How to Obtain and Run Light and Efficient Deep Learning Networks

Yiran Chen

Electrical and Computing Engineering Department

Duke Center for Evolutionary Intelligence (CEI)

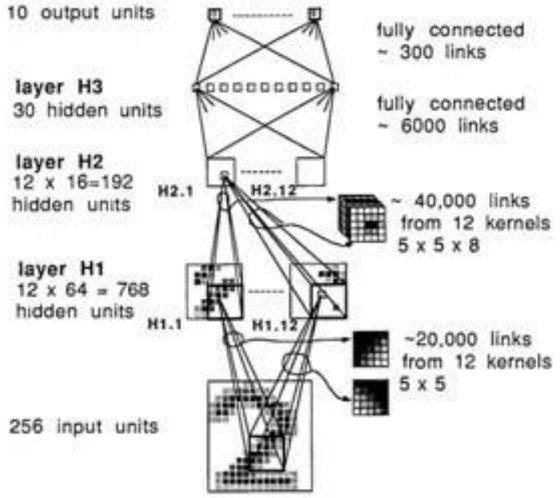
NSF IUCRC For Alternative Sustainable and Intelligent Computing (ASIC)

Outline

- Introduction
- Learning on IBM TrueNorth Chip
  - DAC 2016 Best Paper Nomination
- Learning Structured Sparsity in Deep Neural Networks
  - NIPS 2016
- TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning
  - NIPS 2017 (oral)
- Our prospective

Rise and Decline of Neural Network

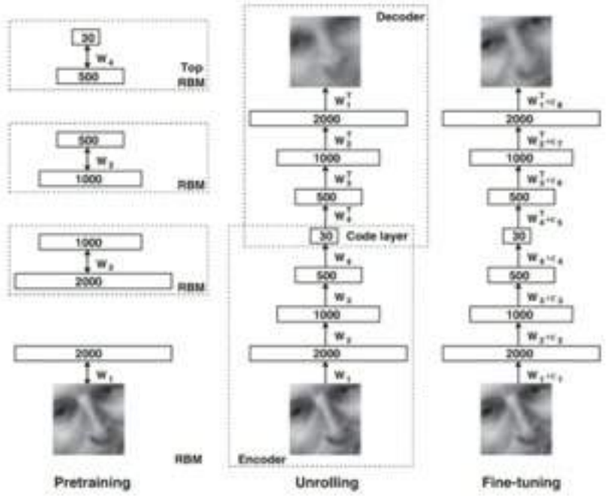
Convolutional Network (1980s)



Dark period (1990s)

- Serious problem: Vanishing gradient
- No benefits observed by adding more layers
- No high performance computing devices

Renaissance (2006 ~ Present)

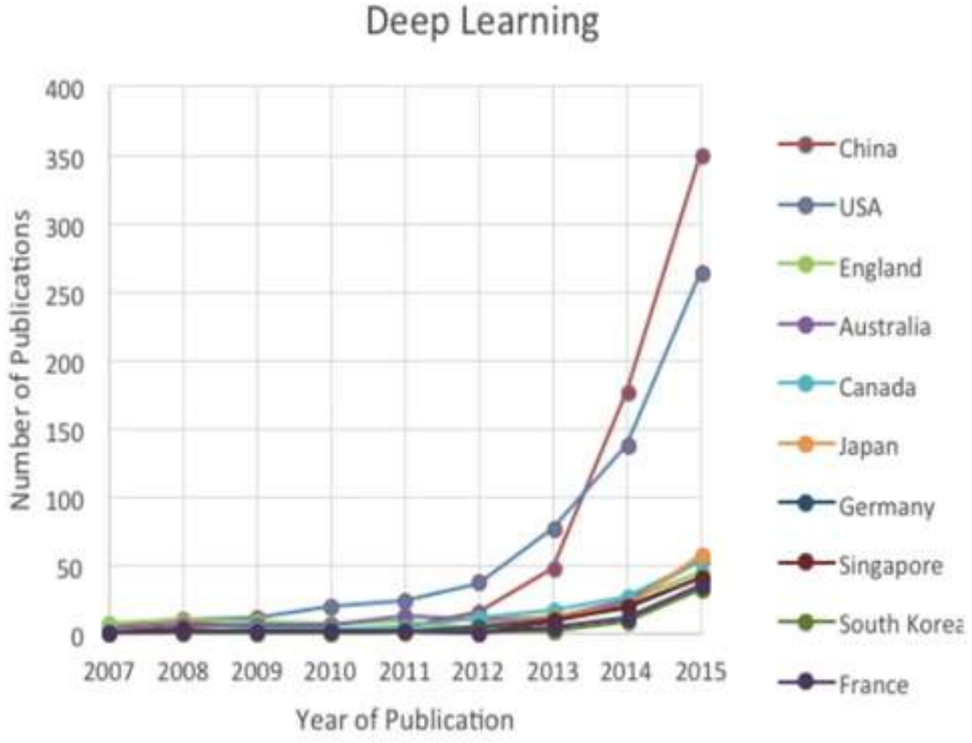


Y. Lecun, *B.* Boser, *J. S.* Denker, *D.* Henderson, *R. E.* Howard, *W.* Hubbard *and L. D.* Jackel. *Backpropagation Applied to Handwritten Zip Code Recognition.* 1989.

