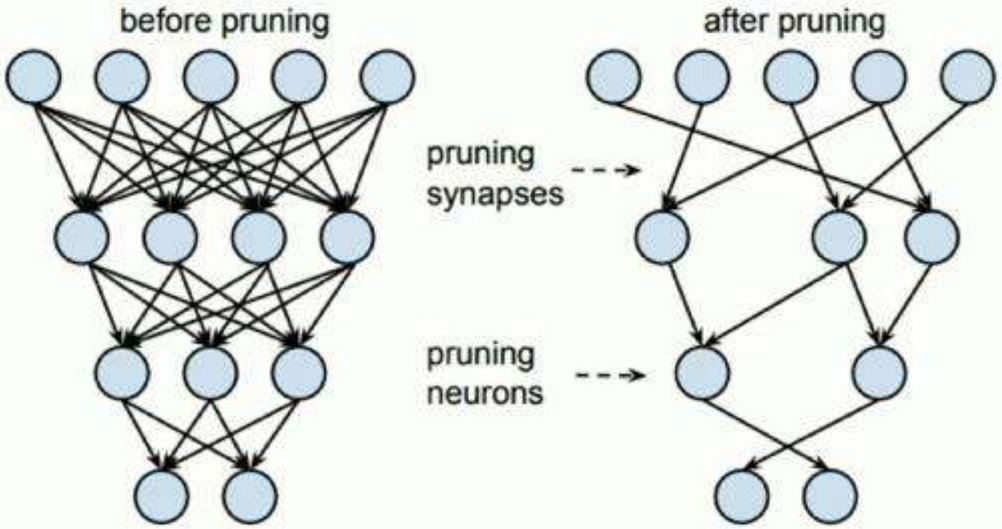
Inefficiency of Sparse Convolutional Neural Networks

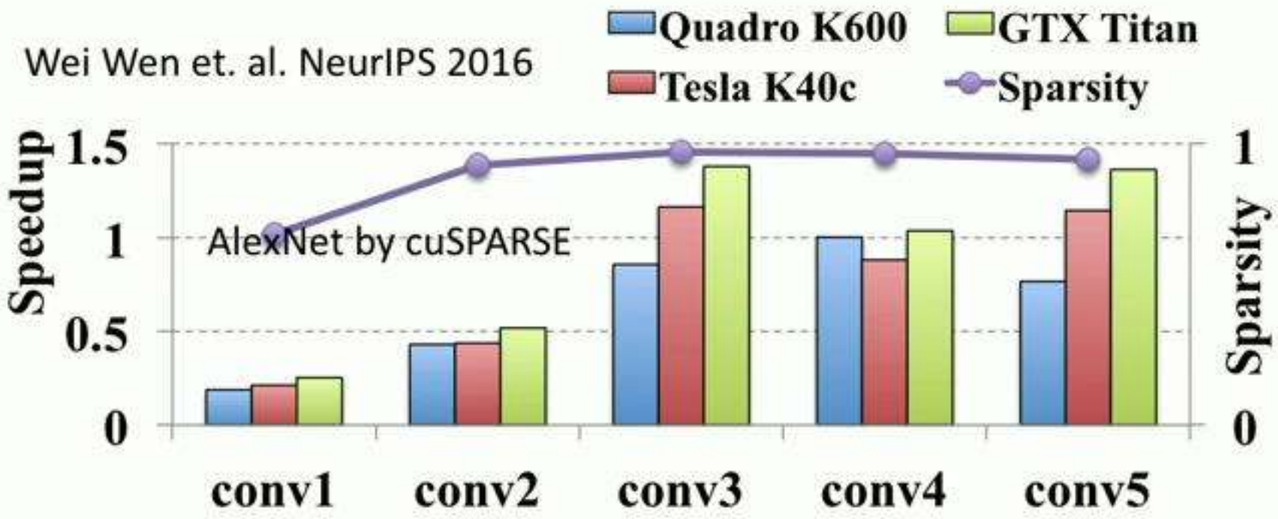


Significantly reduces the storage size of DNNs [1]
Good speedup with customized hardware [2]

[1] S. Han et. al. NeurIPS 2015
[2] S. Han et. al. ISCA 2016.

Inefficiency of Sparse Convolutional Neural Networks

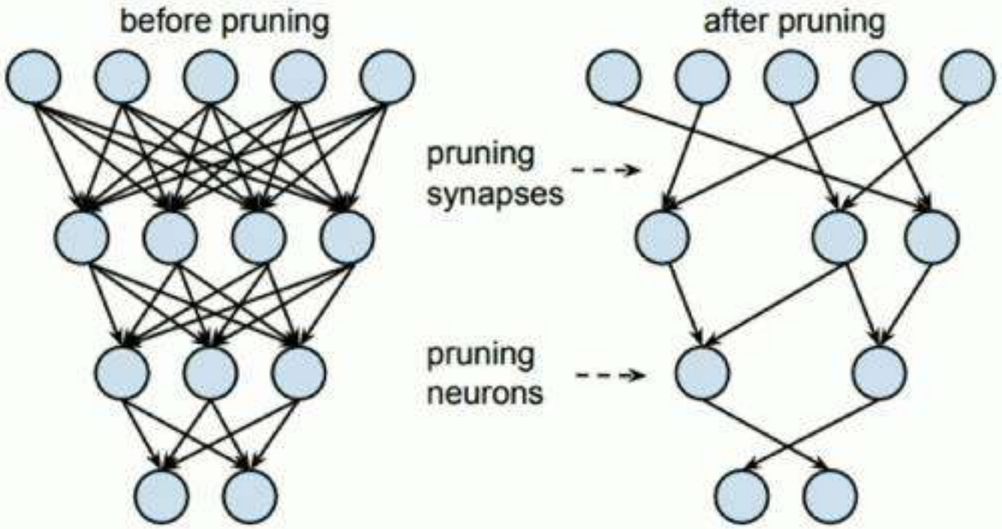




Significantly reduces the storage size of DNNs [1]
Good speedup with customized hardware [2]

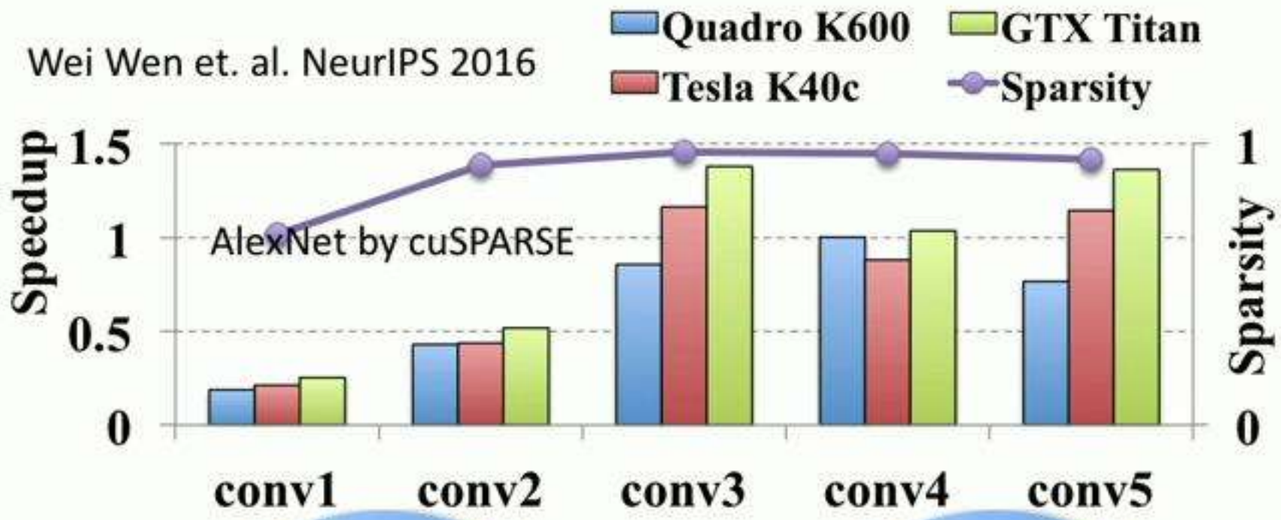
[1] S. Han et. al. NeurIPS 2015
[2] S. Han et. al. ISCA 2016.

Inefficiency of Sparse Convolutional Neural Networks



Significantly reduces the storage size of DNNs [1]
Good speedup with customized hardware [2]

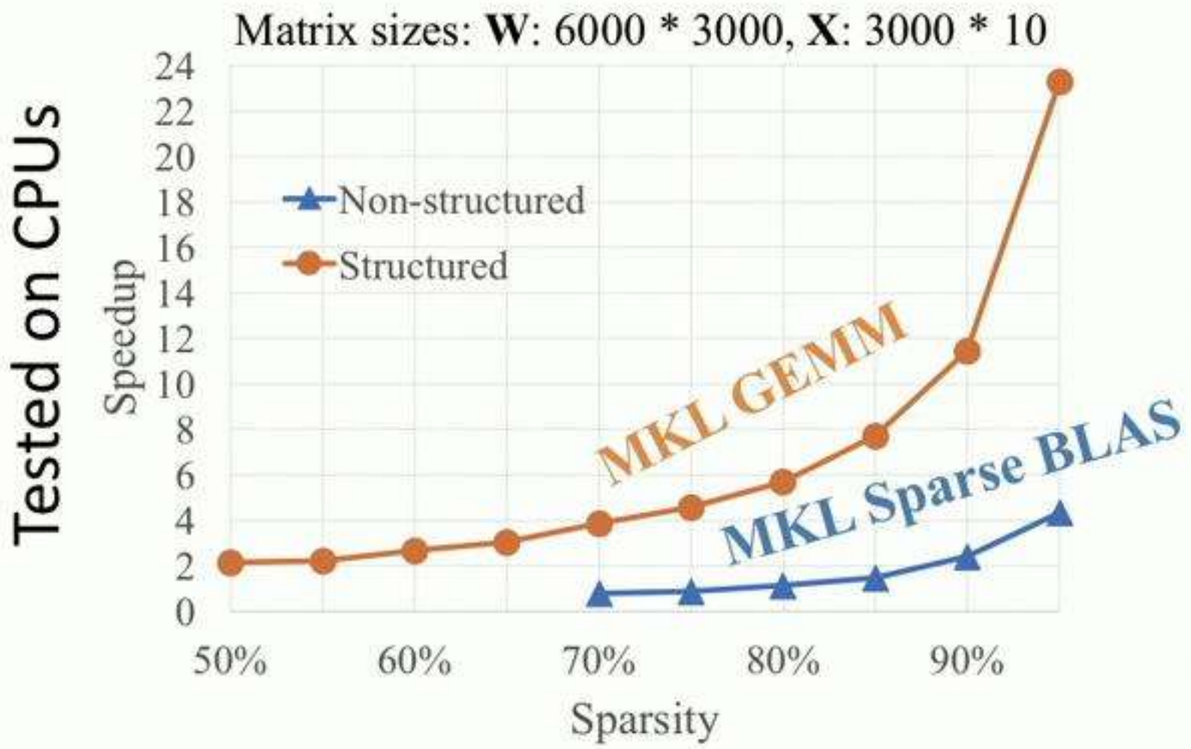
[1] S. Han et. al. NeurIPS 2015
[2] S. Han et. al. ISCA 2016.



