Deep Learning in Cloud-Edge AI Systems

Wei Wen

COMPUTATIONAL EVOLUTIONARY INTELLIGENCE LAB

Electrical and Computing Engineering Department

Duke University

www.pittnuts.com

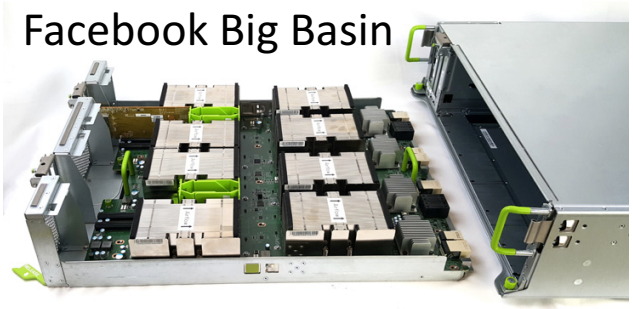
AI Systems Landing

Into the Cloud:

Microsoft Azure

amazon web services

Facebook Big Basin



AI Systems Landing

Upon the Edge:

Challenges in landing???

The Rocket Analogy of Deep Learning

An analogy by Andrew Ng

Rocket: **Big** Neural Networks
Engine: **Big** Computing
Fuel: **Big** Data

3 BIGs

Deep Learning in the Cloud

Google TPU Cloud

Tons of Cables!!!

Communication Bottleneck!!!

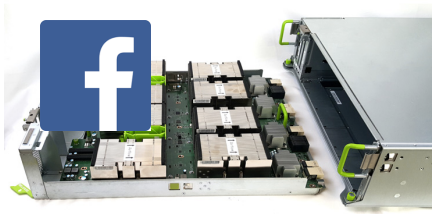
Parallelism

Deep Learning on the Edge

Model Size and Inference Speed Matter!!!

Research Highlights

- Distributed Training in the Cloud
  - ***TernGrad***: Ternary Gradients to Reduce Communication in Distributed Deep Learning, *NIPS 2017 (oral)*
  - On-going work on large-batch training
- Efficient Inference on the Edge
  - ***Structurally Sparse DNNs*** (*NIPS 2016 & ICLR 2018*)
  - Lower-rank DNNs (*ICCV 2017*)
  - A Compact DNN: Approaching GoogLeNet-Level Accuracy of Classification and Domain Adaptation (*CVPR 2017*)
  - Direct Sparse Convolution (*ICLR 2017*)



TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning

NIPS 2017 (Oral)

Wei Wen[1], Cong Xu[2], Feng Yan[3], Chunpeng Wu[1],
Yandan Wang[4], Yiran Chen[1], Hai (Helen) Li[1]

Duke University[1], Hewlett Packard Labs[2],
University of Nevada - Reno[3], University of Pittsburgh[4]

https://github.com/wenwei202/terngrad

Background – Stochastic Gradient Descent

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

Batch gradient descent: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta_t}{n}\sum_{i=1}^{n} g_t^{(i)}$ Computation expensive

$g_t^{(i)} = \nabla Q(\boldsymbol{z}_i, \boldsymbol{w}_t)$

Mini-batch stochastic gradient descent (SGD):

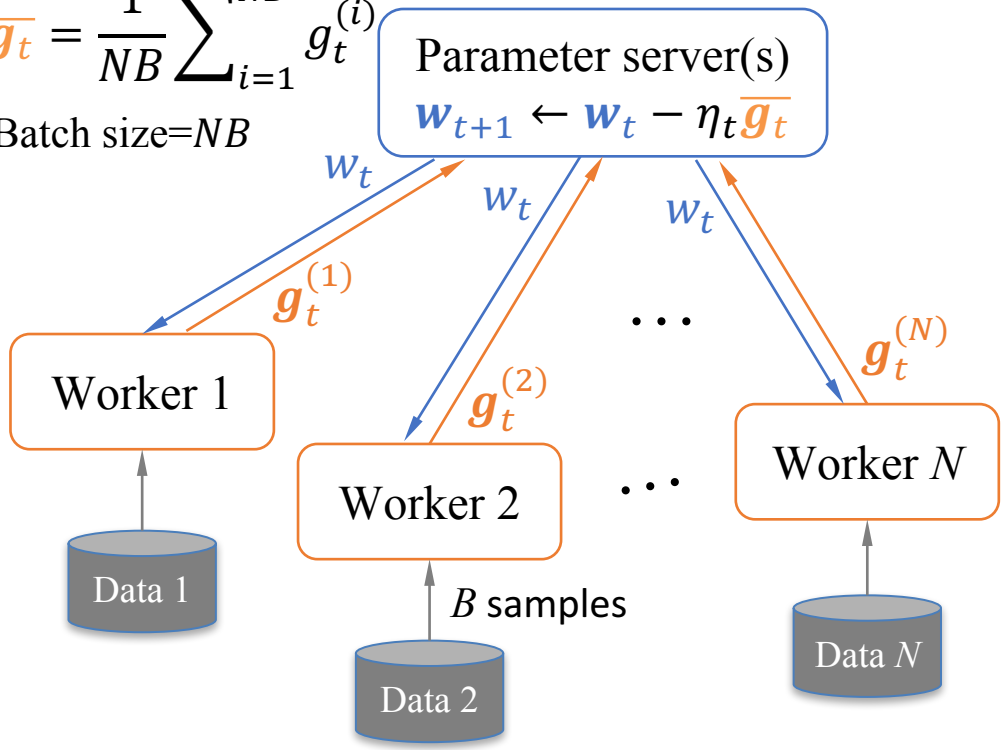
$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta_t}{B}\sum_{b=1}^{B} g_t^{(b)}$ Computation cheap

B<<n samples are randomly drawn from training dataset

Background - Distributed Deep Learning

$$\overline{g_t} = \frac{1}{NB}\sum_{i=1}^{NB} g_t^{(i)}$$

Batch size=NB



Synchronized Data Parallelism for Stochastic Gradient Descent (SGD):

1. Training data is split to *N* subsets
2. Each worker has a model replica (copy)
3. Each replica is trained on a data subset
4. Synchronization in parameter server(s)

Scalability:

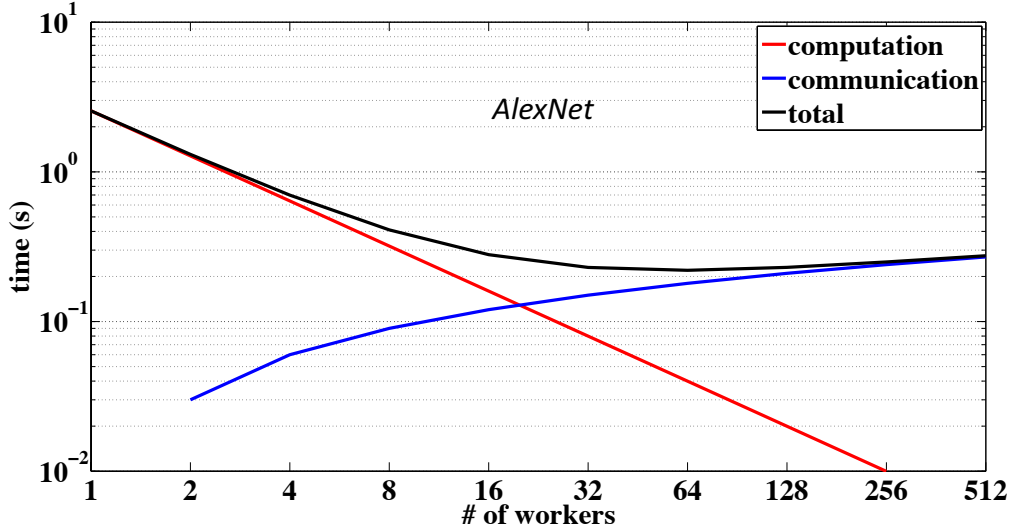
1. Computing time decreases with N
2. Communication can be the bottleneck
3. This work: quantizing gradient to three (i.e., *ternary*) levels {-1, 0, 1} (<2bits)