J. Schmidhuber. Deep Learning in Neural Networks: An Overview. arxiv, 2014.

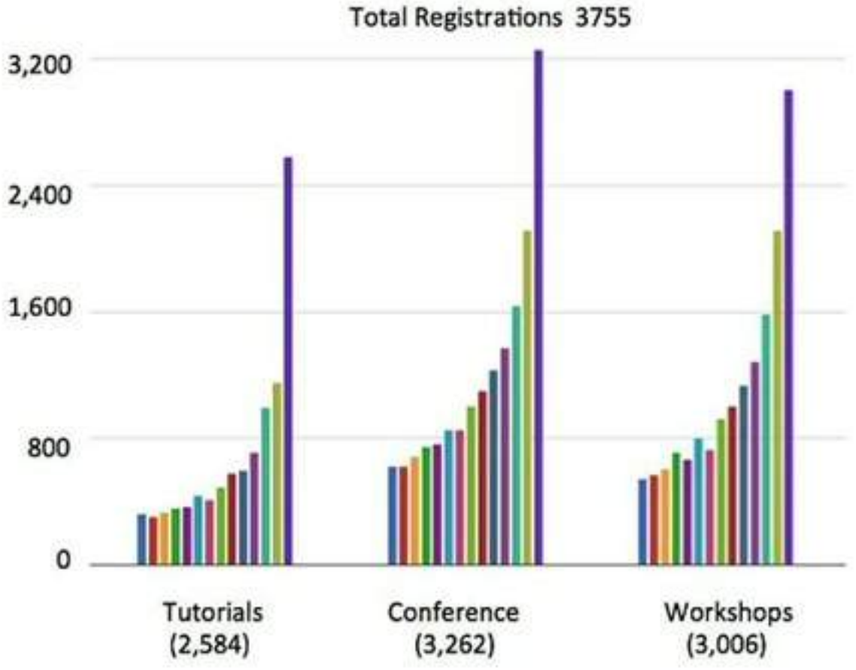
Machine Learning in Academia

Journal articles mentioning "**deep learning**" or "**deep neural networks**"



Source: Office of Science and Technology Policy/The White House

NIPS registrations growth 2015: 3755, 2016: 6000+



NIPS does not organize sponsor presentations at the Conference or Workshops an
sponsors. Satellite meetings should be organized by sponsors immediatley before

2017 Levels of Sponsorship:

All Levels of Sponsorships are SOLD OUT.

Platinum

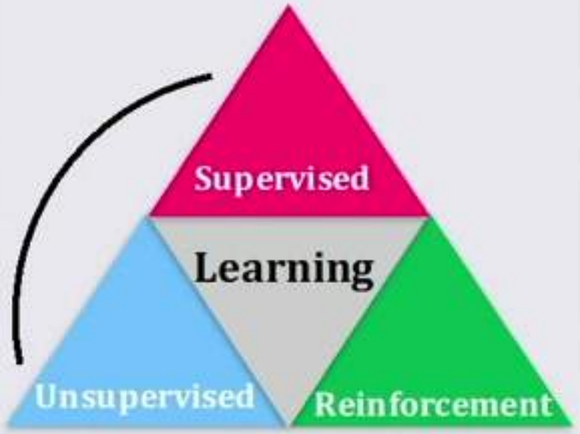
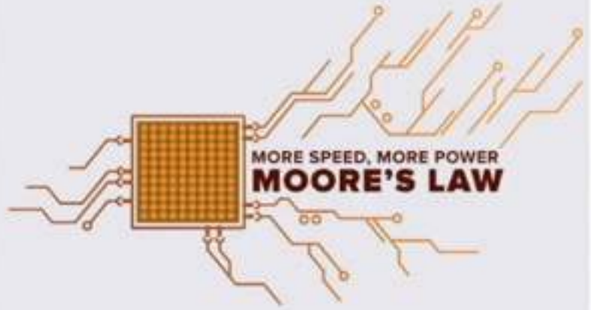
Machine Learning in the Market

Technology cycle - from *PC*, to *smartphone*, to ***artificial intelligence?***

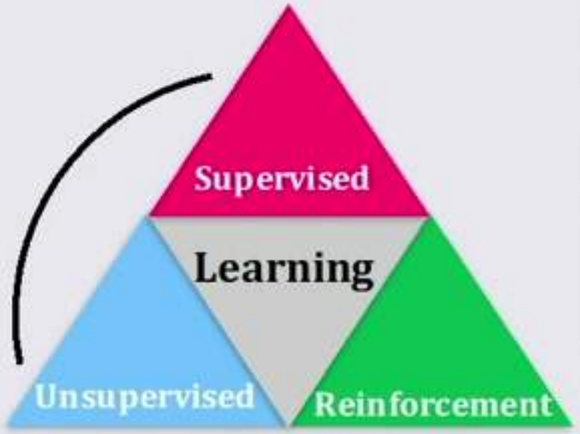
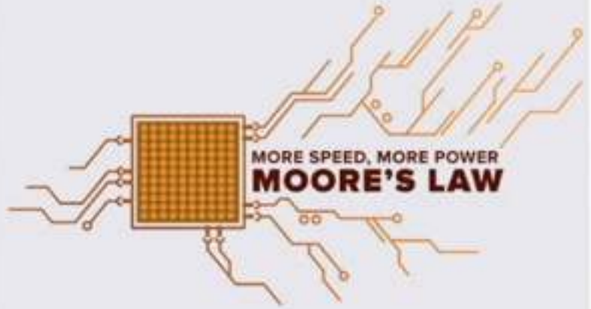
"Pure Play" Share Price Performance

Source: Bloomberg, Jefferies

Why is Deep Learning Hot Now?

Application	Algorithm	Hardware
Big Data	**ML Techniques**	**Computation Power**
		
Numbers: 5 KB/record Text: 500 KB/record Image: 1000 KB/picture Audio: 5000 KB/song Video: 5,000,000 KB/movie	Advances in algorithm innovation, including neural networks, leading to better accuracy in training models	• Transistor density doubles every 18mo • Computation/kwh doubles every 18mo • Cost/Gigabyte in 1995: $1000.00 • Cost/Gigabyte in 2015: $0.03