Efficiency of Structured Sparsity

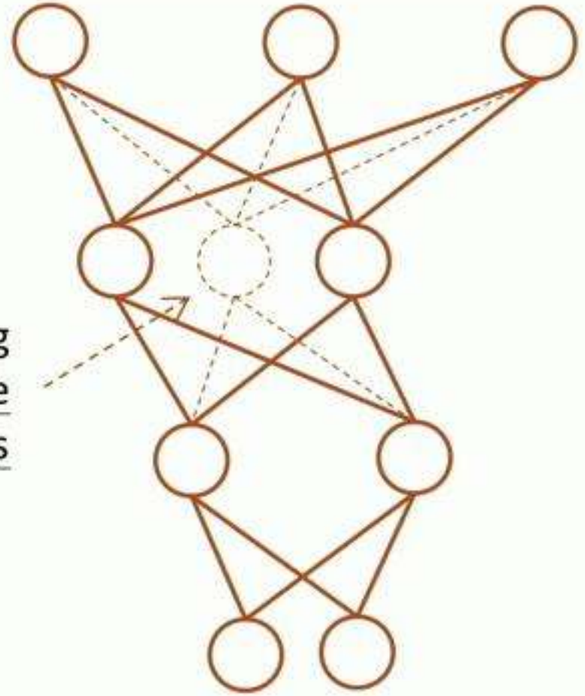
Structured sparsity

Non-structured sparsity



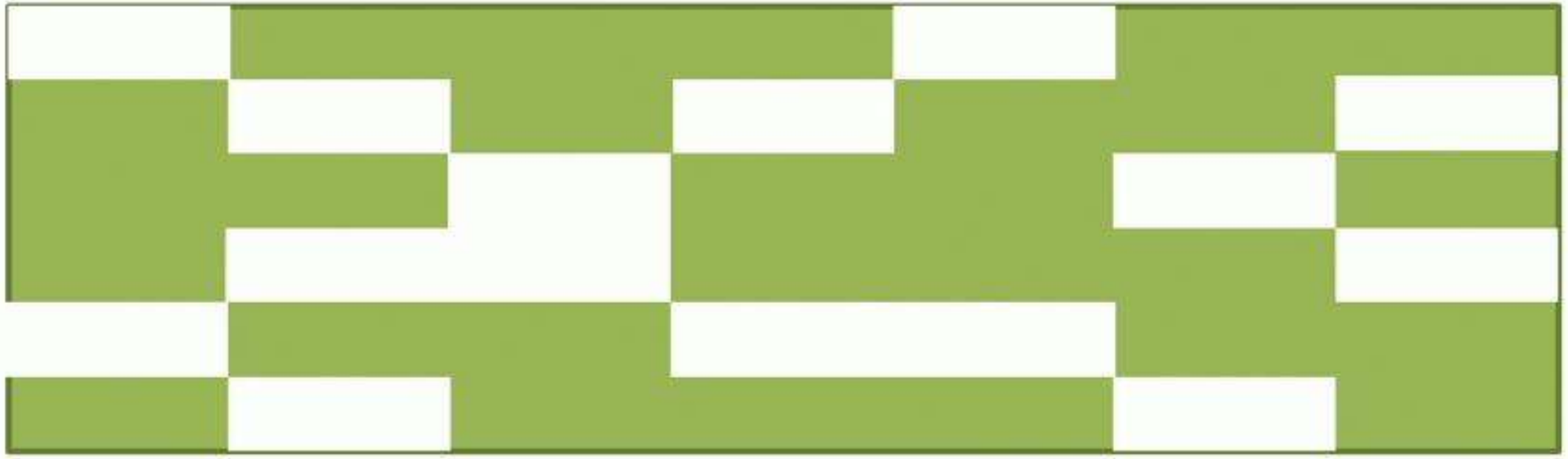
Structurally Sparse DNNs (SSDNNs)

Neural Architecture



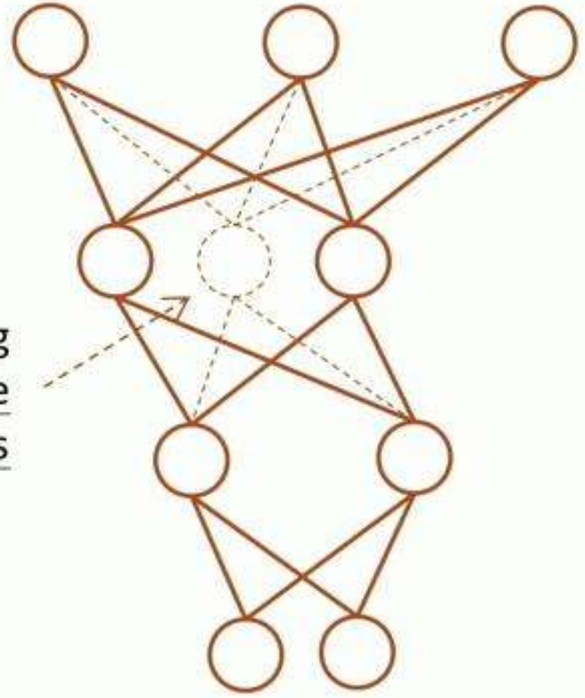
Pruning dense structures

Weight matrix



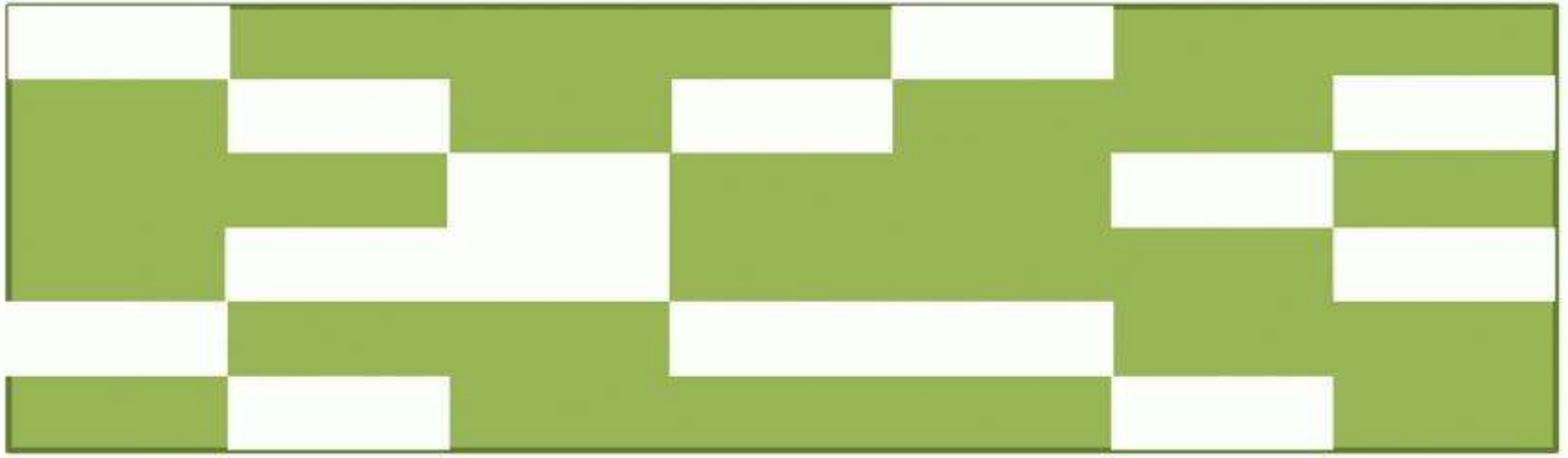
Structurally Sparse DNNs (SSDNNs)

Neural Architecture



Pruning dense structures

Weight matrix



Connections/Weights are removed group by group.

In neural architecture: a group can be a neuron, a layer, a filter, a hidden state, etc.

In weight matrix: a group can be a rectangle block, a row, a column or even the whole matrix

Group Lasso [1] Regularization is All you Need for SSDNNs

Step 1: Weights are split to G groups $\mathbf{w}^{(1..G)}$

e.g. $(w_1, w_2, w_3, w_4, w_5)$ -> group 1: $\mathbf{w}^{(1)} = (w_1, w_2, w_3)$, group 2: $\mathbf{w}^{(2)} = (w_4, w_5)$

[1] Yuan et al. 2006

Group Lasso [1] Regularization is All you Need for SSDNNs

Step 1: Weights are split to G groups $\mathbf{w}^{(1..G)}$

e.g. (w_1, w_2, w_3, w_4, w_5) -> group 1: $\mathbf{w}^{(1)}$ = (w_1, w_2, w_3), group 2: $\mathbf{w}^{(2)}$ = (w_4, w_5)

Step 2: Add group Lasso on each group $\mathbf{w}^{(g)}$ $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$ i.e. vector length

sqrt($w_1^2 + w_2^2 + w_3^2$), sqrt($w_4^2 + w_5^2$)

[1] Yuan et al. 2006

Group Lasso [1] Regularization is All you Need for SSDNNs

Step 1: Weights are split to *G* groups $\mathbf{w}^{(1..G)}$

e.g. (w_1, w_2, w_3, w_4, w_5) -> group 1: $\mathbf{w}^{(1)}$ = (w_1, w_2, w_3), group 2: $\mathbf{w}^{(2)}$ = (w_4, w_5)

Step 2: Add group Lasso on each group $\mathbf{w}^{(g)}$ $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$ i.e. vector length

sqrt($w_1^2 + w_2^2 + w_3^2$), sqrt($w_4^2 + w_5^2$)

Step 3: Sum group Lasso over all groups as a regularization: $R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g$,

sqrt($w_1^2 + w_2^2 + w_3^2$) + sqrt($w_4^2 + w_5^2$)

[1] Yuan et al. 2006

Group Lasso [1] Regularization is All you Need for SSDNNs

Step 1: Weights are split to *G* groups $\mathbf{w}^{(1..G)}$

e.g. (w_1, w_2, w_3, w_4, w_5) -> group 1: $\mathbf{w}^{(1)}$ = (w_1, w_2, w_3), group 2: $\mathbf{w}^{(2)}$ = (w_4, w_5)

Step 2: Add group Lasso on each group $\mathbf{w}^{(g)}$ $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$ i.e. vector length

sqrt($w_1^2 + w_2^2 + w_3^2$), sqrt($w_4^2 + w_5^2$)

Step 3: Sum group Lasso over all groups as a regularization: $R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g$,

sqrt($w_1^2 + w_2^2 + w_3^2$) + sqrt($w_4^2 + w_5^2$)

Step 4: SGD optimizing $\arg\min_{\mathbf{w}}\left\{E\left(\mathbf{w}\right)\right\} = \arg\min_{\mathbf{w}}\left\{E_D\left(\mathbf{w}\right) + \lambda_g \cdot R_g\left(\mathbf{w}\right)\right\}$

[1] Yuan et al. 2006

Group Lasso [1] Regularization is All you Need for SSDNNs

Step 1: Weights are split to *G* groups $\mathbf{w}^{(1..G)}$

e.g. (w_1, w_2, w_3, w_4, w_5) -> group 1: $\mathbf{w}^{(1)}$ = (w_1, w_2, w_3), group 2: $\mathbf{w}^{(2)}$ = (w_4, w_5)

Step 2: Add group Lasso on each group $\mathbf{w}^{(g)}$ $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$ i.e. vector length

sqrt($w_1^2 + w_2^2 + w_3^2$), sqrt($w_4^2 + w_5^2$)

Step 3: Sum group Lasso over all groups as a regularization: $R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g$,

sqrt($w_1^2 + w_2^2 + w_3^2$) + sqrt($w_4^2 + w_5^2$)

Step 4: SGD optimizing $\arg\min_{\mathbf{w}}\left\{E\left(\mathbf{w}\right)\right\} = \arg\min_{\mathbf{w}}\left\{E_D\left(\mathbf{w}\right) + \lambda_g \cdot R_g\left(\mathbf{w}\right)\right\}$

We refer to our method as ***Structured Sparsity Learning (SSL)***

[1] Yuan et al. 2006

Structured Sparsity Learning (*SSL*)

- Why can SSL learn to remove weights group by group?

The gradient-based explanation:

$$\mathbf{w}^{(g)} \leftarrow \mathbf{w}^{(g)} - \eta \left(\frac{\partial E_D(\mathbf{w})}{\partial \mathbf{w}^{(g)}} + \lambda_g \frac{\mathbf{w}^{(g)}}{\|\mathbf{w}^{(g)}\|_2} \right)$$

A group of weights

Regular gradients to minimize error

Structured Sparsity Learning (*SSL*)

- Why can SSL learn to remove weights group by group?

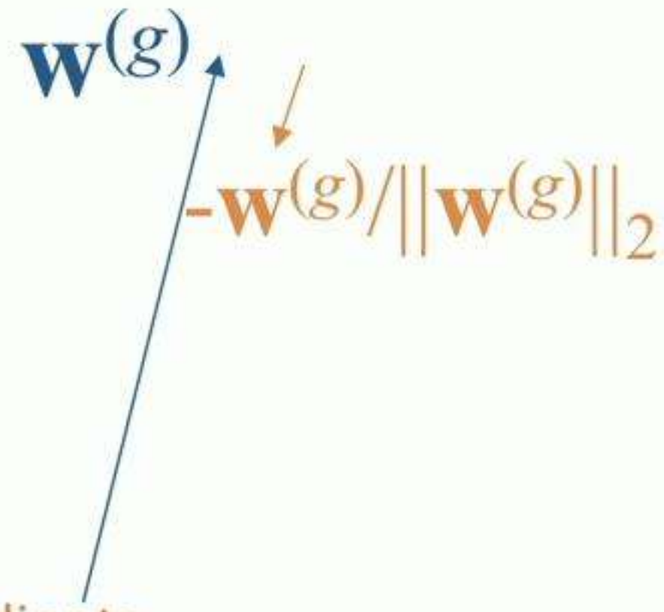
The gradient-based explanation:

$$\mathbf{w}^{(g)} \leftarrow \mathbf{w}^{(g)} - \eta \left(\frac{\partial E_D(\mathbf{w})}{\partial \mathbf{w}^{(g)}} + \lambda_g \frac{\mathbf{w}^{(g)}}{\|\mathbf{w}^{(g)}\|_2} \right)$$

A group of weights

Regular gradients to minimize error

Additional gradients to learn sparsity



Structured Sparsity Learning (*SSL*)

- Why can SSL learn to remove weights group by group?