Communication Bottleneck

$$t_{comp} = \frac{flop}{flops \cdot n}$$

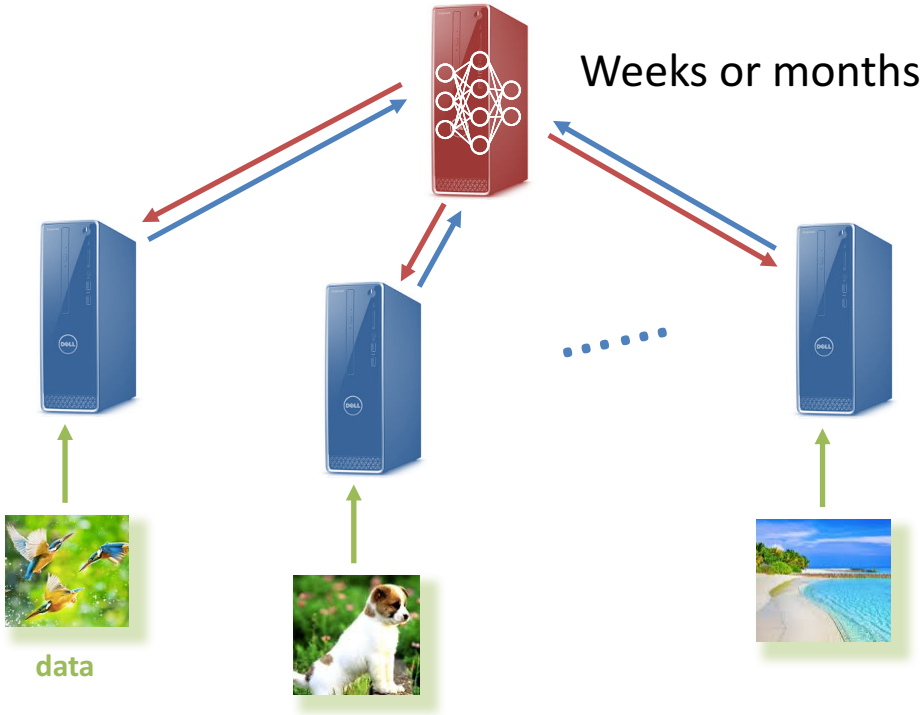
$$t_{comm} = \frac{2 \cdot |\mathbf{W}| \cdot g(n)}{b}$$

$$g(n) = \log_2(n)$$



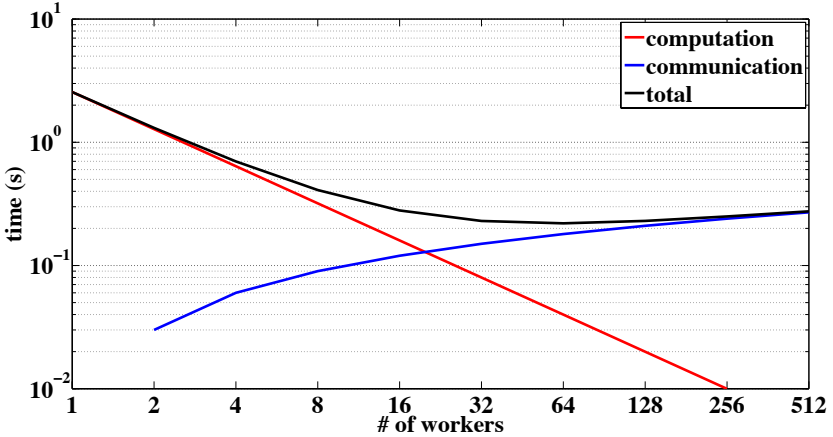
Credit: Alexander Ulanov

Background - Distributed Deep Learning

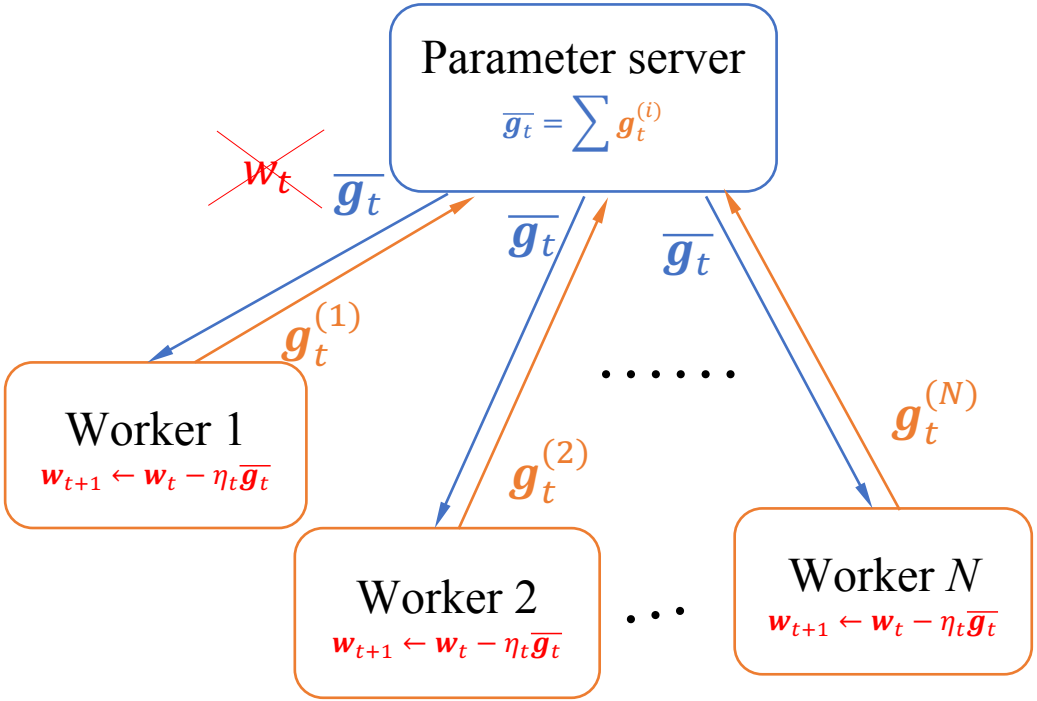


Weeks or months -> hours

Communication Bottleneck!

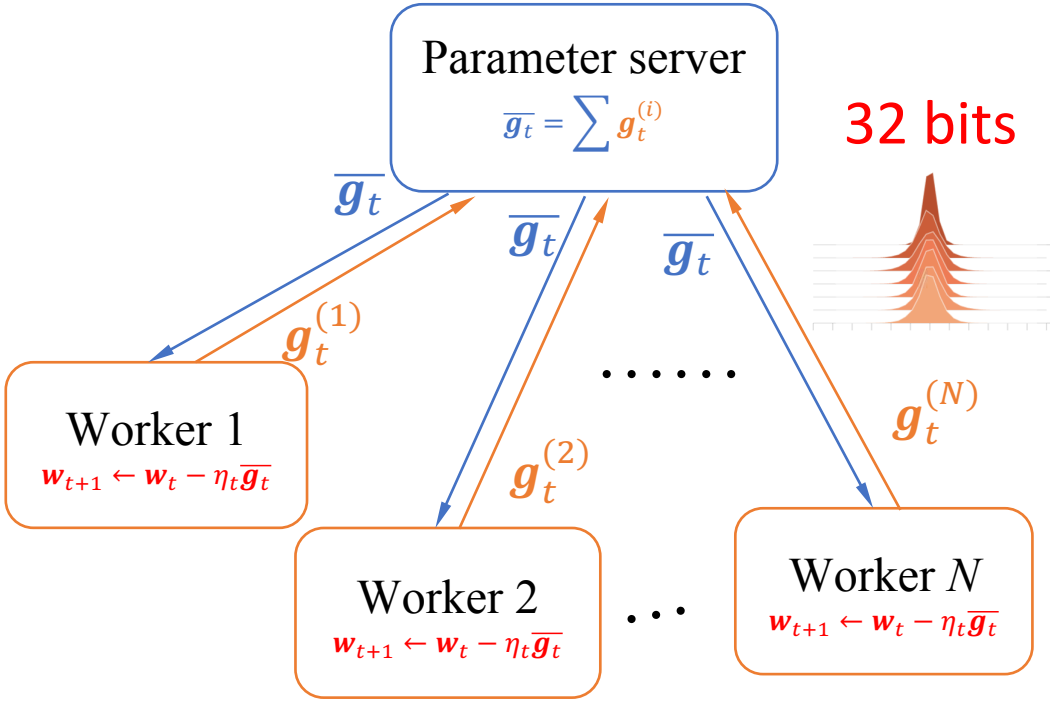


An Alternative Setting



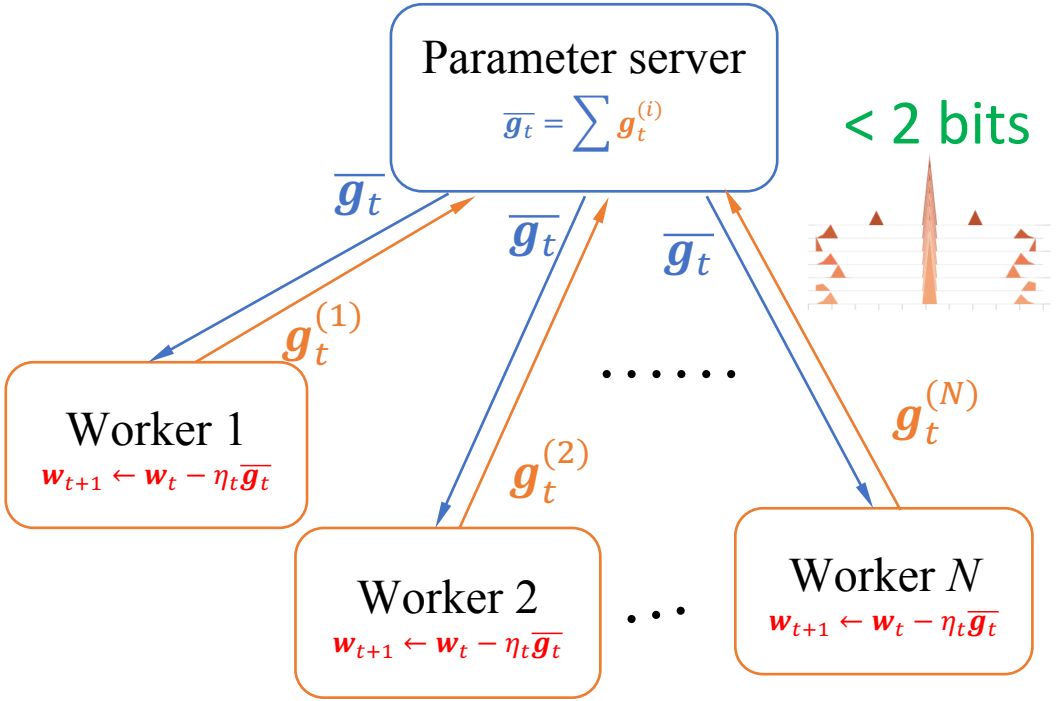
1. Only exchange gradients
2. Gradient quantization can reduce communication in both directions