Our Works on Deep Learning

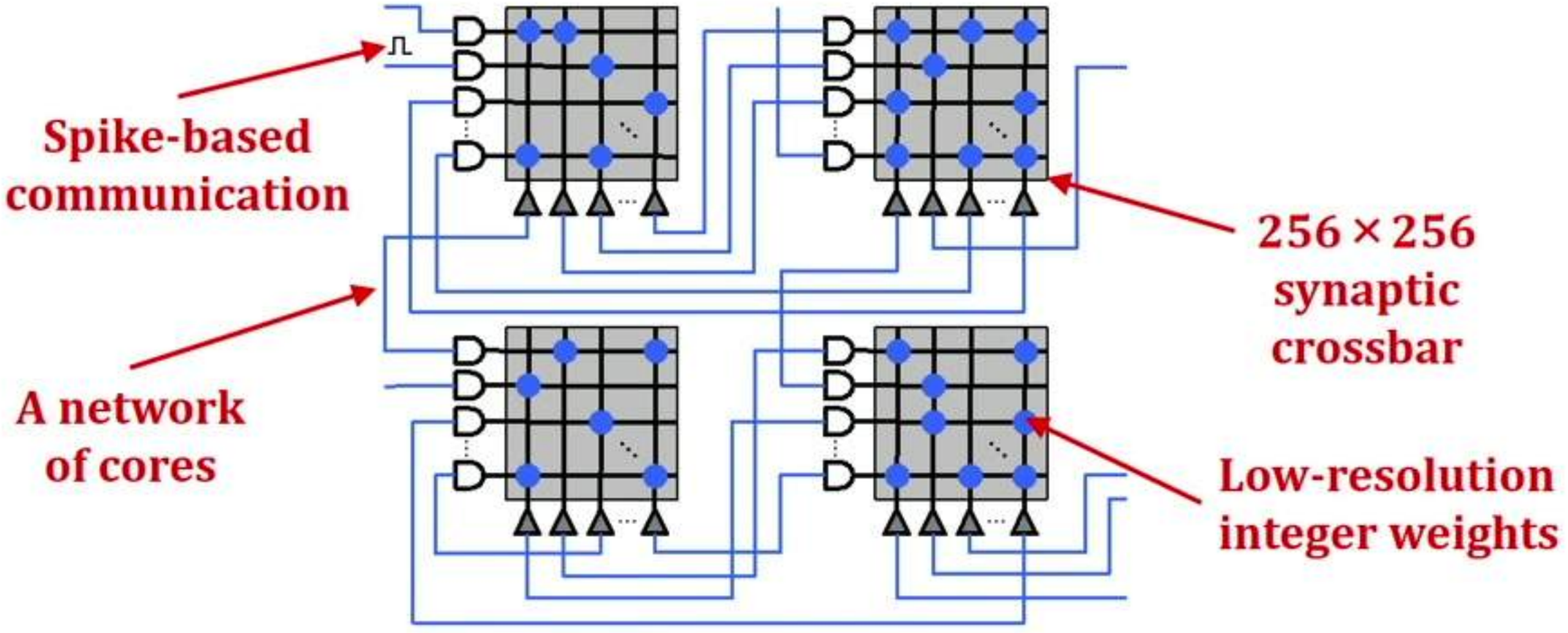
Application	Algorithm	Hardware
Big Data	**ML Techniques**	**Computation Power**
		
Image Segmentation for Self-driving **Adversarial attack, differential privacy**	**Efficient Algorithms for DNN deployment on the Cloud and the Edge**	**Heterogeneous Deep Learning Acceleration** **Learning on IBM TrueNorth Chip** **ReRAM-based Neuromorphic Computing**

Learning on IBM TrueNorth Chip

Duke
UNIVERSITY

IBM TrueNorth – Architecture

- 4,096 neurosynaptic cores
- 1 million neurons
- 256 million synapses
- A 65mW real-time neurosynaptic processor [1][2]