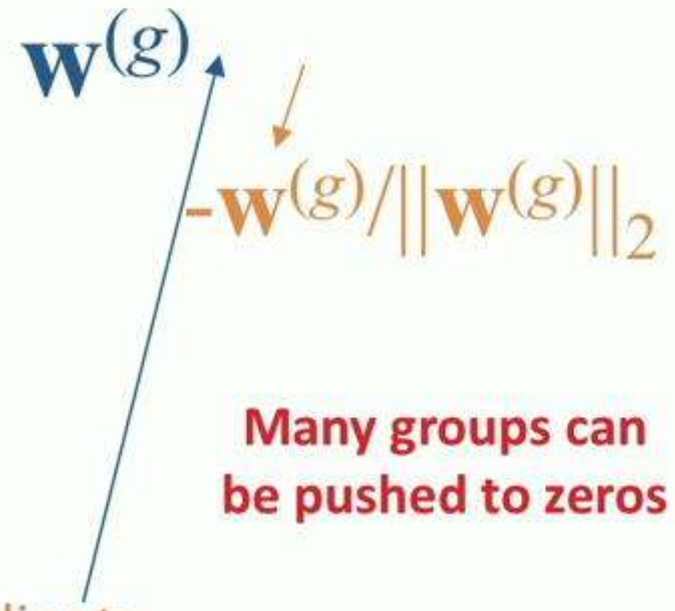
The gradient-based explanation:

$$\mathbf{w}^{(g)} \leftarrow \mathbf{w}^{(g)} - \eta \left(\frac{\partial E_D(\mathbf{w})}{\partial \mathbf{w}^{(g)}} + \lambda_g \frac{\mathbf{w}^{(g)}}{\|\mathbf{w}^{(g)}\|_2} \right)$$

A group of weights

Regular gradients to minimize error

Additional gradients to learn sparsity

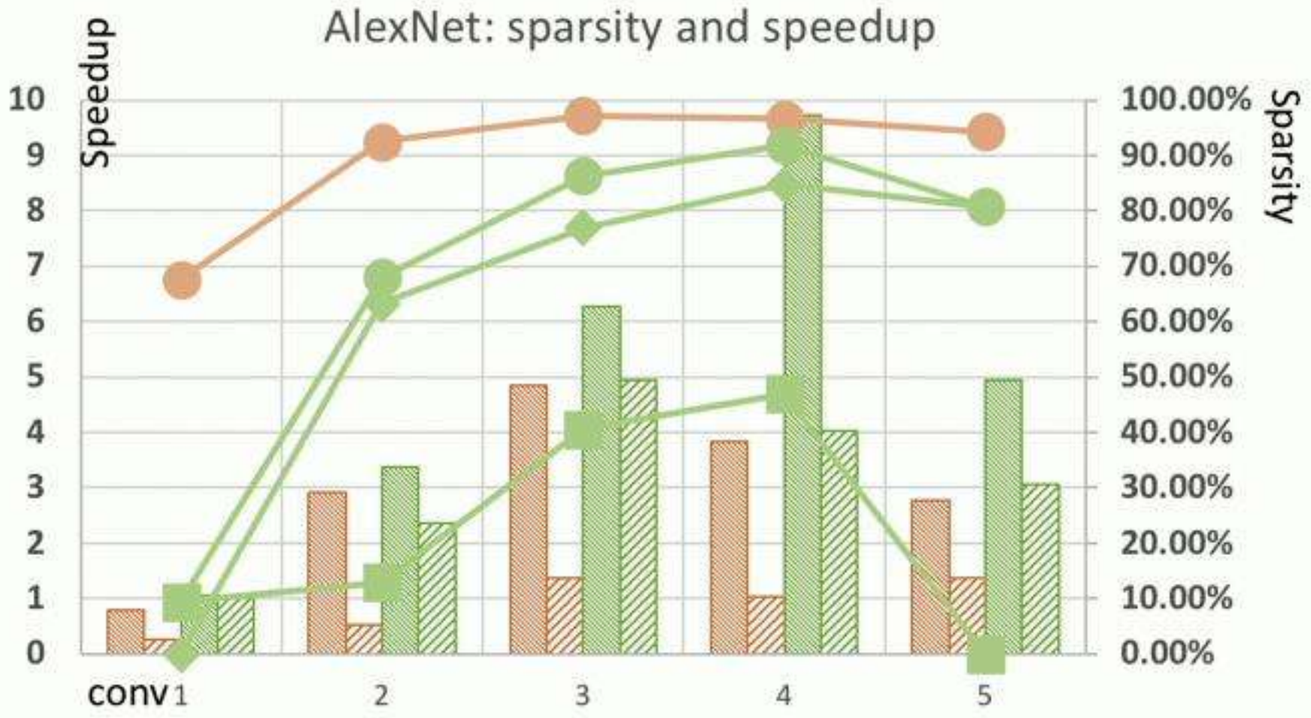


Many groups can be pushed to zeros

Structurally Sparse *AlexNet*

Structured Sparsity (removing columns and rows):

Non-structured sparsity



Structured Sparsity Learning (*SSL*)

- Why can SSL learn to remove weights group by group?

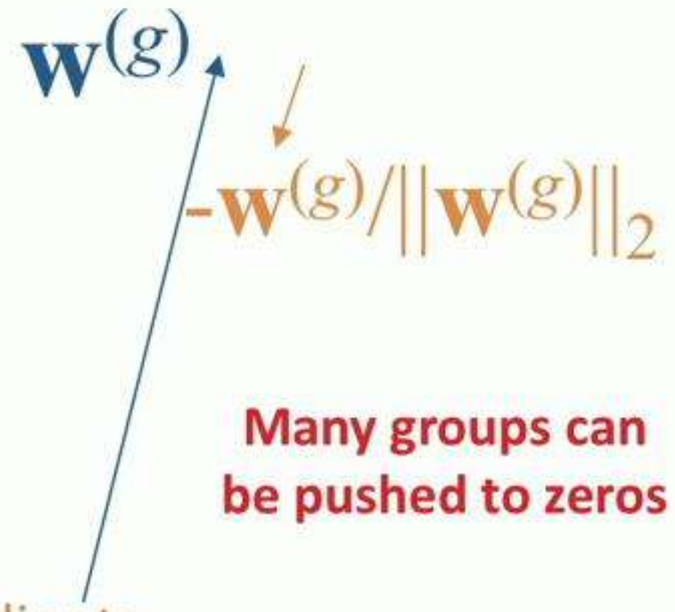
The gradient-based explanation:

$$\mathbf{w}^{(g)} \leftarrow \mathbf{w}^{(g)} - \eta \left(\frac{\partial E_D(\mathbf{w})}{\partial \mathbf{w}^{(g)}} + \lambda_g \frac{\mathbf{w}^{(g)}}{\|\mathbf{w}^{(g)}\|_2} \right)$$

A group of weights

Regular gradients to minimize error

Additional gradients to learn sparsity



Many groups can be pushed to zeros

Structurally Sparse *AlexNet*

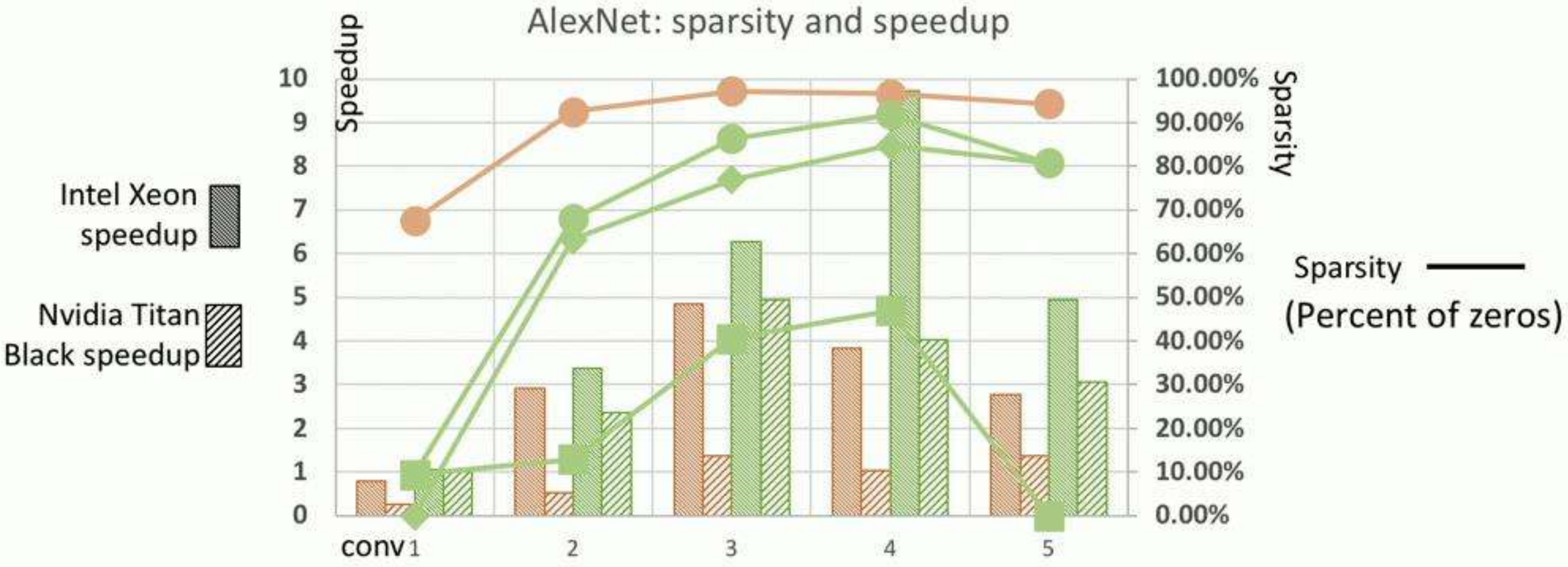
Structured Sparsity (removing columns and rows):

Non-structured sparsity

Structurally Sparse *AlexNet*

Structured Sparsity (removing columns and rows):

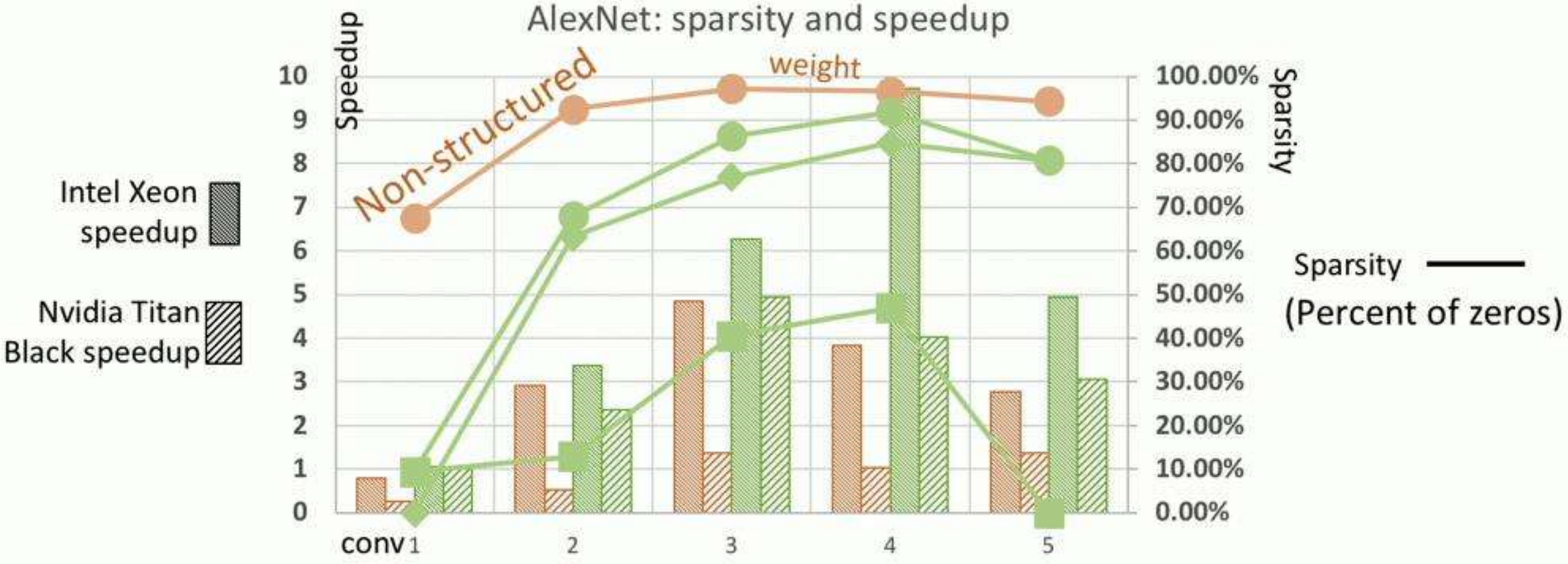
Non-structured sparsity



Structurally Sparse *AlexNet*

Structured Sparsity (removing columns and rows):