Gradient Quantization for Communication Reduction



Only exchange quantized gradients

Gradient Quantization for Communication Reduction



Only exchange quantized gradients

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

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

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

No bias

TernGrad is Simple

```
146 def stochastical_binarize_gradients(grads_and_vars, scalers):
147   """Stochastically binarize gradients."""
148   gradients, variables = zip(*grads_and_vars)
149   binarized_gradients = []
150   for gradient, scaler in zip(gradients, scalers):
151     if gradient is None:
152       binarized_gradients.append(None)
153       continue
154     if isinstance(gradient, tf.IndexedSlices):
155       gradient_shape = gradient.dense_shape
156     else:
157       gradient_shape = gradient.get_shape()
158
159     zeros = tf.zeros(gradient_shape)
160     abs_gradient = tf.abs(gradient)
161     sign_gradient = tf.sign( gradient )
162     rnd_sample = tf.random_uniform(gradient_shape,0,scaler)
163     where_cond = tf.less(rnd_sample, abs_gradient)
164     binarized_gradient = tf.cond(tf.size(gradient) < FLAGS.size_to_binarize,
165                                  lambda: gradient,
166                                  lambda: tf.where(where_cond, sign_gradient * scaler, zeros))
```

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):

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

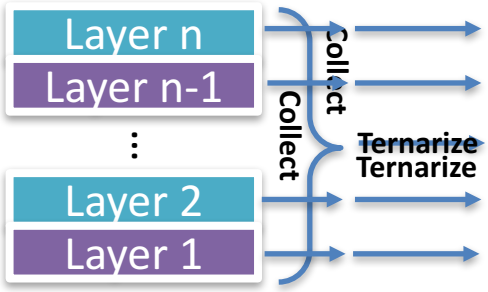
TernGrad almost-truly converges

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

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

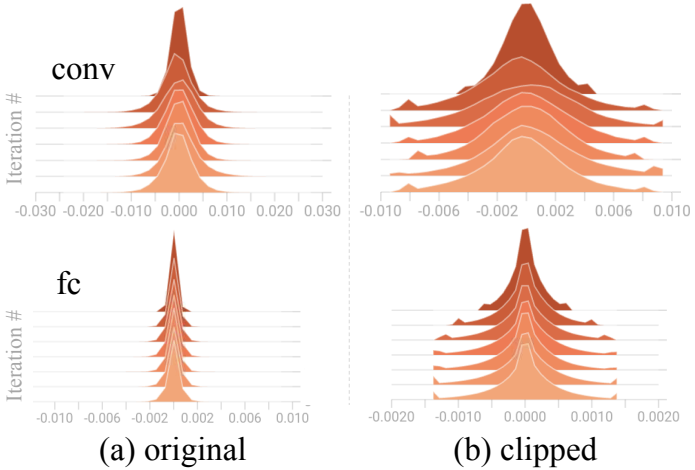
Approach:

1. Split gradients to buckets
2. Do TernGrad bucket by bucket

When bucket size == 1, TernGrad is floating SGD

Closing Bound Gap

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



(a) original (b) clipped

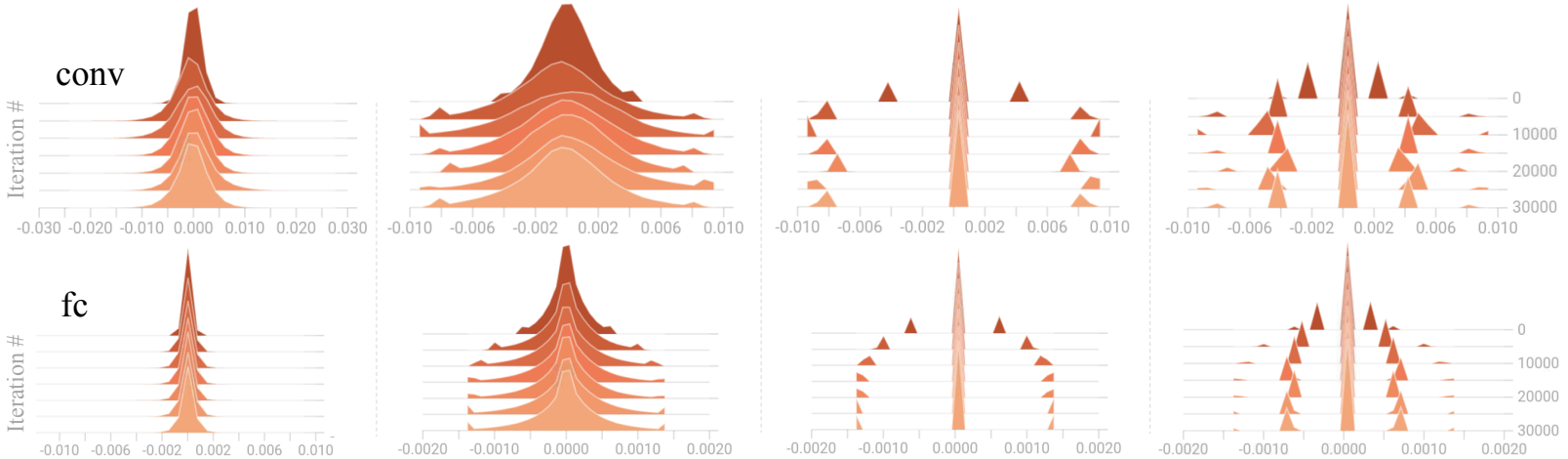
Gradient clipping

$$f(g_i) = \begin{cases} g_i & |g_i| \leq c\sigma \\ sign(g_i) \cdot c\sigma & |g_i| > c\sigma \end{cases}$$

c=2.5 works well for all tested datasets, DNNs and optimizers.

Suppose Gaussian distribution:
1. Change length 1.0%-1.5%
2. Change direction 2° -3°
3. Small bias with variance reduced

Gradient Histograms



(a) original (b) clipped (c) ternary (d) final

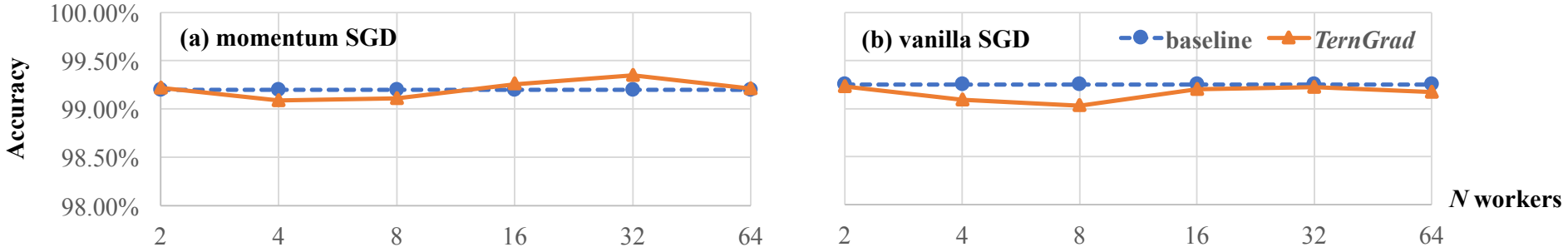
Average gradients are not ternary, but quantized with $2N+1$ levels (N: worker number).