[1] A. S. Cassidy, etc, *SC'14*; [2] F. Akopyan, *etc.*, *TCAD ICS'15*

IBM TrueNorth – Learning & Deploying

Database

Raw Pixels

[1] S. K. Esser, etc., *NIPS'15*

IBM TrueNorth – Learning & Deploying

Database

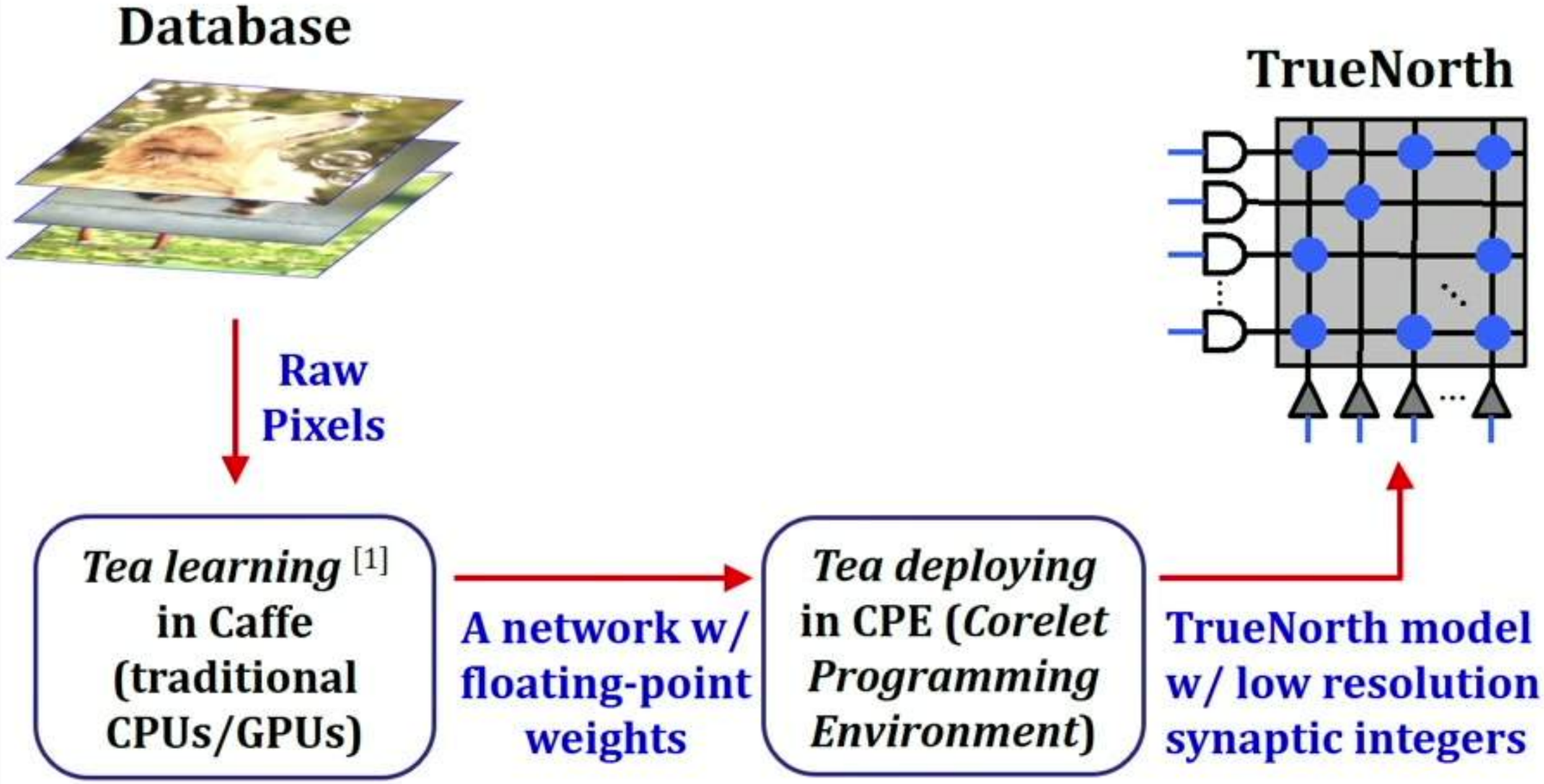
Raw Pixels

Tea learning [1] **in Caffe (traditional CPUs/GPUs)**

A network w/ floating-point weights

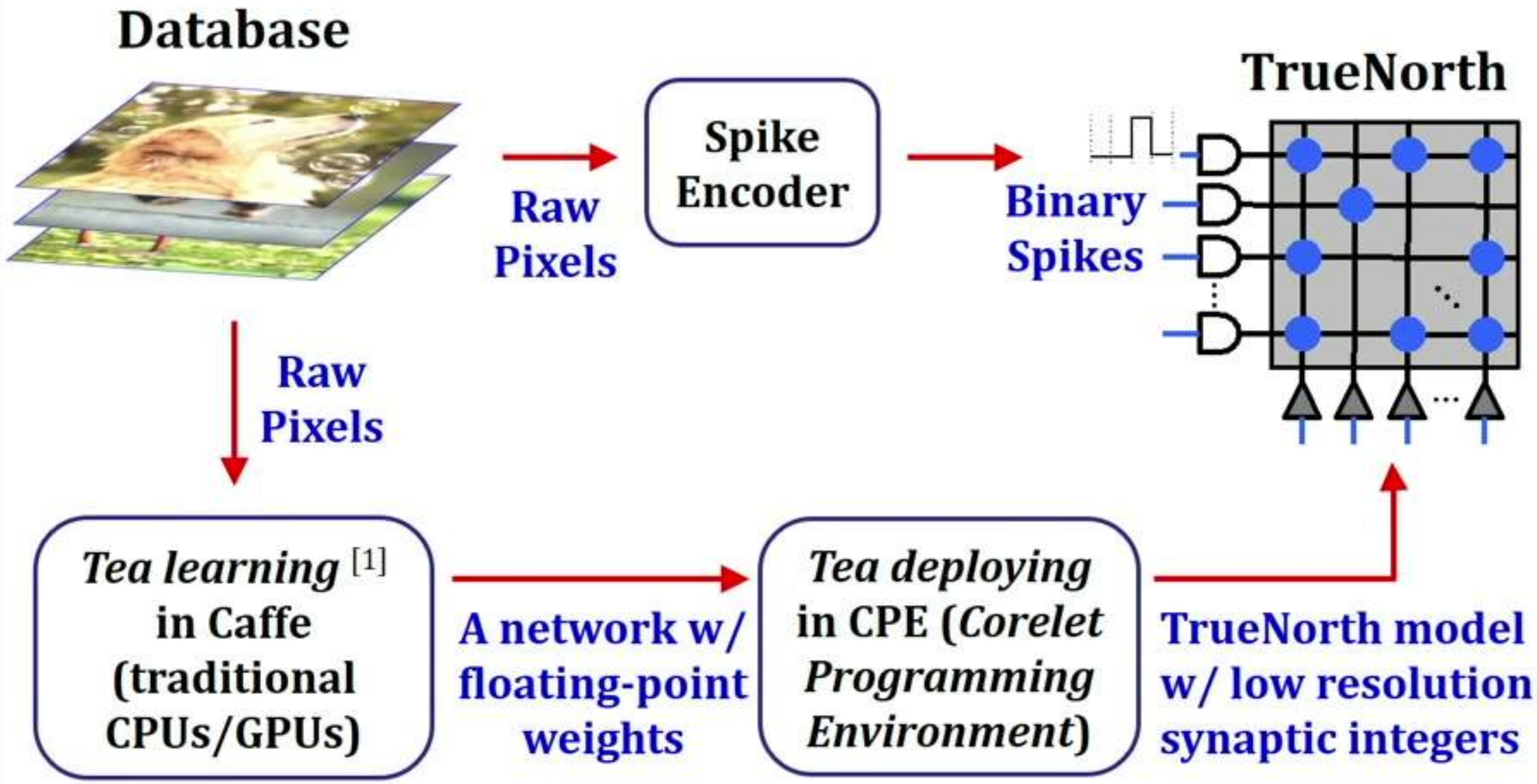
[1] S. K. Esser, etc., *NIPS'15*

IBM TrueNorth – Learning & Deploying



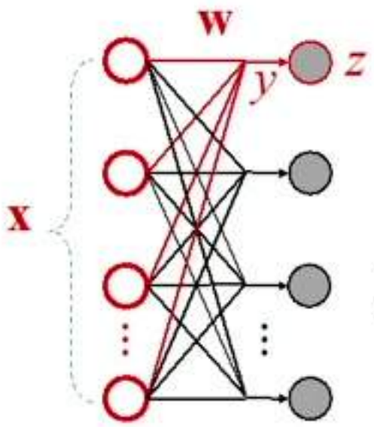
[1] S. K. Esser, etc., *NIPS'15*

IBM TrueNorth – Learning & Deploying



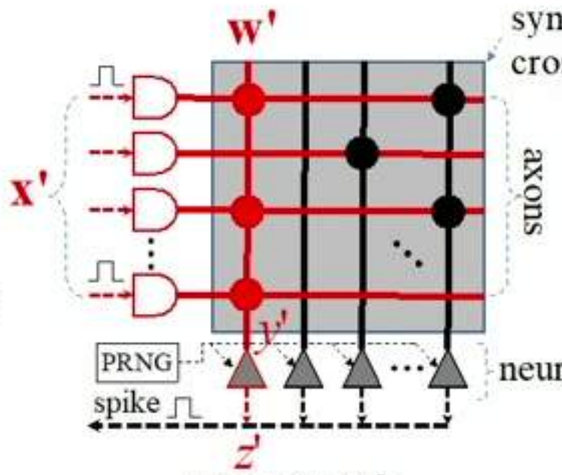
[1] S. K. Esser, etc., *NIPS'15*

Deploy a Network on TrueNorth



$$y = \mathbf{w} \cdot \mathbf{x} + b$$
$$z = h(y)$$

Traditional neuron model



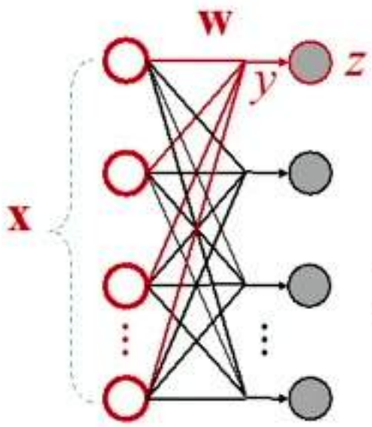
TrueNorth neuron model

McCulloch-Pitts neuron model:

$$y' = \mathbf{w}' \cdot \mathbf{x}' - \lambda$$

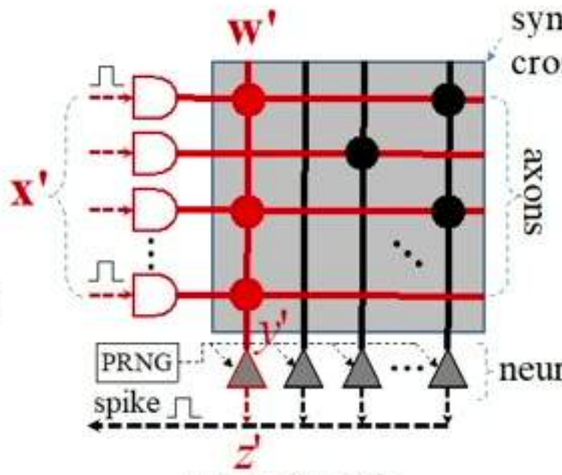
$$z' = \begin{cases} 1, \textit{ reset } y' = 0; \textit{ if } y' \geq 0 \\ 0, \textit{ reset } y' = 0; \textit{ if } y' < 0 \end{cases}$$

Deploy a Network on TrueNorth



$$y = \mathbf{w} \cdot \mathbf{x} + b$$
$$z = h(y)$$

Traditional neuron model



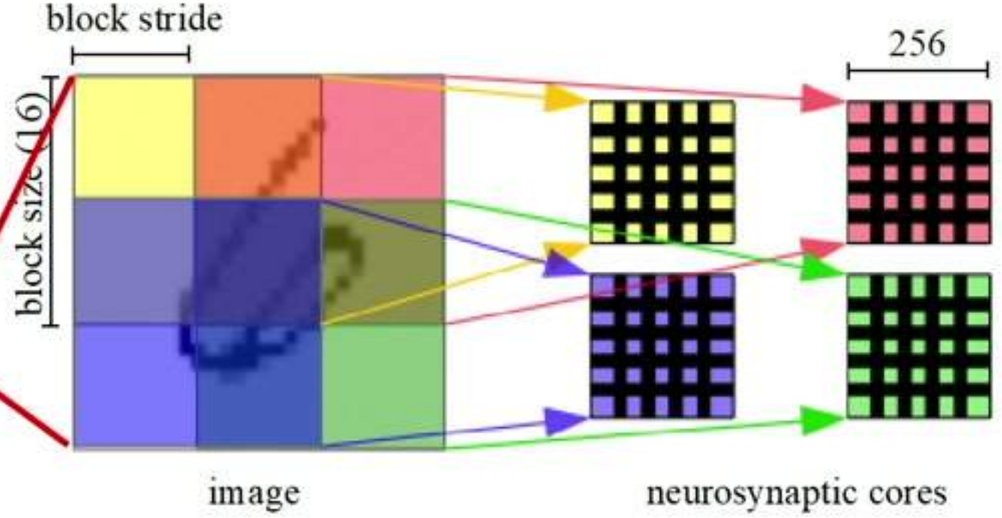
TrueNorth neuron model

McCulloch-Pitts neuron model:

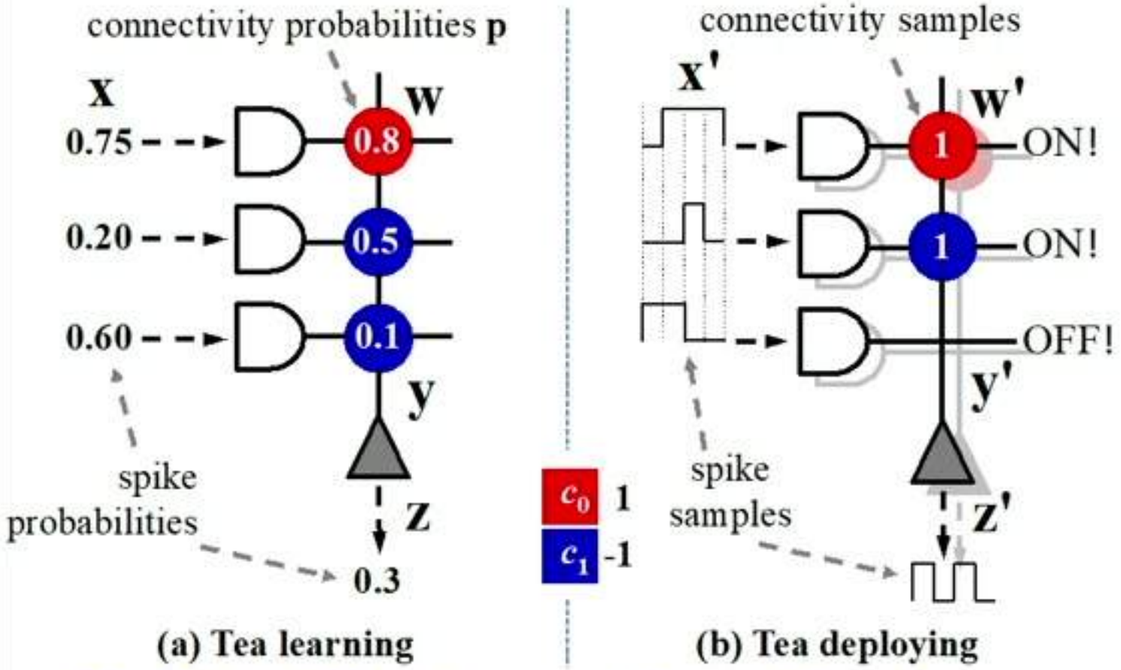
$$y' = \mathbf{w}' \cdot \mathbf{x}' - \lambda$$

$$z' = \begin{cases} 1, \textit{ reset } y' = 0; \textit{ if } y' \geq 0 \\ 0, \textit{ reset } y' = 0; \textit{ if } y' < 0 \end{cases}$$