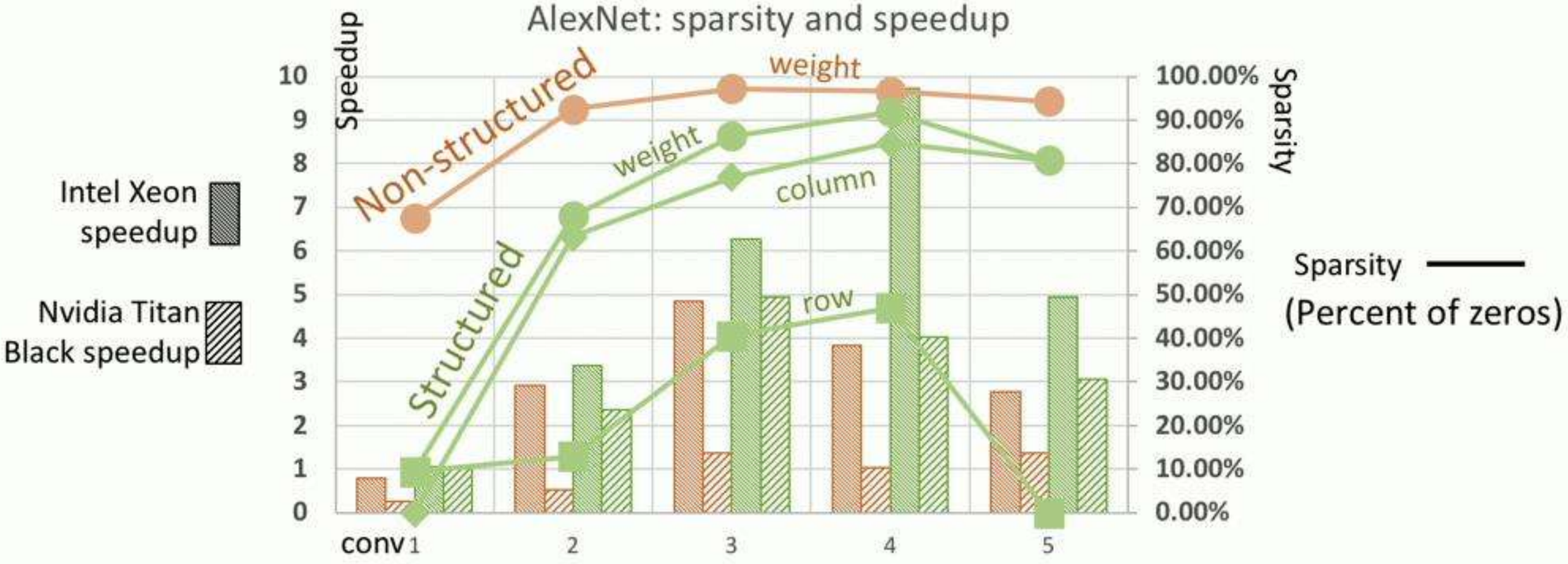
Non-structured sparsity



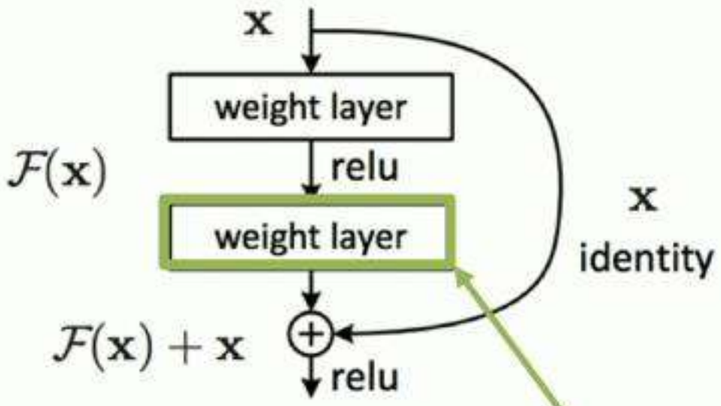
Structurally Sparse *AlexNet*

Structured Sparsity (removing columns and rows):

Non-structured sparsity



Structurally Sparse *ResNets* – Removing Layers

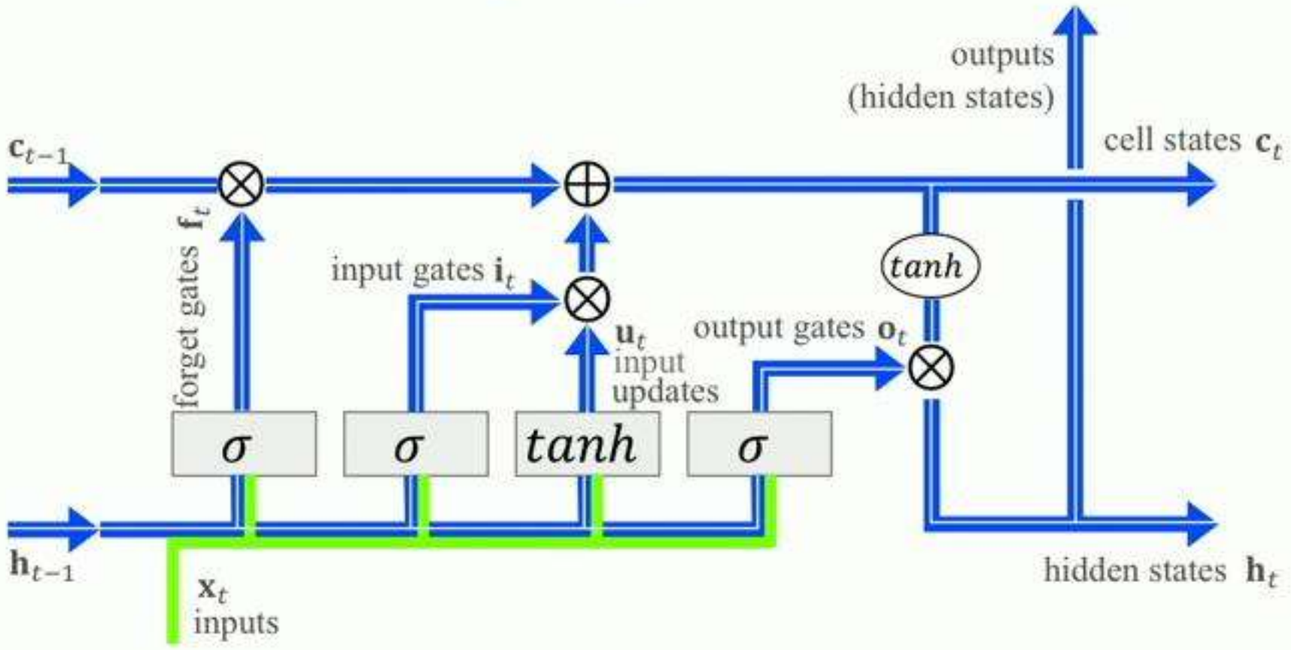


K. He et al. CVPR 2016.

One layer is one group.

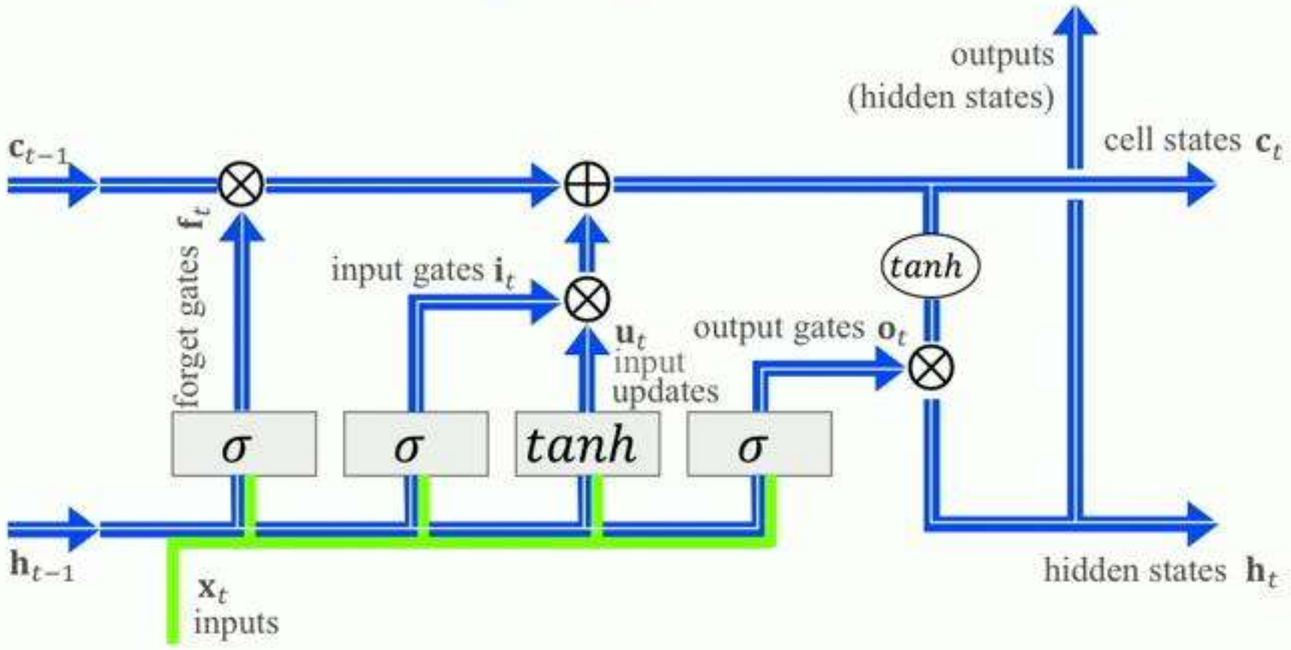
	# layers	error
ResNet	20	8.82%
SSL-ResNet	14	8.54%
Reduced	**6/20**	
ResNet	32	7.51%
SSL-ResNet	18	7.40%
Reduced	**14/32**	

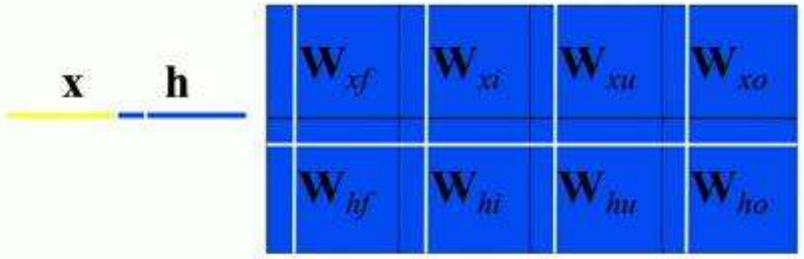
Structurally Sparse LSTMs: Removing Hidden Structures



$$\mathbf{i}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xi} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{b}_i\right)$$
$$\mathbf{f}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xf} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{b}_f\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xo} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{ho} + \mathbf{b}_o\right)$$
$$\mathbf{u}_t = tanh\left(\mathbf{x}_t \cdot \mathbf{W}_{xu} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hu} + \mathbf{b}_u\right)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot tanh\left(\mathbf{c}_t\right)$$

Structurally Sparse LSTMs: Removing Hidden Structures





Weights matrices in LSTM

Weights in next layer(s)

$$\mathbf{i}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xi} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{b}_i\right)$$
$$\mathbf{f}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xf} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{b}_f\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xo} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{ho} + \mathbf{b}_o\right)$$
$$\mathbf{u}_t = tanh\left(\mathbf{x}_t \cdot \mathbf{W}_{xu} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hu} + \mathbf{b}_u\right)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot tanh\left(\mathbf{c}_t\right)$$

White strips = *one* group of weights in SSL

Removing one group = reducing hidden size by one

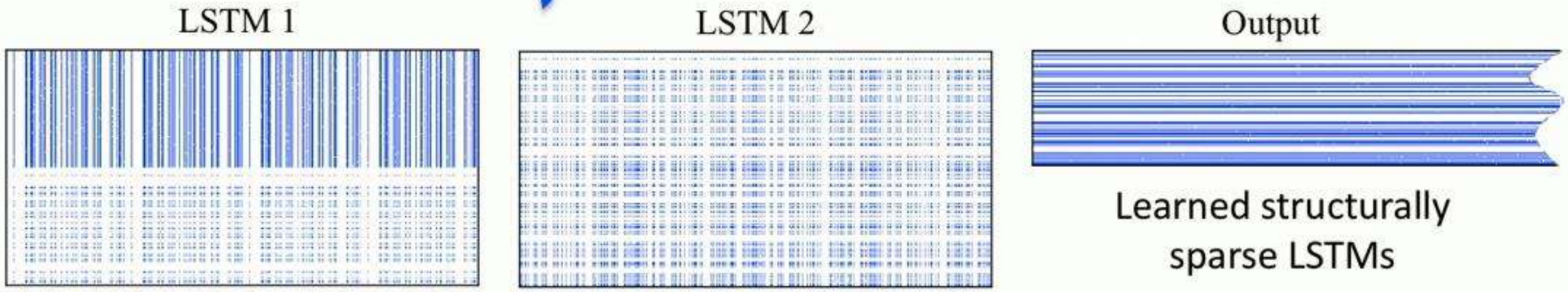
Structurally Sparse *LSTMs*

Method	Test perplexity	Hidden size of (1st, 2nd) LSTMs	Real-time speedup
Zaremba et. al. 2014	78.57	(1500, 1500)	1.00X
Ours (sparsifying Zaremba et. al. 2014)	78.64	(373, 315)	10.59X
	76.03	(381, 535)	7.10X
Train a smaller one	85.66	(373, 315)	10.59X