Communication reduction: $\frac{32}{\log_2(2N+1)} > 1$, unless $N \geq 2^{30}$

Integration with Manifold Optimizers

(All experiments: All hyper-parameters are tuned for standard SGD and fixed in *TernGrad*)

LeNet (total mini-batch size 64): close accuracy & randomness in *TernGrad* results in small variance



Integration with Manifold Optimizers

CIFAR-10, mini-batch size 64 per worker

SGD	base LR	total mini-batch size	iterations	gradients	workers	accuracy
Adam	0.0002	128	300K	floating	2	86.56%
				TernGrad	2	85.64% (-0.92%)
Adam	0.0002	2048	18.75K	floating	16	83.19%
				TernGrad	16	82.80% (-0.39%)

Adam: D. P. Kingma, 2014

Tune hyper-parameters specifically for *TernGrad* may reduce accuracy gap

TernGrad works with manifold optimizers: Vanilla SGD, Momentum, Adam

Scaling to Large-scale Deep Learning

TernGrad: Randomness & regularization → (1) decrease randomness in dropout or (2) use smaller weight decay

No new hyper-parameters added

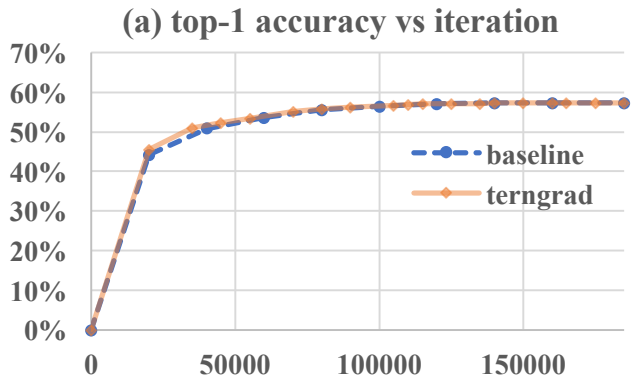
AlexNet

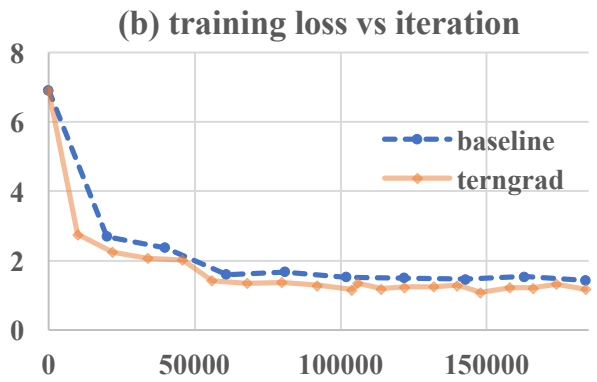
base LR	mini-batch size	workers	iterations	gradients	weight decay	DR†	top-1	top-5
0.01	256	2	370K	floating	0.0005	0.5	57.33%	80.56%
				TernGrad	0.0005	0.2	57.61%	80.47%
				TernGrad-noclip ‡	0.0005	0.2	54.63%	78.16%
0.02	512	4	185K	floating	0.0005	0.5	57.32%	80.73%
				TernGrad	0.0005	0.2	57.28%	80.23%
0.04	1024	8	92.5K	floating	0.0005	0.5	**56.62%**	80.28%
				TernGrad	0.0005	0.2	**57.54%**	80.25%

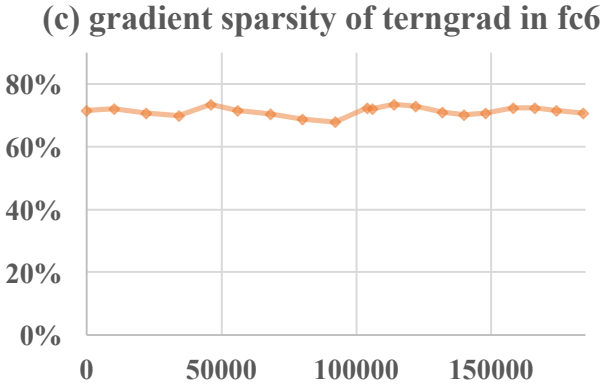
† DR: dropout ratio, the ratio of dropped neurons. ‡ *TernGrad* without gradient clipping.

N. S. Keskar, et al., ICLR 2017

Convergence Curves







AlexNet trained on 4 workers with mini-batch size 512

Coverages within the same epochs under the same base learning rate

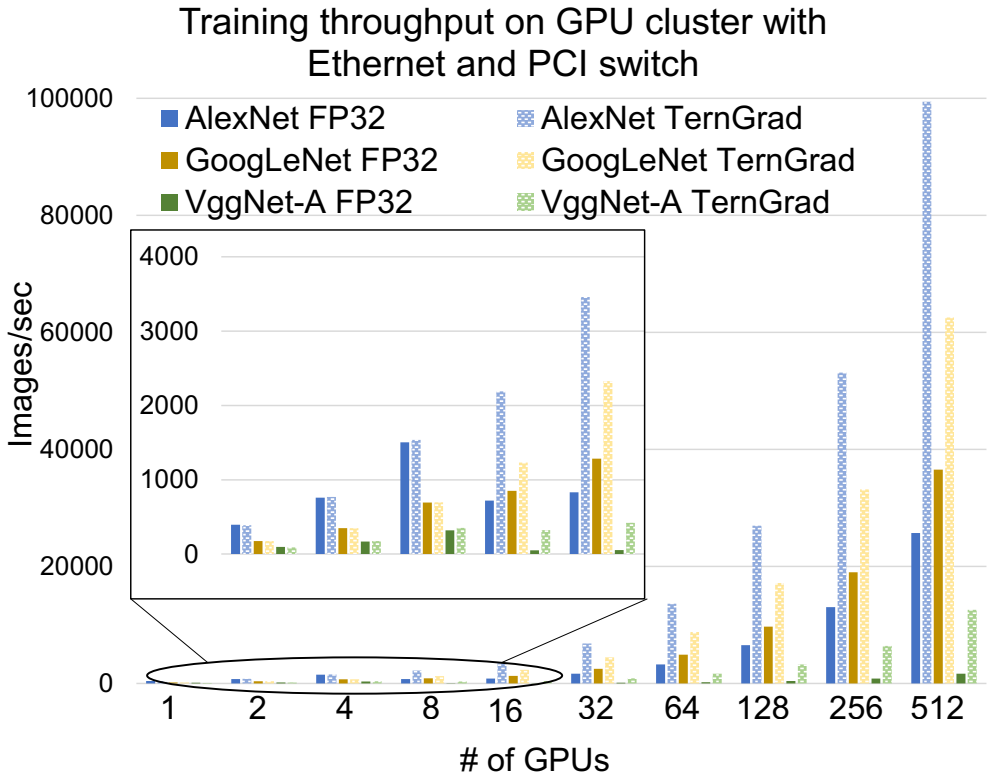
Scaling to Large-scale Deep Learning

base LR	mini-batch size	workers	iterations	gradients	weight decay	DR	top-5
0.04	128	2	600K	floating	4e-5	0.2	88.30%
				TernGrad	1e-5	0.08	86.77%
0.08	256	4	300K	floating	4e-5	0.2	87.82%
				TernGrad	1e-5	0.08	85.96%
0.10	512	8	300K	floating	4e-5	0.2	89.00%
				TernGrad	2e-5	0.08	86.47%

GoogLeNet accuracy loss is <2% on avgerage.

Tune hyper-parameters specifically for *TernGrad* may reduce accuracy gap

Performance Model



TernGrad gives higher speedup when

1. using more workers
2. using smaller communication bandwidth (Ethernet vs InfiniBand)
3. training DNNs with more fully-connected layers (VggNet vs GoogLeNet)

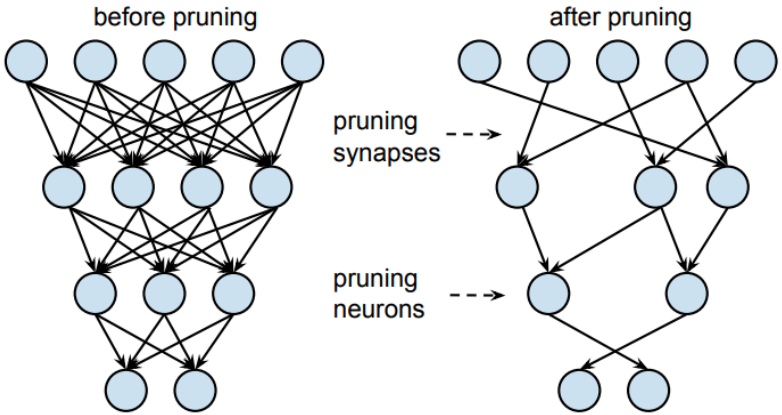


Structurally Sparse Deep Neural Networks

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, Hai Li, "*Learning Structured Sparsity in Deep Neural Networks*", NIPS, 2016. [paper][code][poster]

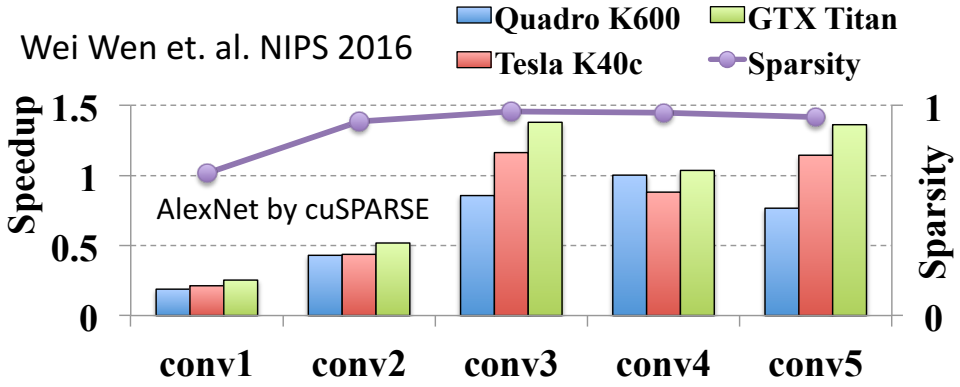
Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, Hai Li, "*Learning Intrinsic Sparse Structures within Long Short-Term Memory*", ICLR, 2018. [paper][code]