MNIST



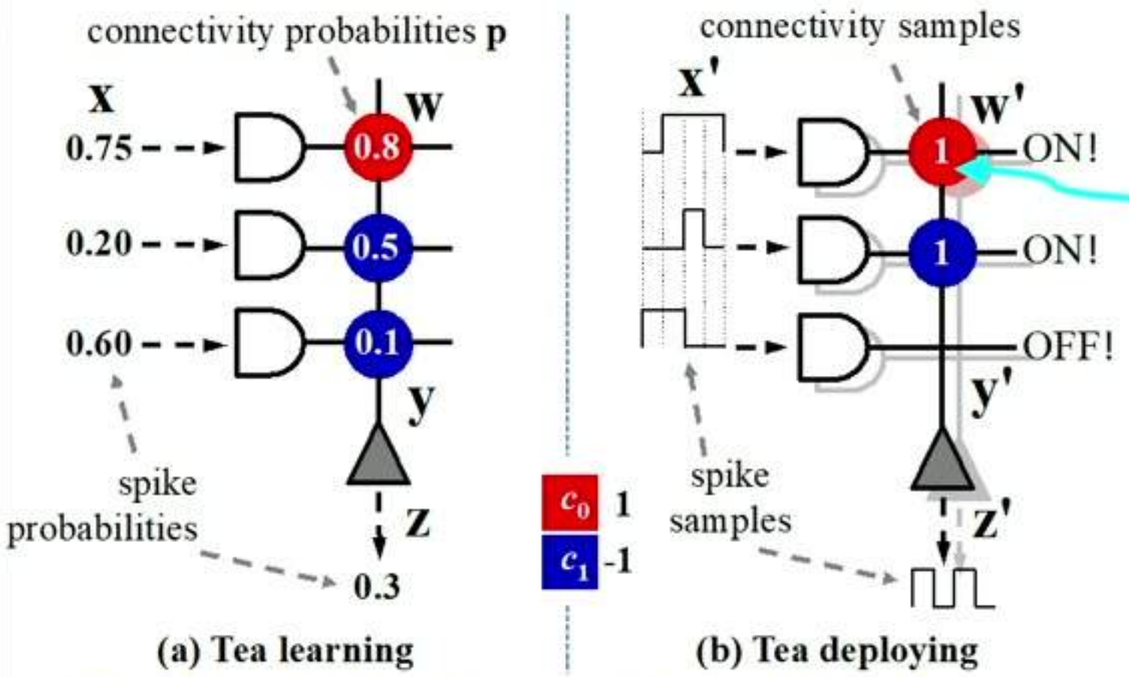
Deploy a Network on TrueNorth



Traditional floating-point precision

Binary/low integer precision *sampled by float-point probability*

Deploy a Network on TrueNorth



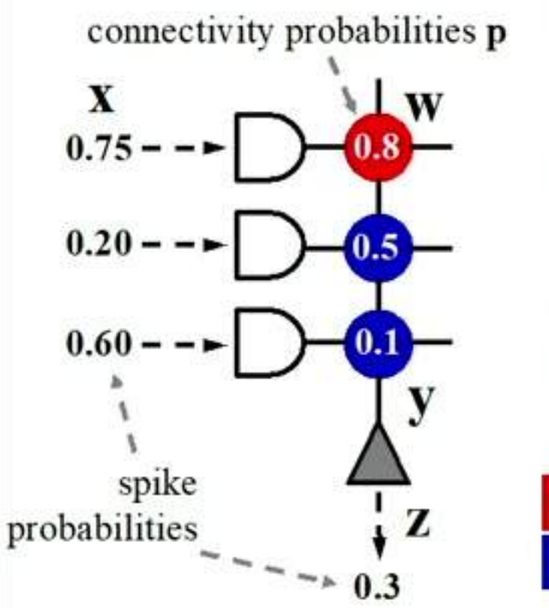
$$p_i = w_i / c^{(i)}, c^{(i)} \in \{c_0, c_1\}$$

$$\begin{cases} P(w_i' = c^{(i)}) = p_i \\ P(w_i' = 0) = 1 - p_i \end{cases}$$

Traditional floating-point precision

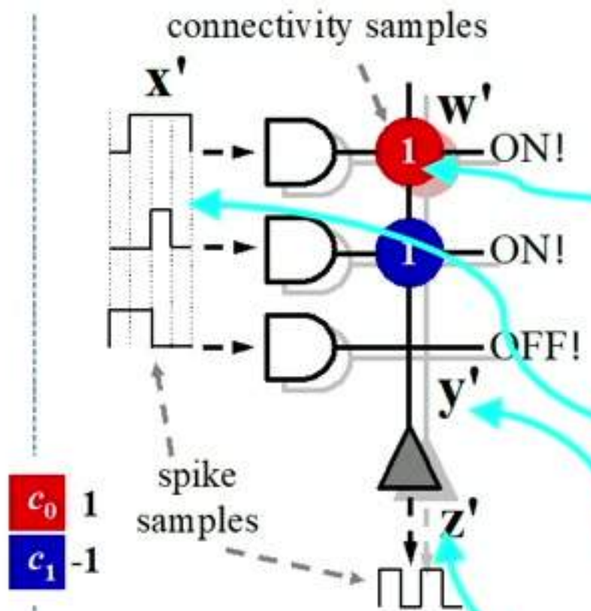
Binary/low integer precision *sampled by float-point probability*