Structurally Sparse *LSTMs*

Method	Test perplexity	Hidden size of (1st, 2nd) LSTMs	Real-time speedup
Zaremba et. al. 2014	78.57	(1500, 1500)	1.00X
Ours (sparsifying Zaremba et. al. 2014)	78.64	(373, 315)	10.59X
	76.03	(381, 535)	7.10X
Train a smaller one	85.66	(373, 315)	10.59X

1. Reduce size while maintaining perplexity

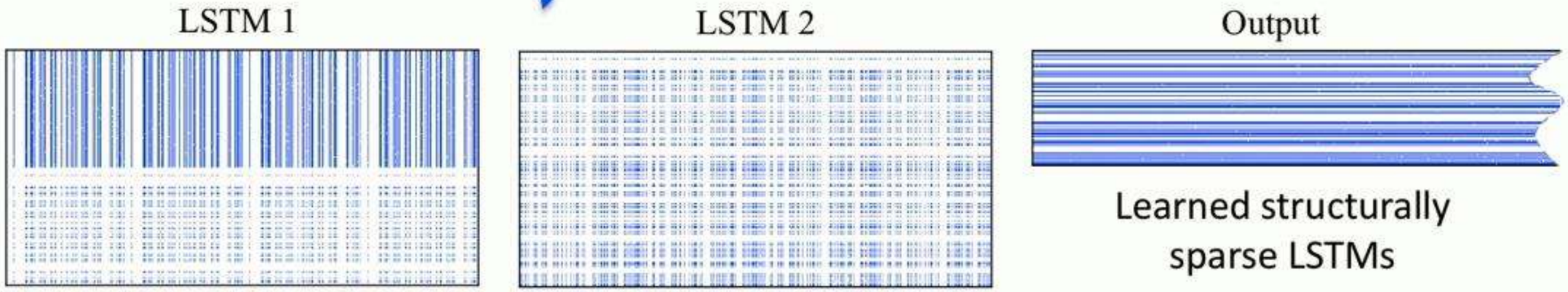


Learned structurally sparse LSTMs

Structurally Sparse *LSTMs*

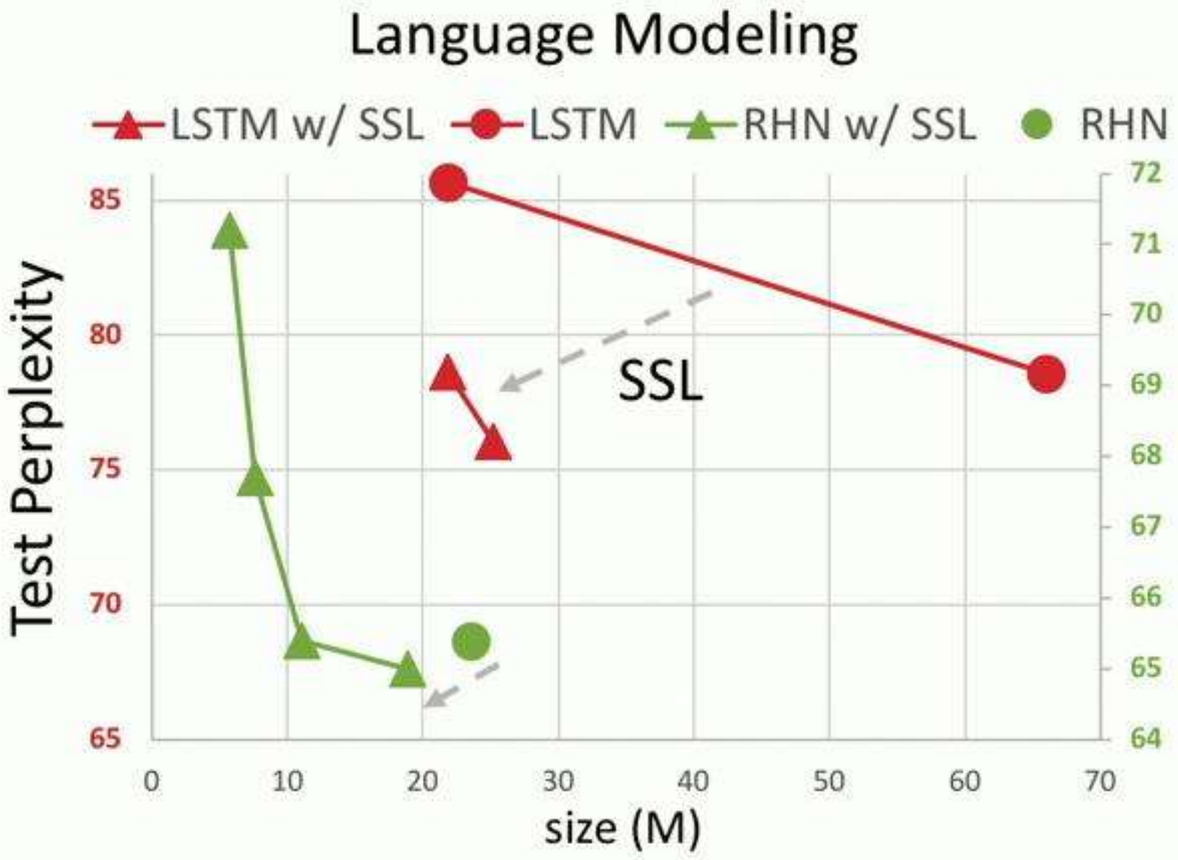
Method	Test perplexity	Hidden size of (1st, 2nd) LSTMs	Real-time speedup
Zaremba et. al. 2014	78.57	(1500, 1500)	1.00X
Ours (sparsifying Zaremba et. al. 2014)	78.64	(373, 315)	10.59X
	76.03	(381, 535)	7.10X
Train a smaller one	85.66	(373, 315)	10.59X

1. Reduce size while maintaining perplexity
2. Achieve lower perplexity
3. Better than training a smaller one



Learned structurally sparse LSTMs

Generalization vs. Model Complexity under SSL



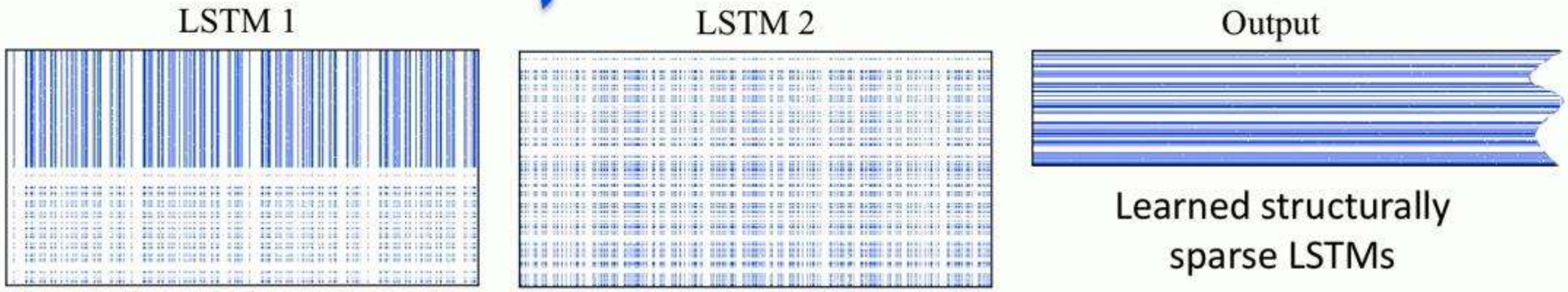
☺ Start from redundant models and sparsify by SSL

RHN: Zilly et al. 2017. BiDAF: Seo et al. 2016

Structurally Sparse *LSTMs*

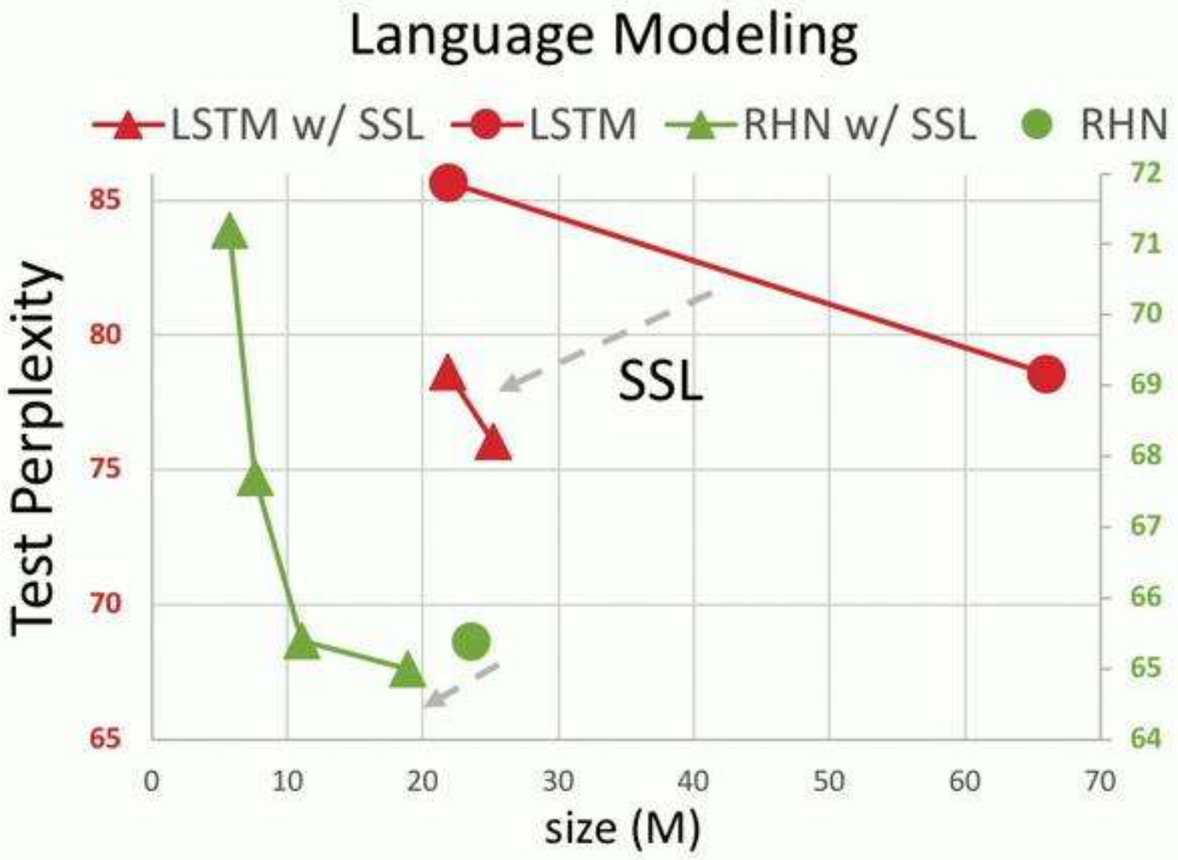
Method	Test perplexity	Hidden size of (1st, 2nd) LSTMs	Real-time speedup
Zaremba et. al. 2014	78.57	(1500, 1500)	1.00X
Ours (sparsifying Zaremba et. al. 2014)	78.64	(373, 315)	10.59X
	76.03	(381, 535)	7.10X
Train a smaller one	85.66	(373, 315)	10.59X

1. Reduce size while maintaining perplexity
2. Achieve lower perplexity
3. Better than training a smaller one



Learned structurally sparse LSTMs

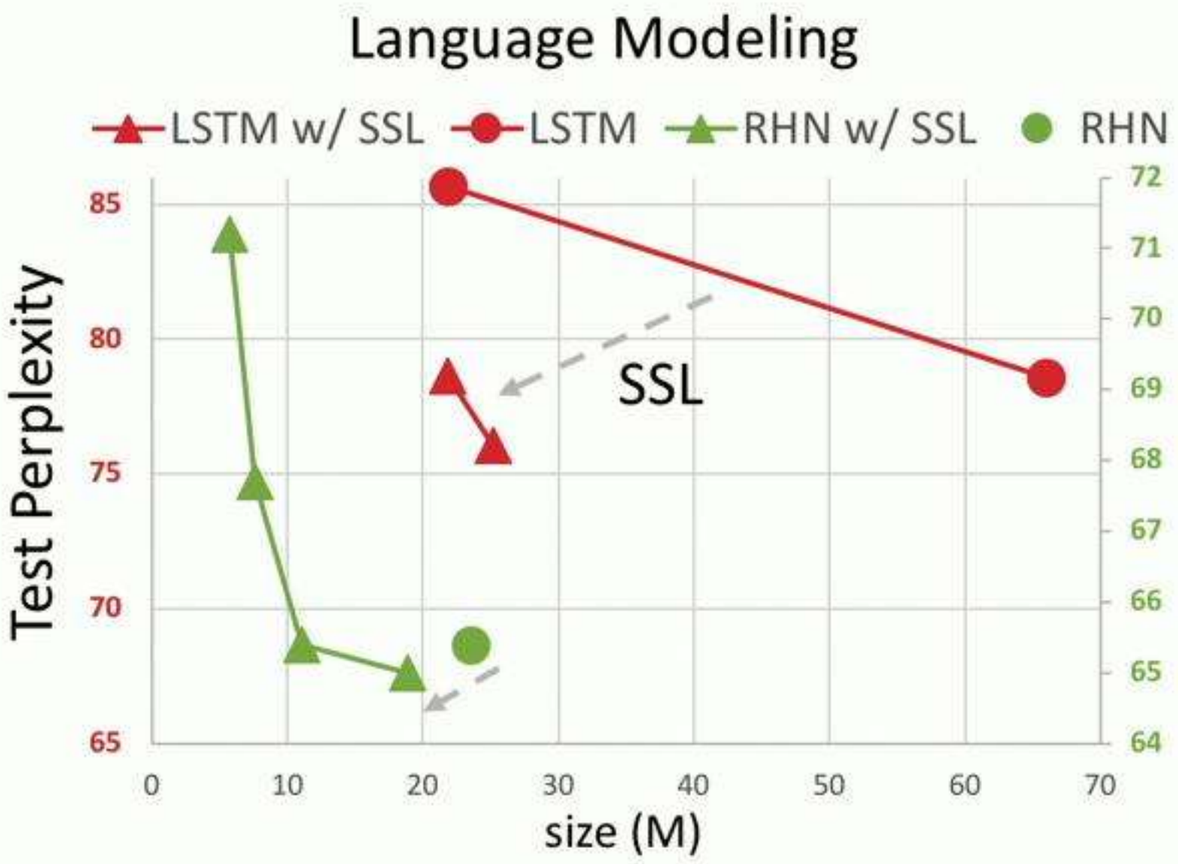
Generalization vs. Model Complexity under SSL