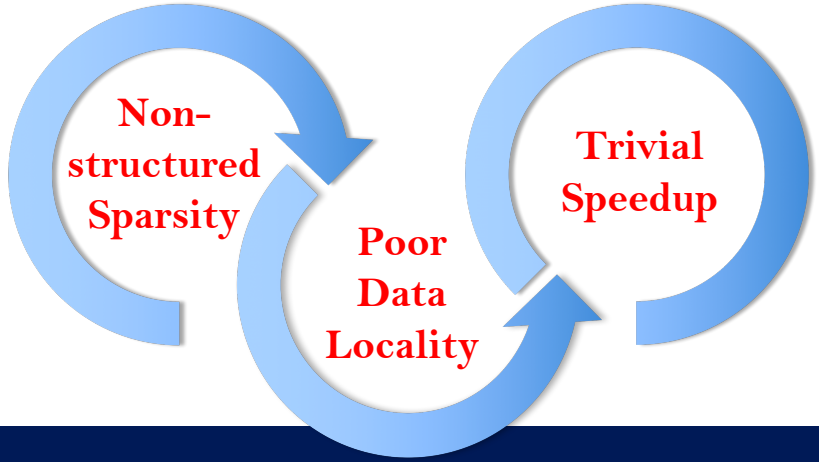
Sparse Convolutional Neural Networks (SCNNs)



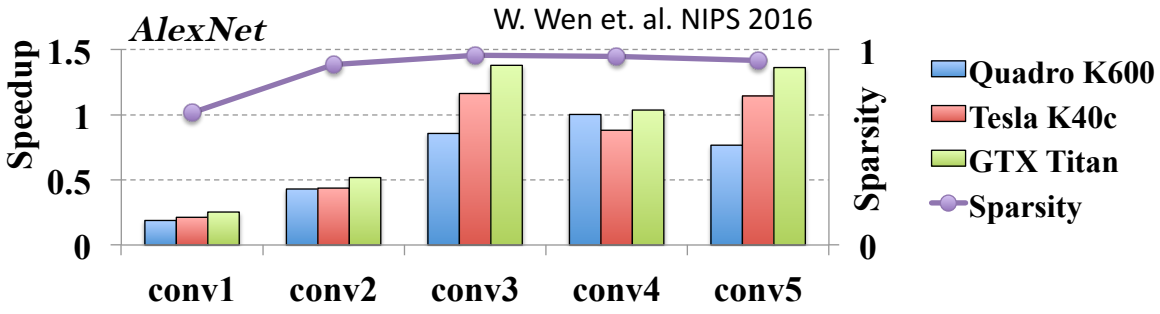
Significantly reduces the storage size of DNNs [1]
Good speedup with customized hardware [2]

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

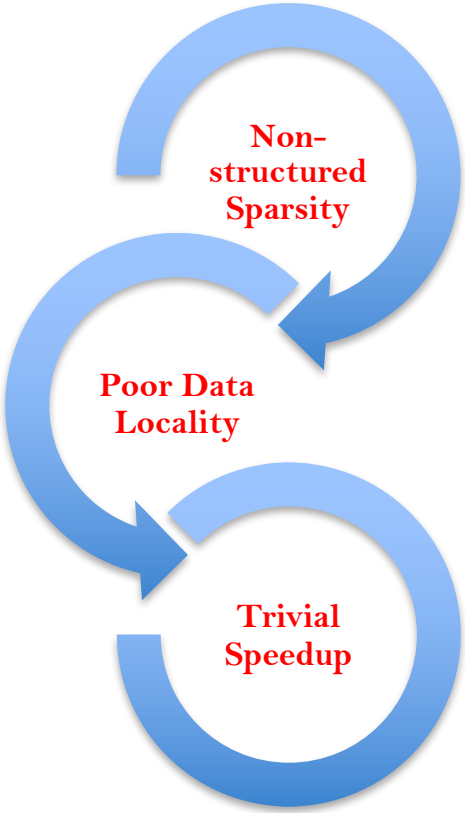




Inefficiency of SCNNs



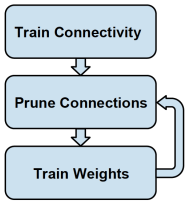
Speedup by SCNNs in GPUs

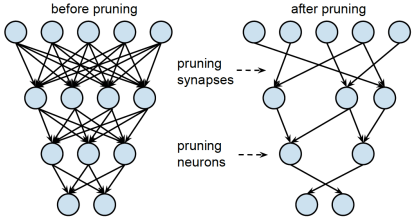


- Format: Sparse matrixes are formatted as Compressed Sparse Row (CSR)
- Acceleration library: cuSPARSE

Structured Sparsity in DNNs

S. Han et. al. NIPS 2015





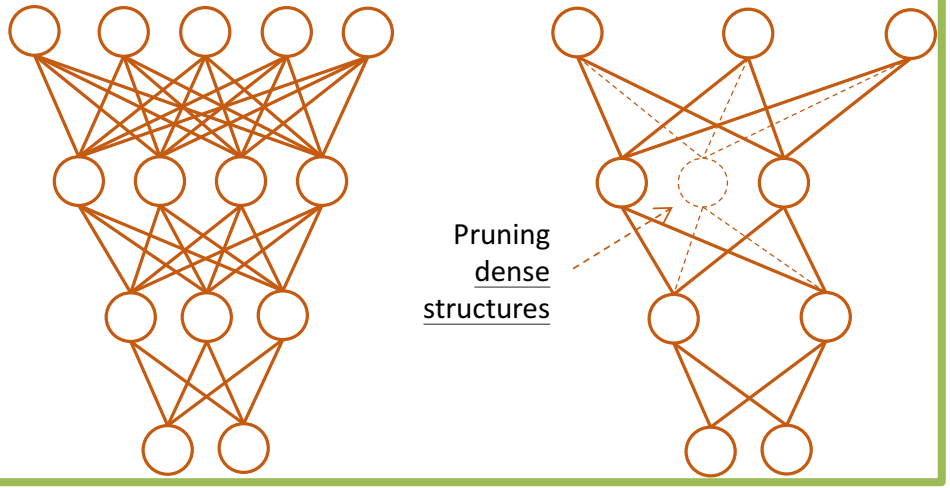
Faster Training speed

Higher practical computing speed

Wei Wen et. al. NIPS 2016
Wei Wen et. al. ICLR 2018

Train weights with Group Lasso Regularization

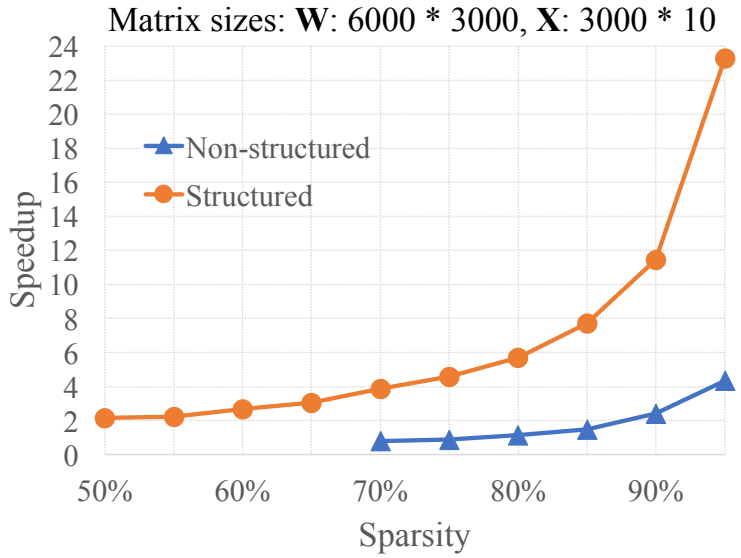
One-shot pruning from scratch (with the same epochs)