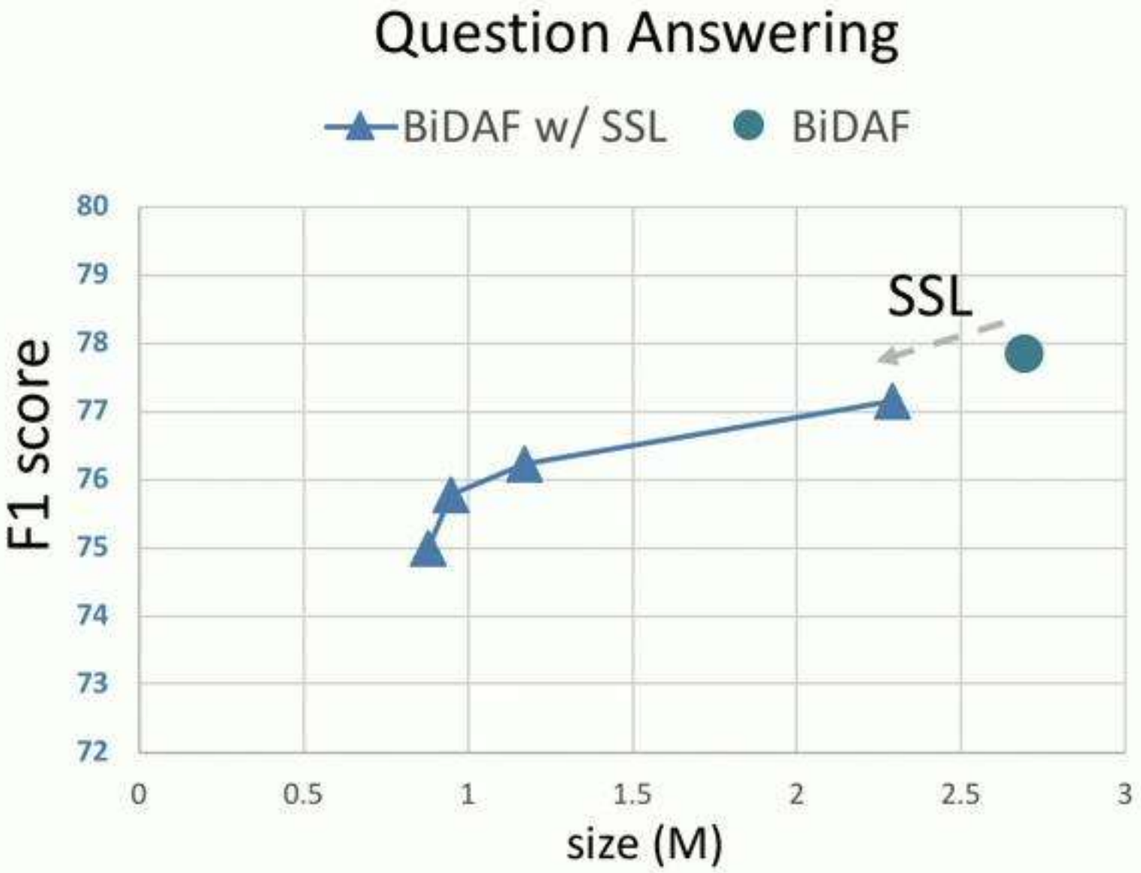


☺ Start from redundant models and sparsify by SSL

RHN: Zilly et al. 2017. BiDAF: Seo et al. 2016

Generalization vs. Model Complexity under SSL





☺ Start from redundant models and sparsify by SSL

☺ Trade-off for compact models with SSL

RHN: Zilly et al. 2017. BiDAF: Seo et al. 2016

Summary of Previous Research

- More scalable distributed deep learning with stochastic low-precision gradients
  - *Effectiveness verified by AI productions*
  - *Can reduce research and production cycle by reducing evaluation/training time of each new model*
  - *More scalable to train larger models*
- Sparse deep neural networks
  - *Enable more ubiquitous AI on edge devices (such as mobile phones, drones, self-driving cars, VR devices, etc)*
  - *Performance gain by starting from a large model and sparsify it down*

Future Research

- Connections to previous research and going beyond
- Scaling Natural Language Processing
- Automated machine learning

SSL Implications and Beyond

Lottery Ticket Hypothesis

(Frankle et al 2019)

"dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation— reach test accuracy comparable to the original network in a similar number of iterations. "

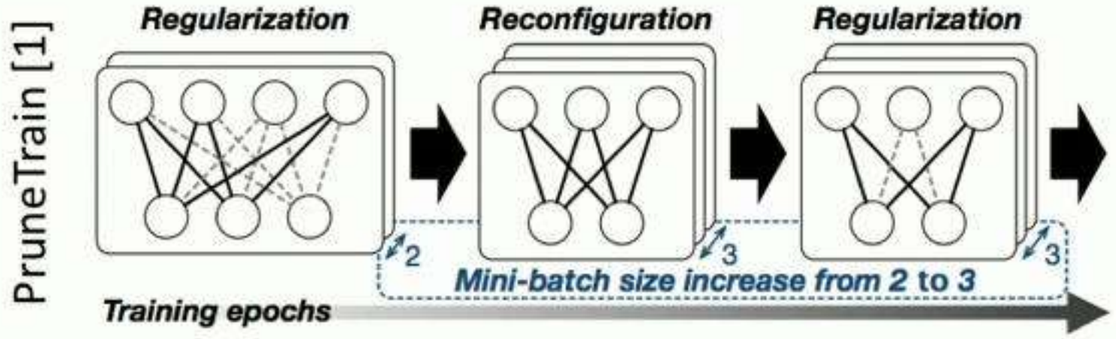
SSL identifies winning tickets in the early stage?

SSL Implications and Beyond

Lottery Ticket Hypothesis
(Frankle et al 2019)

"dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation— reach test accuracy comparable to the original network in a similar number of iterations. "

SSL identifies winning tickets in the early stage?



39% training time reduction of ResNet50 for ImageNet.

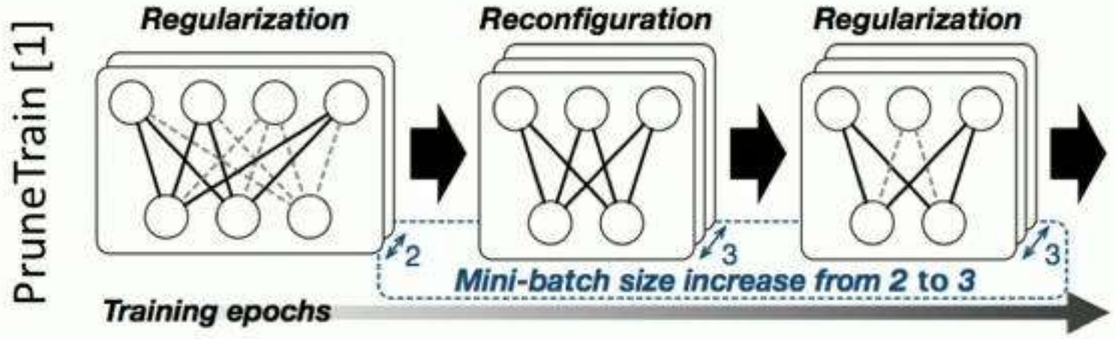
[1] Lym, S, E Choukse, S Zangeneh, **W Wen**, M Erez, and S Shanghavi. "PruneTrain: Gradual Structured Pruning from Scratch for Faster Neural Network Training."*International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2019).
Best Student Paper Finalist.

SSL Implications and Beyond

Lottery Ticket Hypothesis
(Frankle et al 2019)

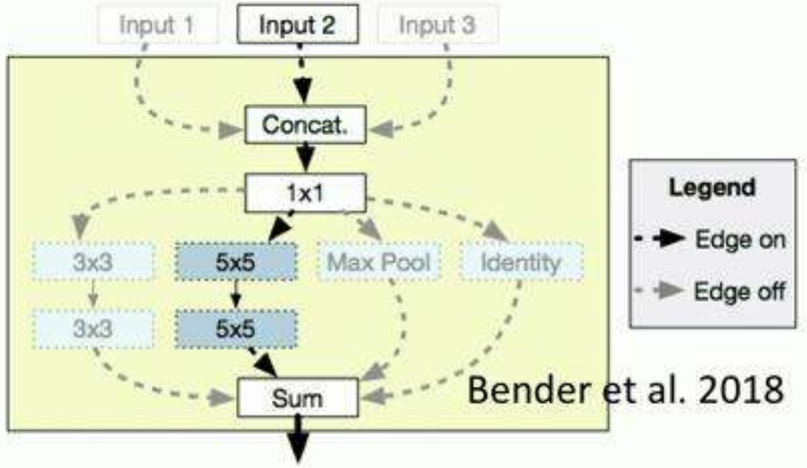
"dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation— reach test accuracy comparable to the original network in a similar number of iterations. "

SSL identifies winning tickets in the early stage?



39% training time reduction of ResNet50 for ImageNet.

AutoML with weight sharing



Bender et al. 2018

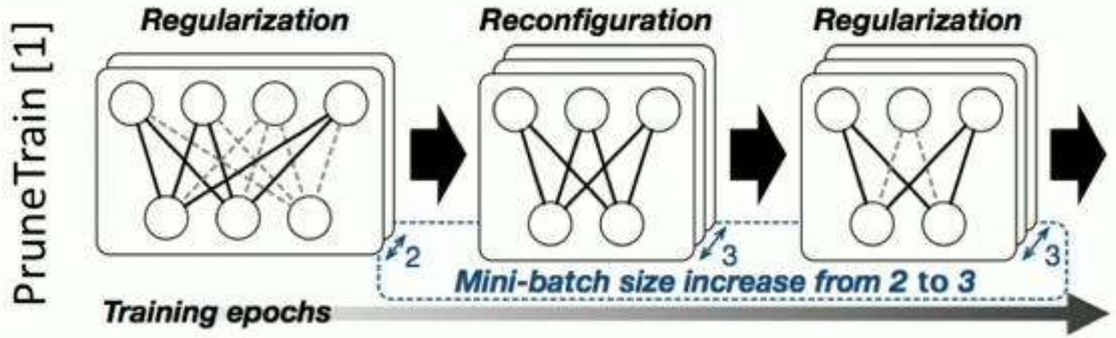
[1] Lym, S, E Choukse, S Zangeneh, **W Wen**, M Erez, and S Shanghavi. "PruneTrain: Gradual Structured Pruning from Scratch for Faster Neural Network Training."*International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2019).
Best Student Paper Finalist.

SSL Implications and Beyond

Lottery Ticket Hypothesis
(Frankle et al 2019)

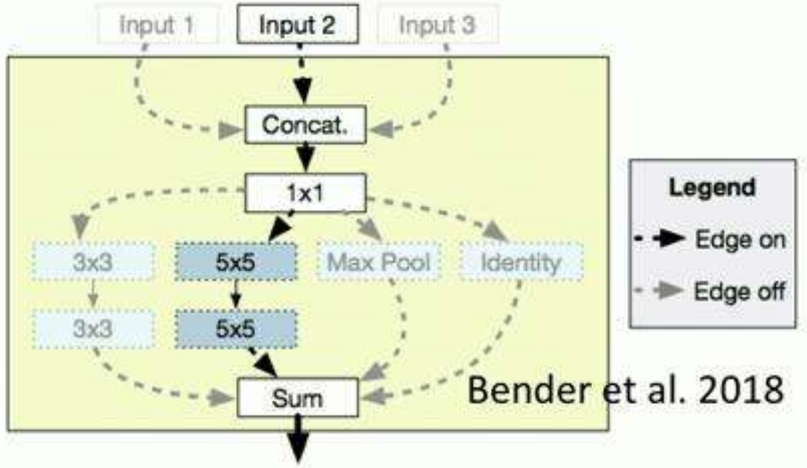
"dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation— reach test accuracy comparable to the original network in a similar number of iterations. "

SSL identifies winning tickets in the early stage?