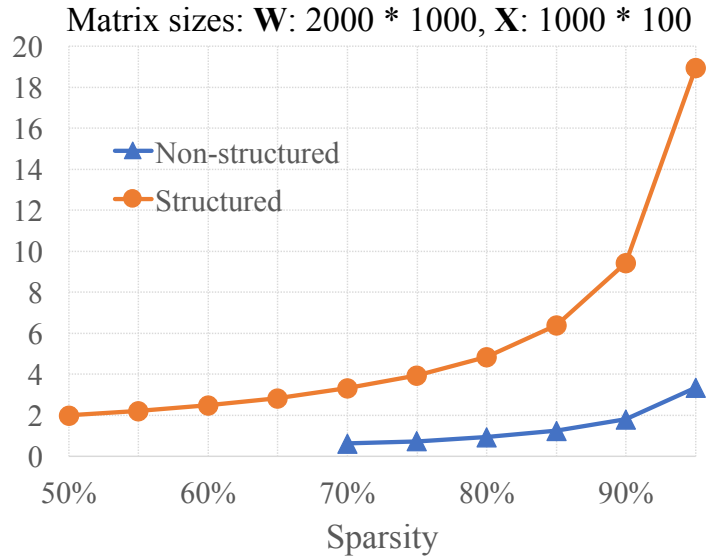


Efficiency of Structured Sparsity

W. Wen et. al. ICLR 2018

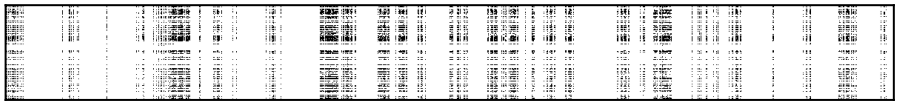
Tested on CPUs





Non-structured sparsity

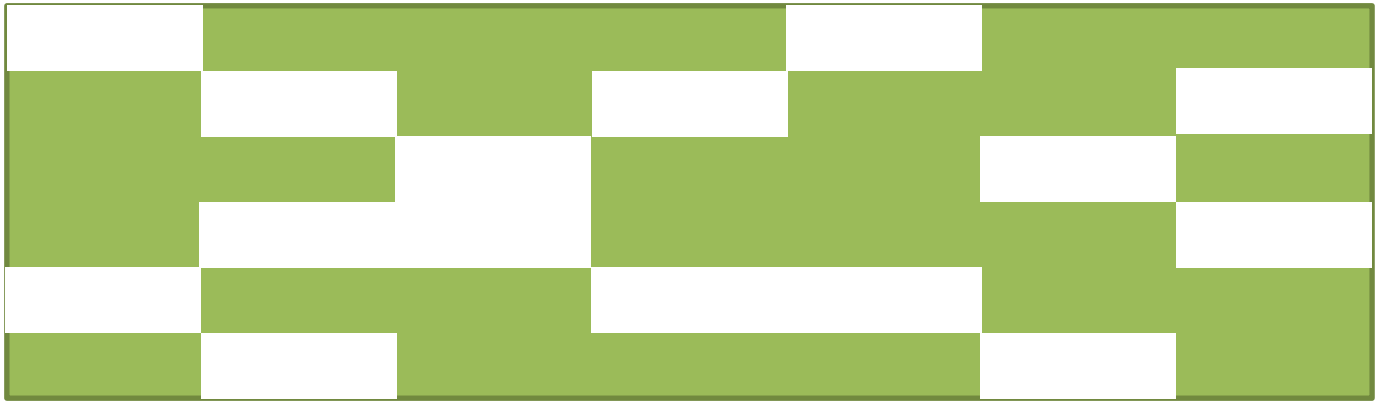
Structured sparsity



Structurally Sparse Deep Neural Networks

- What is Structurally Sparse Deep Neural Networks (SSDNNs)?

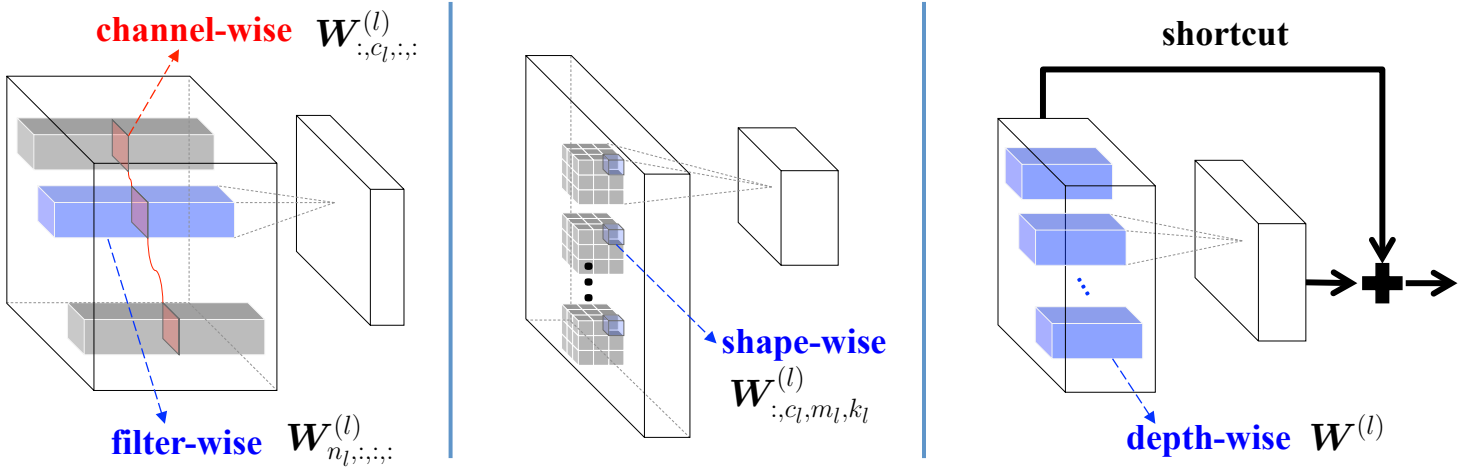
  Weights/connections are removed group by group.
  A group can be a rectangle block, a row, a column or even the whole matrix



Structurally Sparse Deep Neural Networks

In a nutshell, a group can be ***any*** form of a weight block, depending on what sparse structure you want to learn

In CNNs, a group of weights can be a channel, a 3D filter, a 2D filter, a filter shape fiber (i.e. a weight column), and even a layer (in ResNets).



Group Lasso 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: (w_1, w_2, w_3), group 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)***

Structured Sparsity Learning (*SSL*)

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

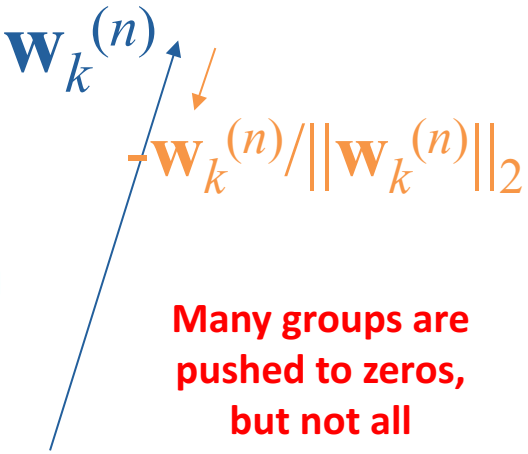
The gradient-based explanation:

$$\mathbf{w}_k^{(n)} \leftarrow \mathbf{w}_k^{(n)} - \eta \cdot \left(\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_k^{(n)}} + \lambda \cdot \frac{\mathbf{w}_k^{(n)}}{\left\| \mathbf{w}_k^{(n)} \right\|_2} \right)$$

A group of weights

Regular gradients to minimize error

Additional gradients to learn sparsity



Many groups are pushed to zeros, but not all

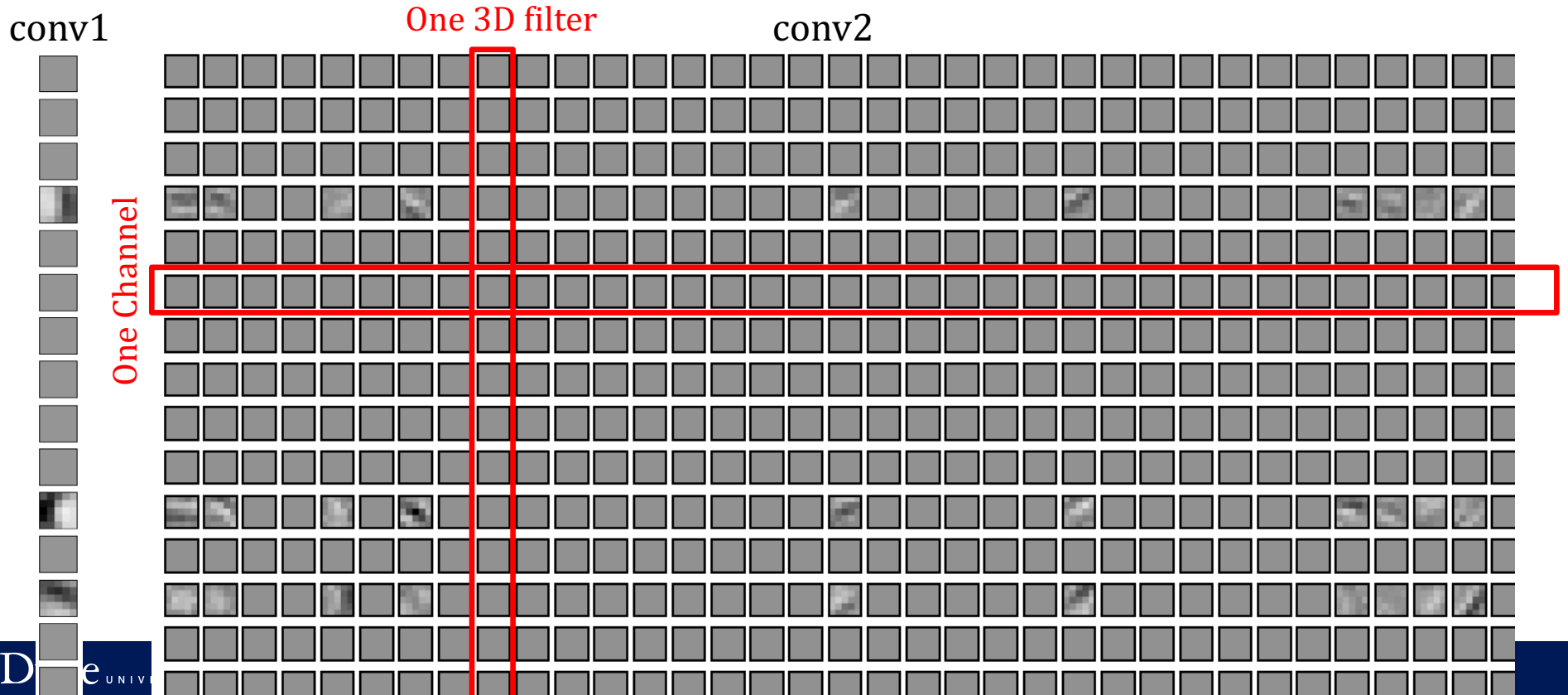
Easy to Implement

```
99  def add_dimen_grouplasso(var, axis=0):
100   with tf.name_scope("DimenGroupLasso"):
101     t = tf.square(var)
102     t = tf.reduce_sum(t, axis=axis) + tf.constant(1.0e-8)
103     t = tf.sqrt(t)
104     reg = tf.reduce_sum(t)
105     return reg
```

```
360             glasso_reg = add_dimen_grouplasso(train_var, axis=0)
361             self._regularization = self._regularization + glasso_reg * coef
```

```
408     grads, _ = tf.clip_by_global_norm(tf.gradients(cost + self._regularization, tvars),
409                                       config.max_grad_norm)
```