39% training time reduction of ResNet50 for ImageNet.

AutoML with weight sharing



Bender et al. 2018

AutoML by SSL?

SSL can prune unimportant branches.

[1] Lym, S, E Choukse, S Zangeneh, **W Wen**, M Erez, and S Shanghavi. "PruneTrain: Gradual Structured Pruning from Scratch for Faster Neural Network Training."*International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2019).

Best Student Paper Finalist.

Scaling Natural Language Processing

#L: layer, #H: hidden size, #A: attention head

Devlin et al. 2018

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Scaling Natural Language Processing

#L: layer, #H: hidden size, #A: attention head

Devlin et al. 2018

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

- Scaling NLP models always gives us performance gain
- VggNet-era NLP
  - Hard to further scale up because of computation/memory cost
  - We need more scalable training methods and models

Scaling Natural Language Processing

#L: layer, #H: hidden size, #A: attention head

Devlin et al. 2018

Hyperparams			Dev Set Accuracy			
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

- Scaling NLP models always gives us performance gain
- VggNet-era NLP
  - Hard to further scale up because of computation/memory cost
  - We need more scalable training methods and models

Direction 1: Design more scalable training methods

- Give an architecture, make its training faster
  - TernGrad, SSL, more?

Direction 2: Design more scalable new models

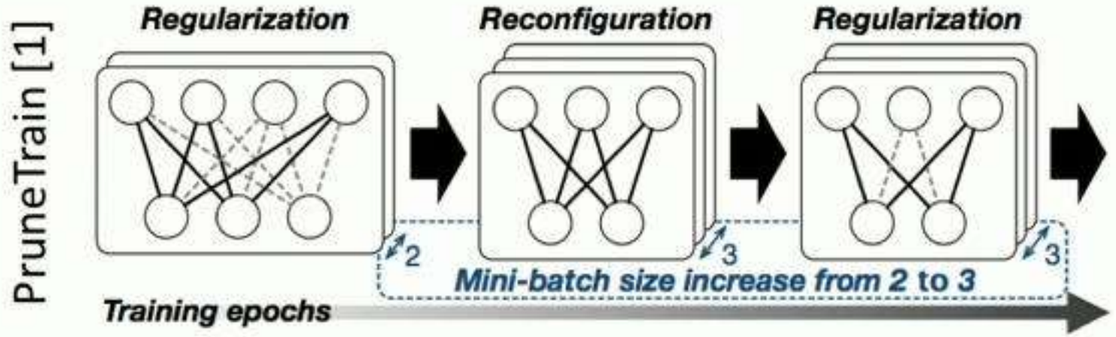
- Design a fast, accurate and cmpact base model and scale up
- Design scalable new models by AutoML

SSL Implications and Beyond

Lottery Ticket Hypothesis
(Frankle et al 2019)

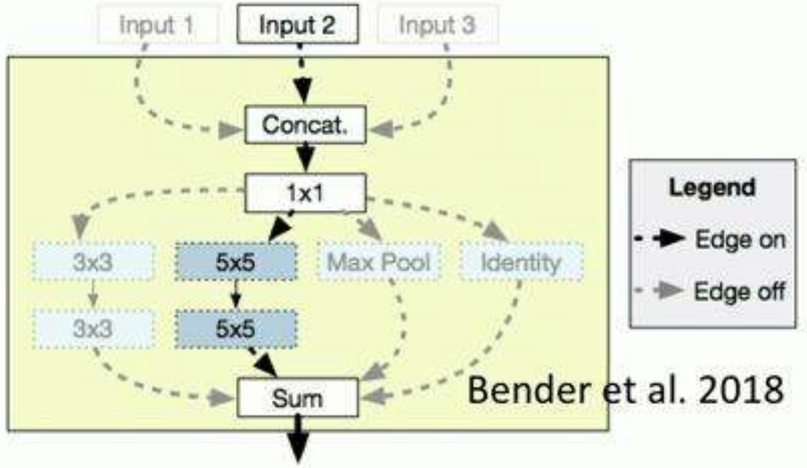
"dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation— reach test accuracy comparable to the original network in a similar number of iterations. "

SSL identifies winning tickets in the early stage?



39% training time reduction of ResNet50 for ImageNet.

AutoML with weight sharing



Bender et al. 2018

AutoML by SSL?

SSL can prune unimportant branches.

[1] Lym, S, E Choukse, S Zangeneh, **W Wen**, M Erez, and S Shanghavi. "PruneTrain: Gradual Structured Pruning from Scratch for Faster Neural Network Training."*International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2019).

Best Student Paper Finalist.

Scaling Natural Language Processing

#L: layer, #H: hidden size, #A: attention head

Devlin et al. 2018

Hyperparams			Dev Set Accuracy			
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

- Scaling NLP models always gives us performance gain
- VggNet-era NLP
  - Hard to further scale up because of computation/memory cost
  - We need more scalable training methods and models

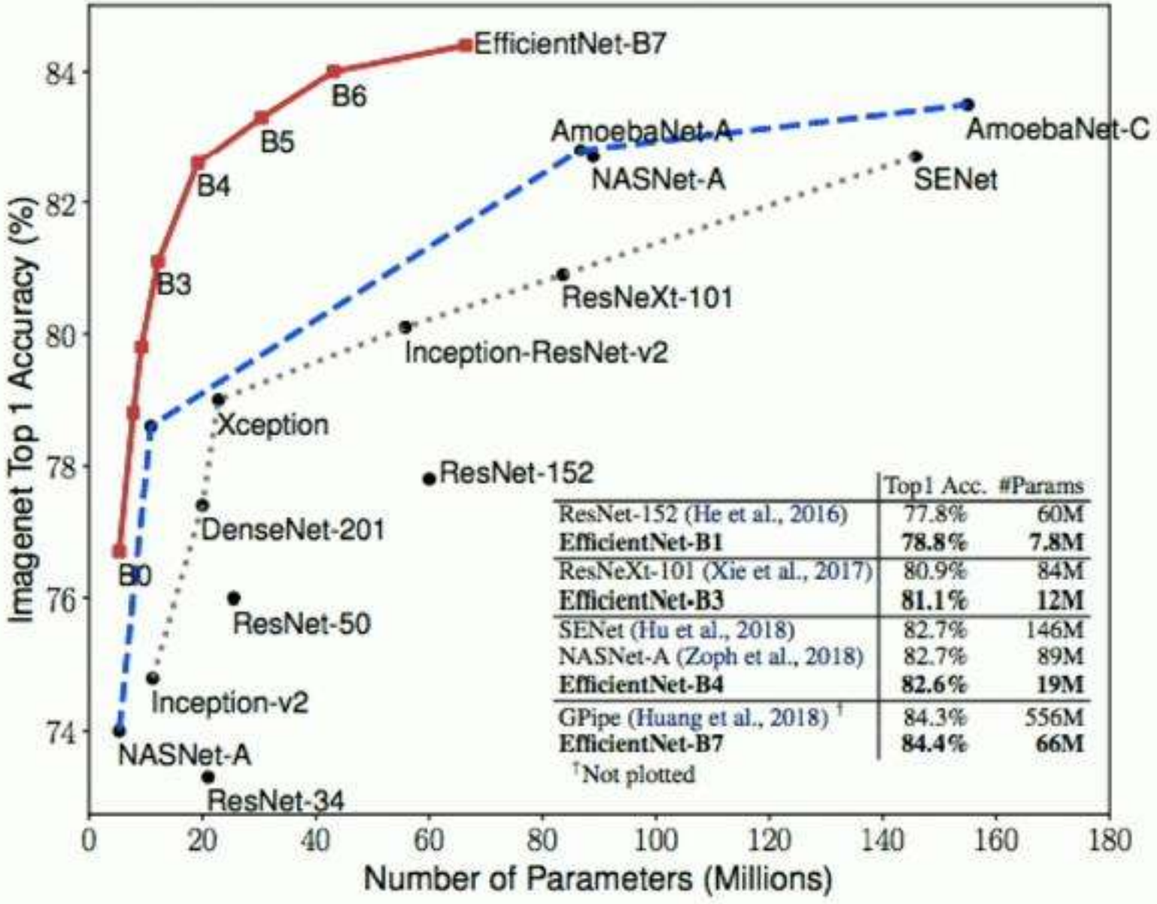
Direction 1: Design more scalable training methods

- Give an architecture, make its training faster
  - TernGrad, SSL, more?

Direction 2: Design more scalable new models

- Design a fast, accurate and cmpact base model and scale up
- Design scalable new models by AutoML

Automated Machine Learning (AutoML)

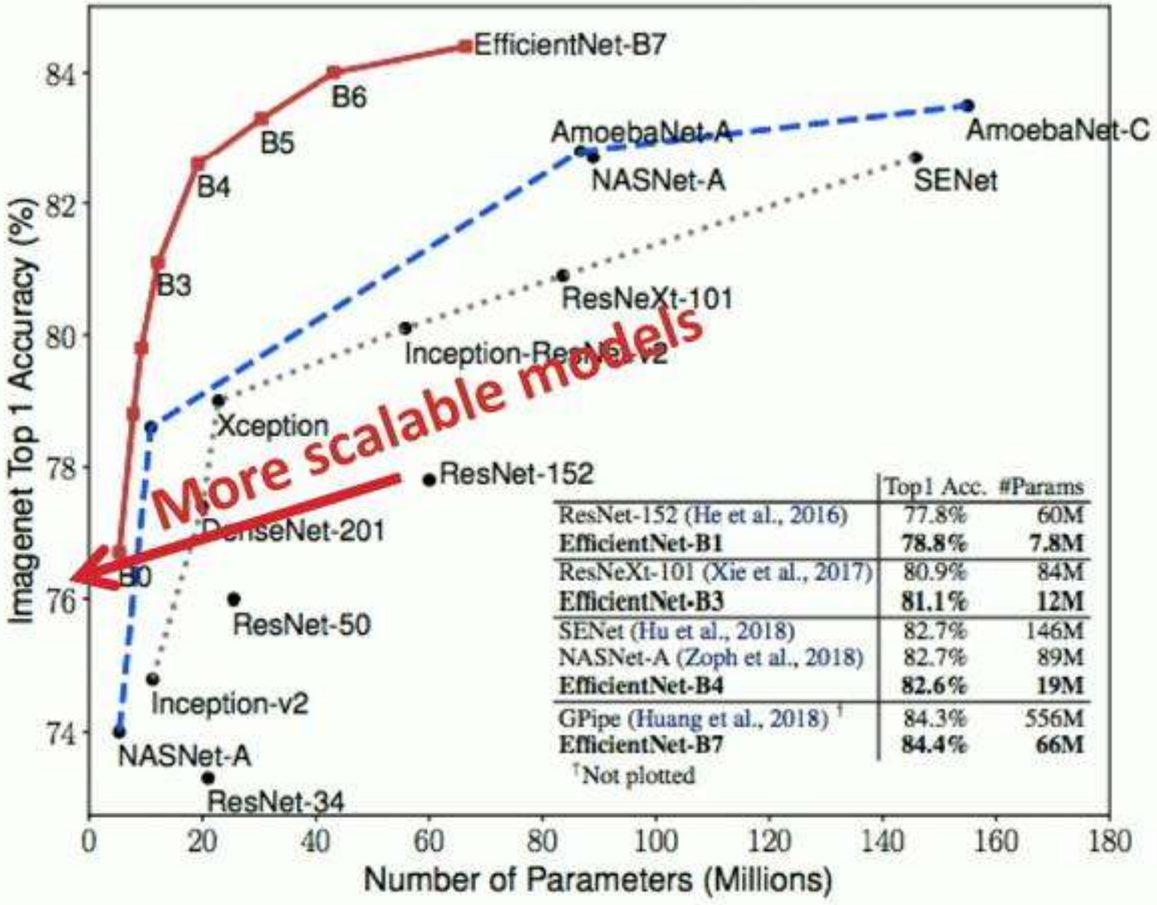


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Tan & Le, 2019

Automated Machine Learning (AutoML)