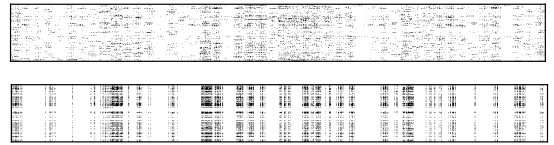
Structurally Sparse *LeNet* – removing channels and filters

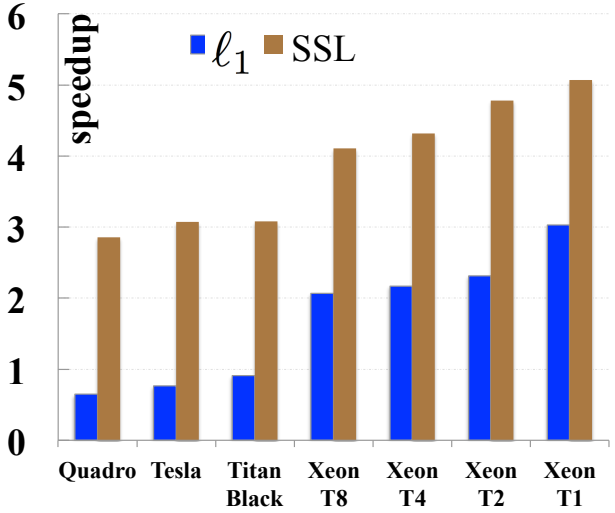


Structurally Sparse *AlexNet* – removing columns and rows

Table 4: Sparsity and speedup of *AlexNet* on ILSVRC 2012

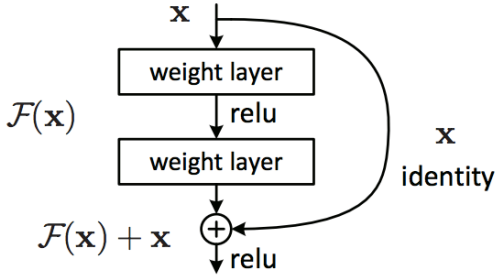
#		Method	Top1 err.	Statistics	conv1	conv2	conv3	conv4	conv5
1	2% loss	ℓ_1	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%
				CPU ×	0.80	2.91	4.84	3.83	2.76
				GPU ×	0.25	0.52	1.38	1.04	1.36
2		SSL	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%
				row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%
				CPU ×	1.05	3.37	6.27	9.73	4.93
				GPU ×	1.00	2.37	4.94	4.03	3.05
3		pruning[6]	42.80%	sparsity	16.0%	62.0%	65.0%	63.0%	63.0%
4	No loss	ℓ_1	42.51%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%
				CPU ×	0.34	0.99	1.30	1.10	0.93
				GPU ×	0.08	0.17	0.42	0.30	0.32
5		SSL	42.53%	column sparsity	0.00%	20.9%	39.7%	39.7%	24.6%
				CPU ×	1.00	1.27	1.64	1.68	1.32
				GPU ×	1.00	1.25	1.63	1.72	1.36





[6] S. Han, et. al. NIPS 2015

Structurally Sparse *ResNets* – removing layers



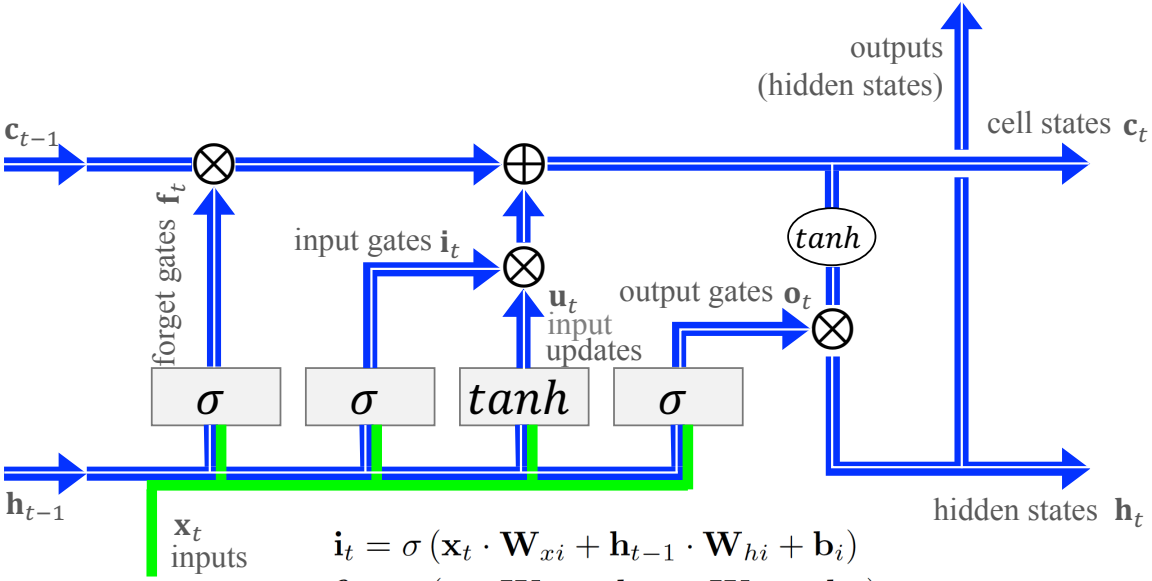
He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *CVPR*. 2016.

ResNet-20/32: baseline with 20/32 layers

SSL-ResNet-#: Ours with # layers after removing layers in ResNet-20

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

Hidden Structures in LSTMs

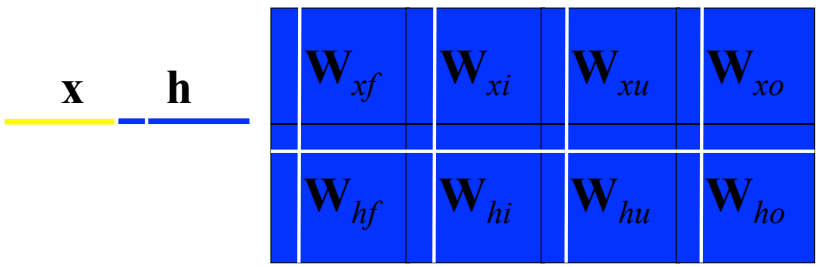


$$\mathbf{i}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xi} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{b}_i\right)$$
$$\mathbf{f}_t = \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xf} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{b}_f\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{x}_t \cdot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

"Dimension Consistency":

1. **Hidden Structures (in blue):** hidden states, input updates, cells, gates, outputs
2. All hidden structures must have the same dimension
3. Hidden structures cannot be independently removed

Learning Sparse Hidden Structures in LSTMs



Weights matrices in LSTM

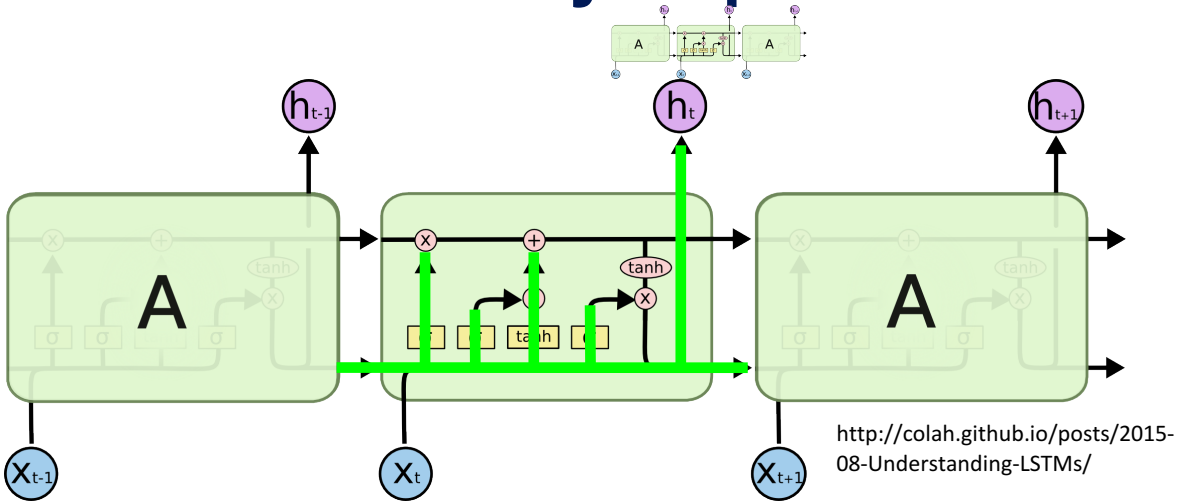
$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xi} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xf} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{x}_t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

Weights in next layer(s)

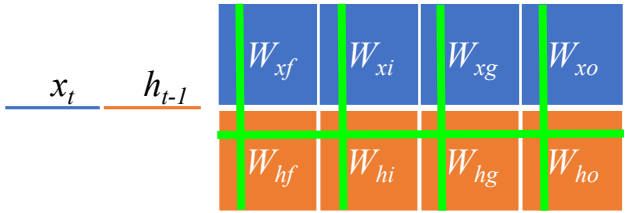
White strips (ISS: Intrinsic Sparse Structures) = *one* group of weights in SSL

Removing one group = reduce hidden size by one

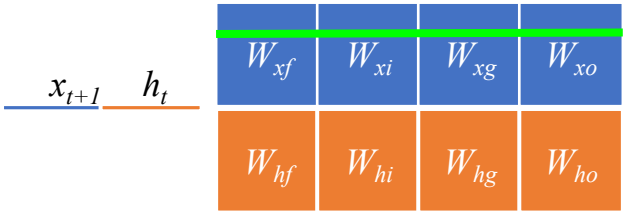
Structurally Sparse *LSTMs*



http://colah.github.io/posts/2015-08-Understanding-LSTMs/



LSTM 1



LSTM 2

A group in SSL

Removing one group == reducing hidden size by one

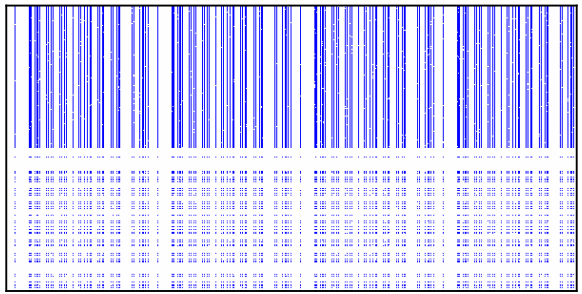
Structurally Sparse *LSTMs*

Method	Dropout keep ratio	Perplexity (validate, test)	ISS # in (1st , 2nd) LSTM	Weight #	Total time*	Speedup	Mult-add reduction†
baseline	0.35	(82.57, 78.57)	(1500, 1500)	66.0M	157.0ms	1.00×	1.00×
ISS	0.60	(82.59, 78.65)	(373, 315)	21.8M	14.82ms	10.59×	7.48×
		(80.24, 76.03)	(381, 535)	25.2M	22.11ms	7.10×	5.01×
direct design	0.55	(90.31, 85.66)	(373, 315)	21.8M	14.82ms	10.59×	7.48×

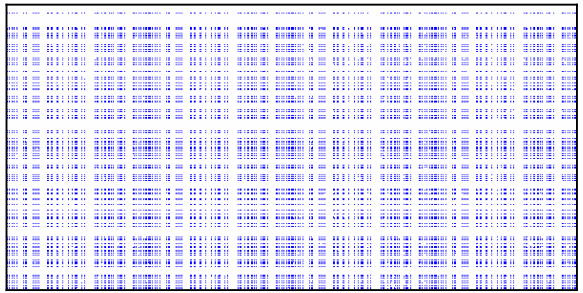
* Measured with 10 batch size and 30 unrolled steps.

† The reduction of multiplication-add operations in matrix multiplication. Defined as (original Mult-add)/(left Mult-add)

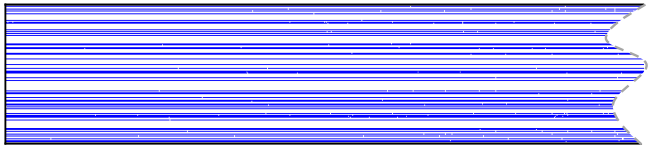
LSTM 1



LSTM 2



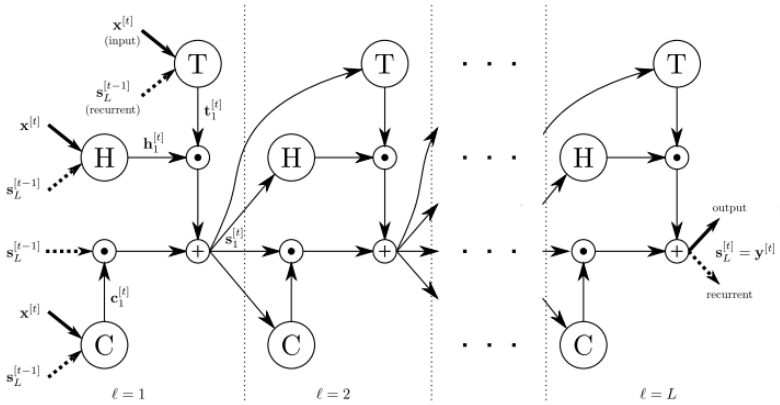
Output



Learned structurally sparse LSTMs

Structurally Sparse *Recurrent Highway Networks*

What weights does a group have?



Zilly, Julian Georg, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. "Recurrent highway networks." *arXiv:1607.03474* (2016).

Table 2: Learning ISS sparsity from scratch in RHNs.

	Method	λ	Perplexity (validate, test)	RHN width	Parameter #
	baseline	0.0	(67.9, 65.4)	830	23.5M
Ours	ISS*	0.004	(67.5, 65.0)	726	18.9M
Ours	ISS*	0.005	**(68.1, 65.4)**	**517**	**11.1M**
Ours	ISS*	0.006	(70.3, 67.7)	403	7.6M
Ours	ISS*	0.007	(74.5, 71.2)	328	5.7M

* All dropout ratios are multiplied by $0.6\times$.

Question Answering with SSDNNs

Question Answering in search engine:

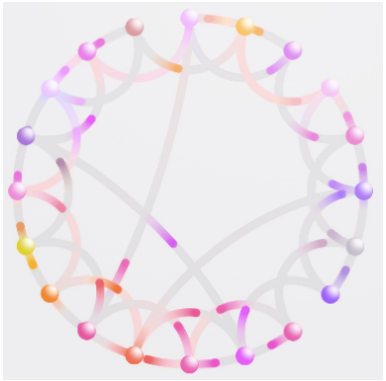
Table 4: Remaining ISS components in BiDAF by training from scratch.

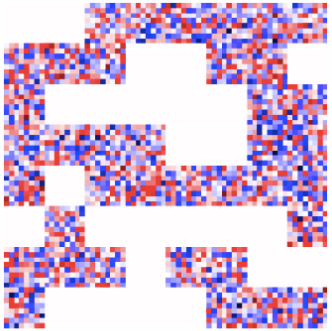
	EM	F1	ModFwd1	ModBwd1	ModFwd2	ModBwd2	OutFwd	OutBwd	weight #	Total time*
Baseline	67.98	77.85	100	100	100	100	100	100	2.69M	6.20ms
Ours	67.36	77.16	87	81	87	92	74	96	2.29M	5.83ms
	66.32	76.22	51	33	42	58	37	26	1.17M	4.46ms
	65.36	75.78	20	33	40	38	31	16	0.95M	3.59ms
	64.60	74.99	23	22	35	35	25	14	0.88M	2.74ms

* Measured with batch size 1.

Efficiency of Structurally Sparse DNNs (after us)

OpenAI



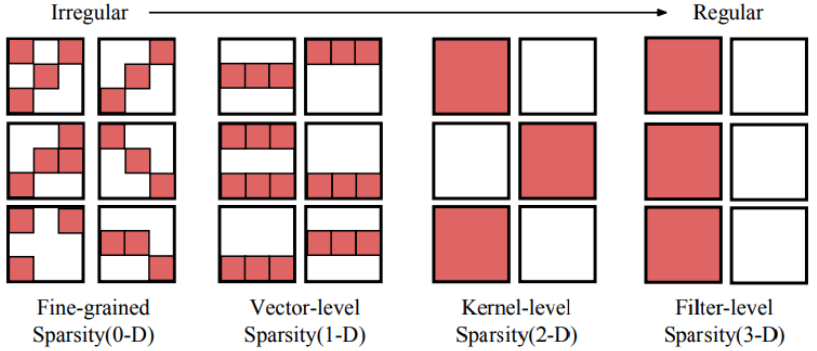


Block-sparse weights

S. Gray, A. Radford and D. P. Kingma, GPU Kernels for Block-Sparse Weights, OpenAI 2017

Structured sparsity is more efficient for computation

DEEPHI 深鉴科技



H. Mao , S. Han, et. al., CVPR Workshop 2017

Structured sparsity is more efficient for storage

Conclusion

- AI systems are landing into the cloud and upon the edge
- Communication bottleneck limits the scalability of distributed deep learning in the cloud
  - We propose *TernGrad* to quantize gradients to reduce communication volume.
- Lightweight computing engine restricts the deployment of cumbersome Deep Neural Networks on the edge
  - We propose Structurally Sparse Deep Neural Networks to simplify the “rocket” to lighten the load on the “engine”
- More done/on-going work on large-batch training and model acceleration
  - http://www.pittnuts.com/#Publications

THANKS! Q&A?

Wei Wen, Duke University
www.pittnuts.com