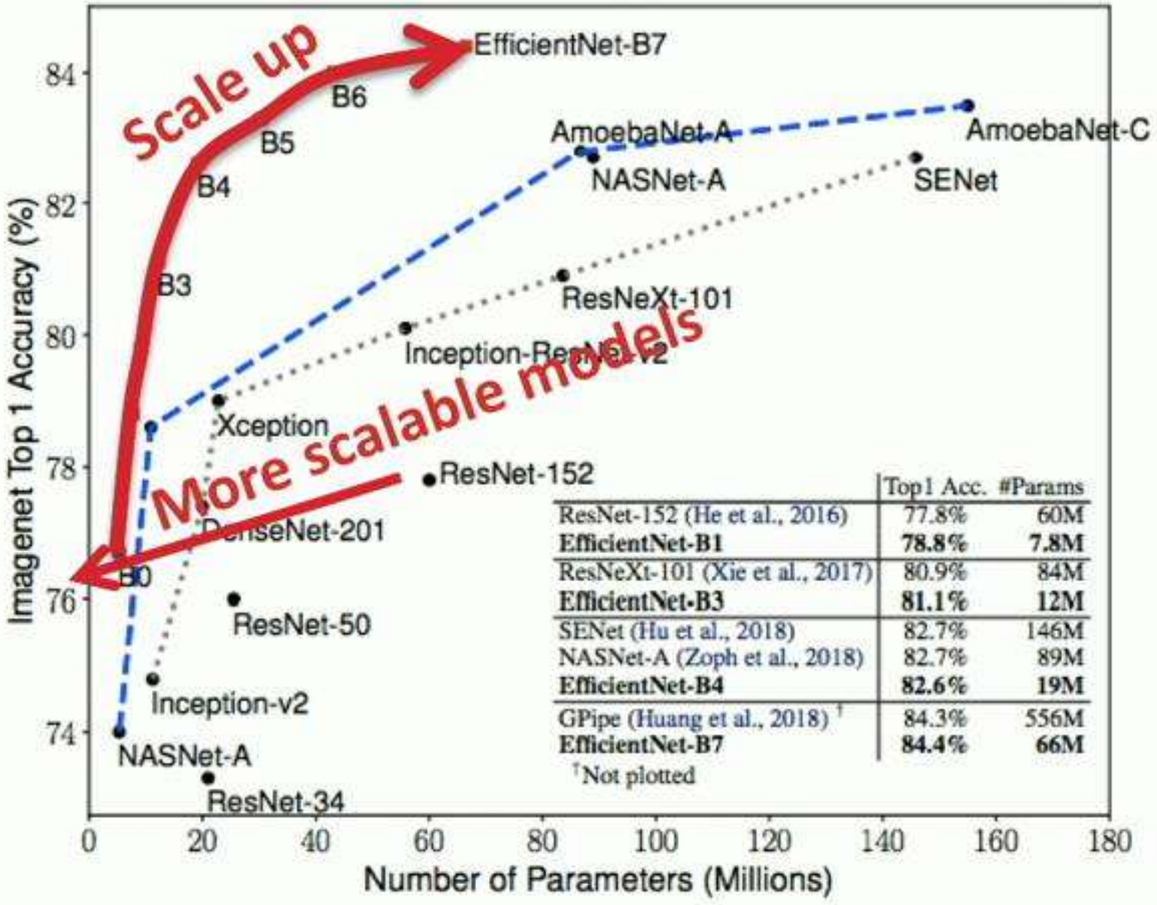


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Tan & Le, 2019

Automated Machine Learning (AutoML)

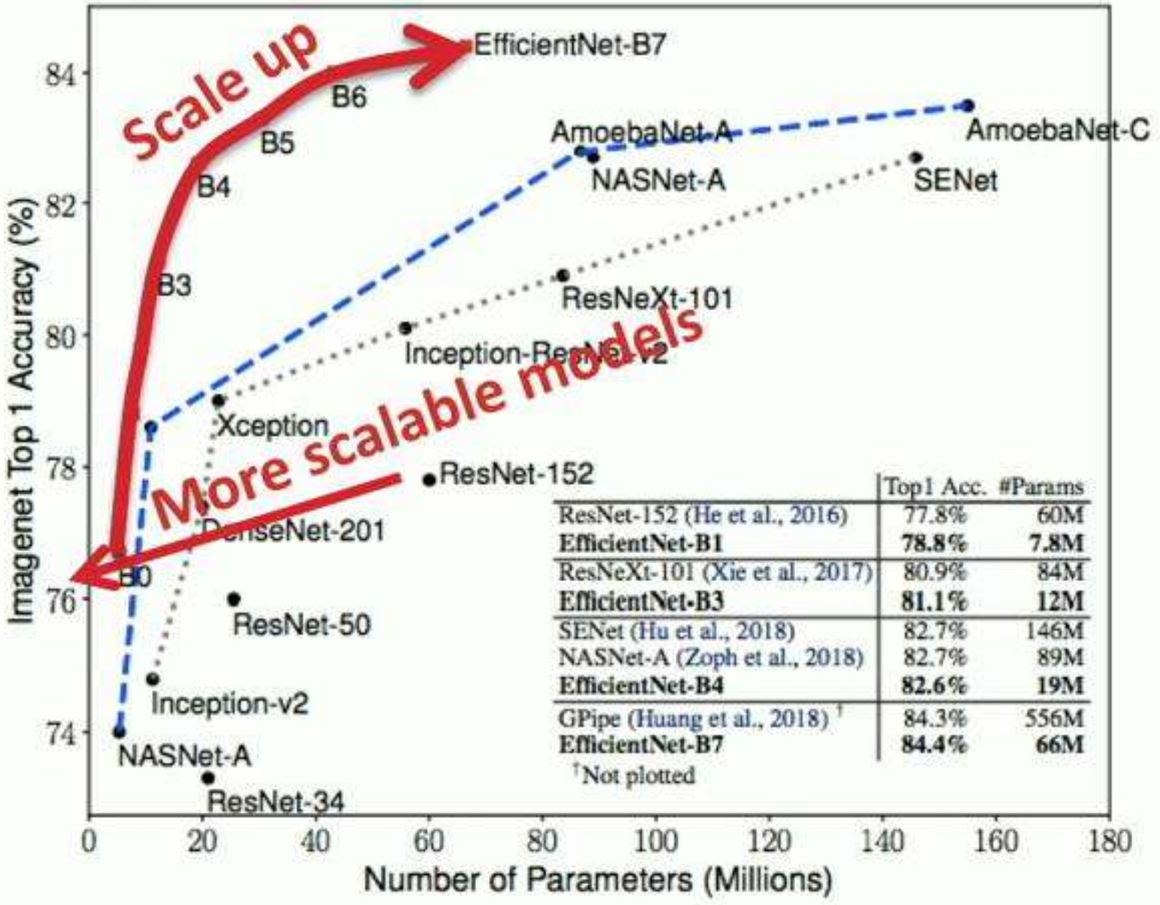


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Tan & Le, 2019

Automated Machine Learning (AutoML)



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

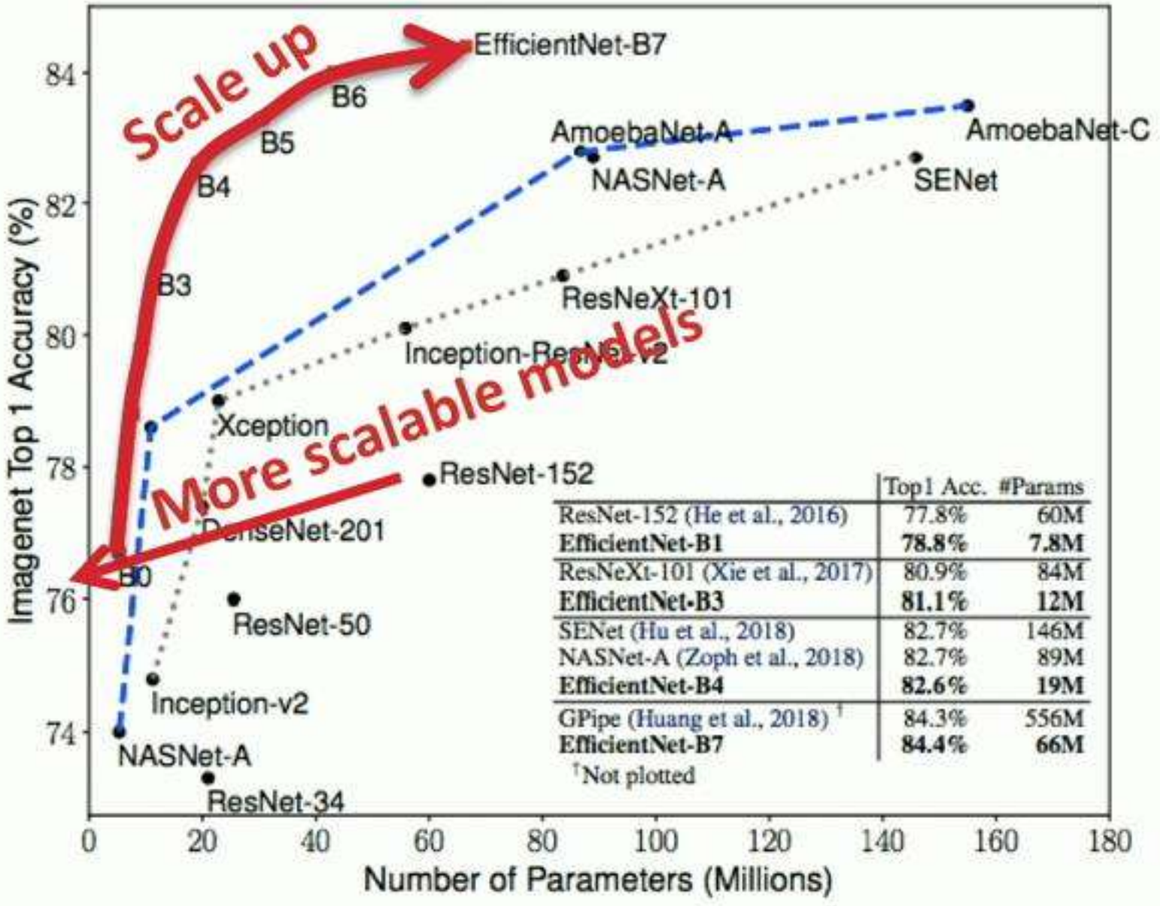
†Not plotted

Tan & Le, 2019

Problems in AutoML:

1. computation/memory intensive
2. Rely on human designed operations and search combination only

Automated Machine Learning (AutoML)



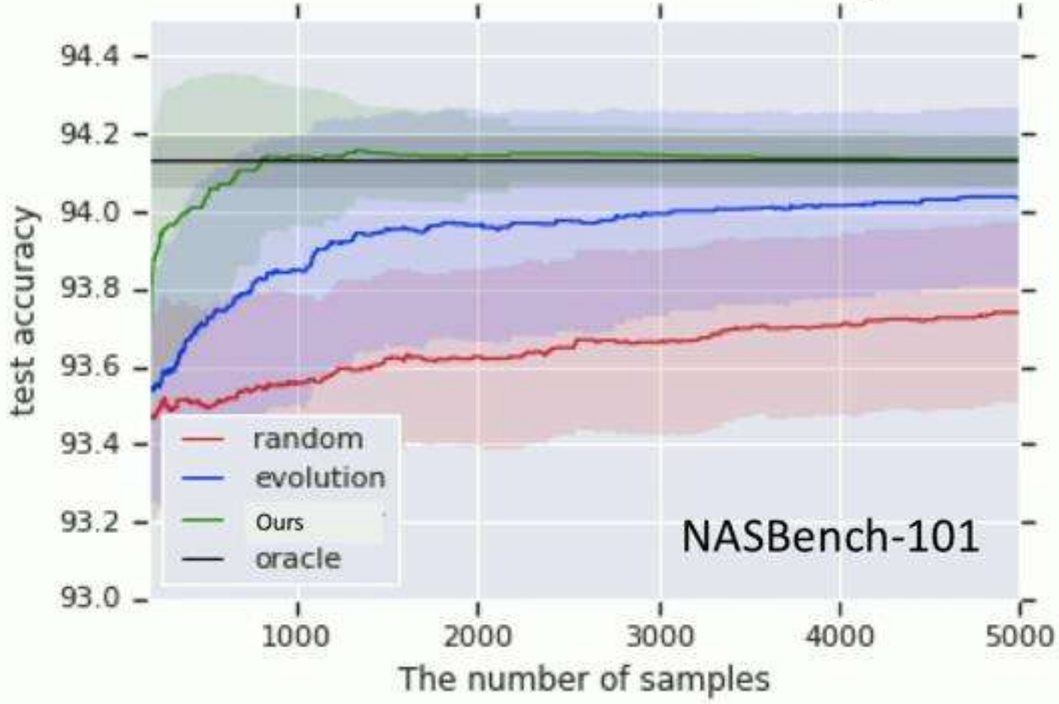
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Tan & Le, 2019

Problems in AutoML:

1. computation/memory intensive
2. Rely on human designed operations and search combination only



References

- Pranav Dar, 25 Open Datasets for Deep Learning Every Data Scientist Must Work With, https://www.analyticsvidhya.com/blog/2018/03/comprehensive-collection-deep-learning-datasets/. 2018
- Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." *arXiv preprint arXiv:1605.07678* (2016).
- Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser et al. "Mastering the game of Go with deep neural networks and tree search." *nature*529, no. 7587 (2016): 484.
- Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks." In *CVPR*, vol. 1, no. 2, p. 3. 2017.
- Dean, Jeffrey, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior et al. "Large scale distributed deep networks." In *Advances in neural information processing systems*, pp. 1223-1231. 2012.
- Goyal, Priya, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. "Accurate, large minibatch SGD: training imagenet in 1 hour." *arXiv preprint arXiv:1706.02677* (2017).

THANKS! Q&A?

Wei Wen, Duke University
@wei_wen_
www.pittnuts.com