Deploy a Network on TrueNorth



(a) Tea learning

Traditional floating-point precision



(b) Tea deploying

Binary/low integer precision *sampled by float-point probability*

$$p_i = w_i / c^{(i)}, c^{(i)} \in \{c_0, c_1\}$$

$$\begin{cases} P(w_i' = c^{(i)}) = p_i \\ P(w_i' = 0) = 1 - p_i \end{cases}$$

$$\begin{cases} P(x_i' = 1) = x_i \\ P(x_i' = 0) = 1 - x_i \end{cases}$$

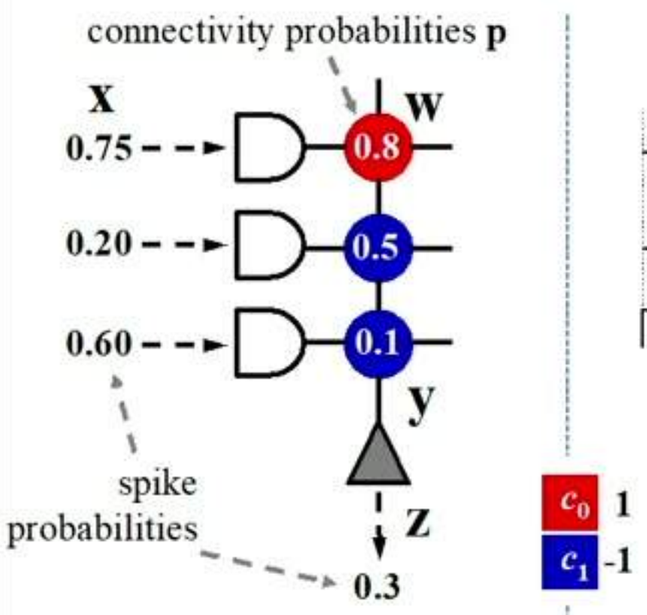
$$y' = \sum_{i=0}^{n-1} w_i' x_i'$$

$$E\{y'\} = E\left\{\sum_{i=0}^{n-1} w_i' x_i'\right\} = \sum_{i=0}^{n-1} E\{w_i'\} E\{x_i'\}$$

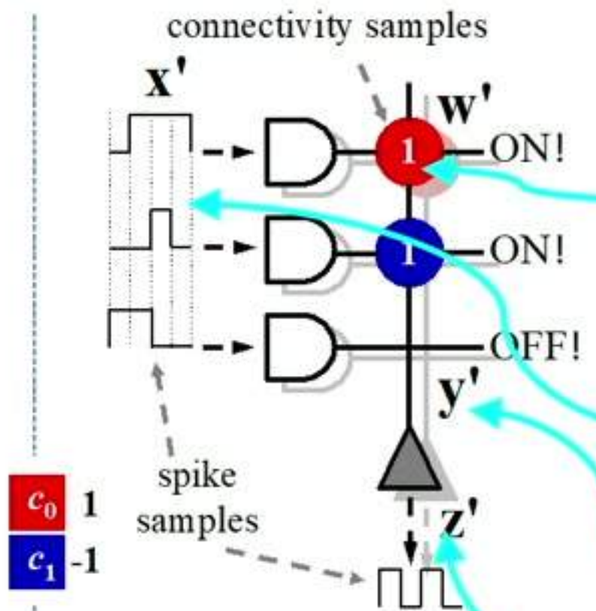
$$= \sum_{i=0}^{n-1} p_i c^{(i)} x_i = \sum_{i=0}^{n-1} w_i x_i = y$$

$$E\{z'\} = P(y' \geq 0) = \frac{1}{2}\left[1 + erf\left(\frac{-\mu_{y'}}{\sqrt{2}\sigma_{y'}}\right)\right]$$

Deploy a Network on TrueNorth



(a) Tea learning



(b) Tea deploying

Traditional floating-point precision

Binary/low integer precision *sampled by float-point probability*

$$p_i = w_i / c^{(i)}, c^{(i)} \in \{c_0, c_1\}$$

$$\begin{cases} P(w_i' = c^{(i)}) = p_i \\ P(w_i' = 0) = 1 - p_i \end{cases}$$

$$\begin{cases} P(x_i' = 1) = x_i \\ P(x_i' = 0) = 1 - x_i \end{cases}$$

$$y' = \sum_{i=0}^{n-1} w_i' x_i'$$

$$E\{y'\} = E\left\{\sum_{i=0}^{n-1} w_i' x_i'\right\} = \sum_{i=0}^{n-1} E\{w_i'\} E\{x_i'\}$$

$$= \sum_{i=0}^{n-1} p_i c^{(i)} x_i = \sum_{i=0}^{n-1} w_i x_i = y$$

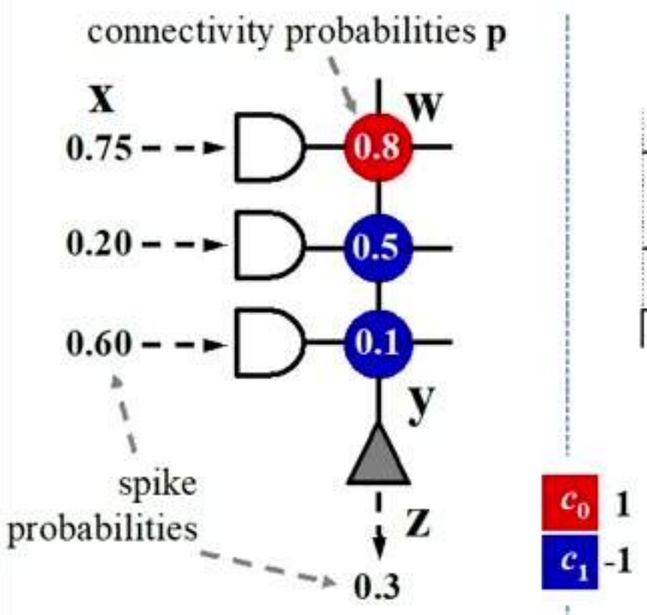
$$E\{z'\} = P(y' \geq 0) = \frac{1}{2}\left[1 + erf\left(\frac{-\mu_{y'}}{\sqrt{2}\sigma_{y'}}\right)\right]$$

MNIST Accuracy:

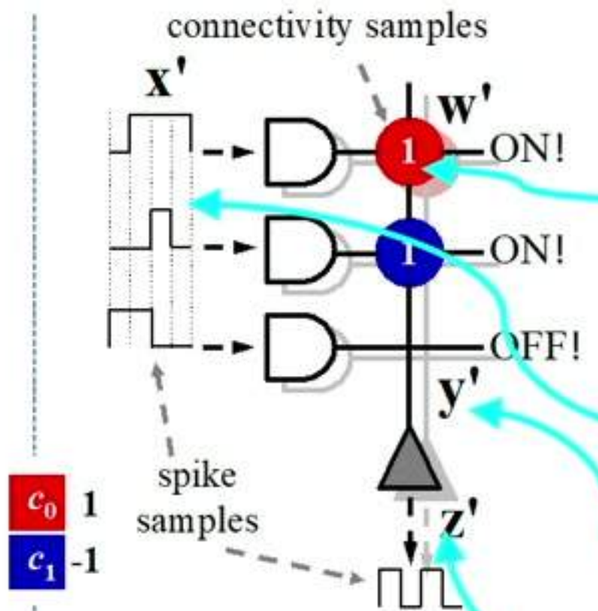
- **95.27% in Caffe**
- **90.04% @1 NN copy & 1 *spf* in TrueNorth**

(NN - neural networks; *spf* – spikes per frame)

Deploy a Network on TrueNorth



(a) Tea learning



(b) Tea deploying

Traditional floating-point precision

Binary/low integer precision *sampled by float-point probability*

$$p_i = w_i / c^{(i)}, c^{(i)} \in \{c_0, c_1\}$$

$$\begin{cases} P(w_i' = c^{(i)}) = p_i \\ P(w_i' = 0) = 1 - p_i \end{cases}$$

$$\begin{cases} P(x_i' = 1) = x_i \\ P(x_i' = 0) = 1 - x_i \end{cases}$$

$$y' = \sum_{i=0}^{n-1} w_i' x_i'$$

$$E\{y'\} = E\left\{\sum_{i=0}^{n-1} w_i' x_i'\right\} = \sum_{i=0}^{n-1} E\{w_i'\} E\{x_i'\}$$

$$= \sum_{i=0}^{n-1} p_i c^{(i)} x_i = \sum_{i=0}^{n-1} w_i x_i = y$$

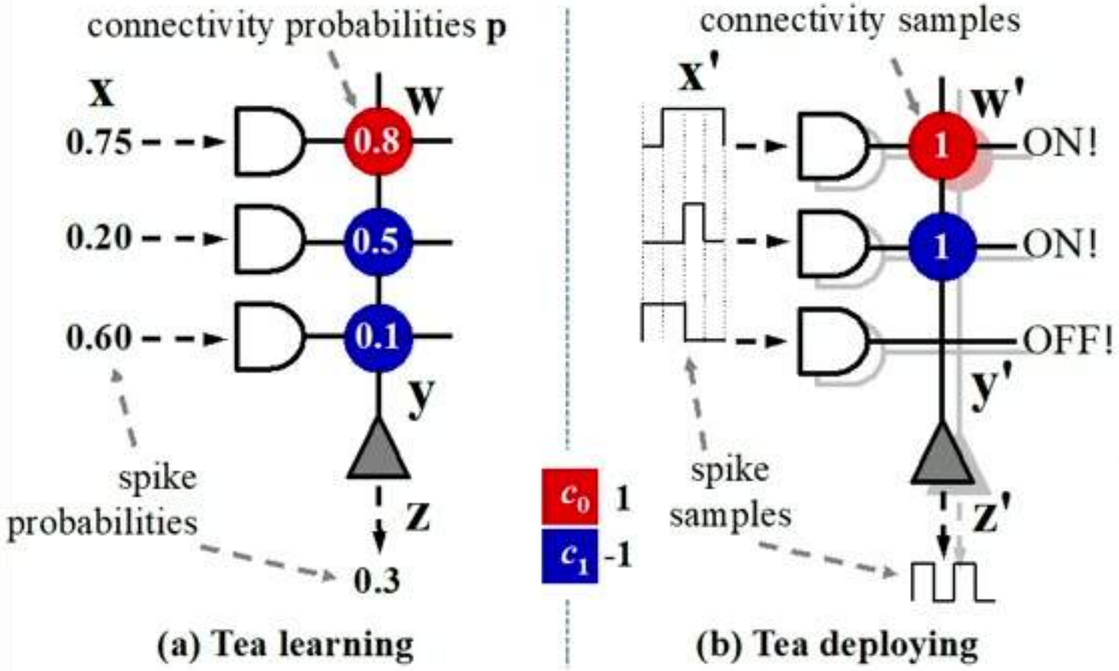
$$E\{z'\} = P(y' \geq 0) = \frac{1}{2}\left[1 + erf\left(\frac{-\mu_{y'}}{\sqrt{2}\sigma_{y'}}\right)\right]$$

MNIST Accuracy:

- **95.27% in Caffe**
- **90.04% @1 NN copy & 1 *spf* in TrueNorth**
- **92.74% @ 1 NN copy & 4 *spf* in TrueNorth**
- **94.63% @16 NN copies & 1 *spf* in TrueNorth**

(NN - neural networks; *spf* – spikes per frame)

Minimizing Deployment Variance



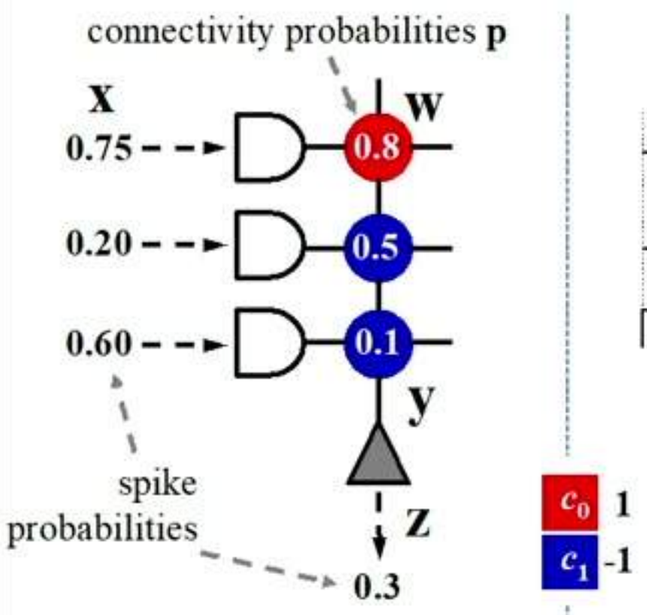
$$\Delta y = y' - y = \sum_{i=0}^{n-1} w_i' x_i' - \sum_{i=0}^{n-1} w_i x_i$$

$$E\{\Delta y\} = 0$$

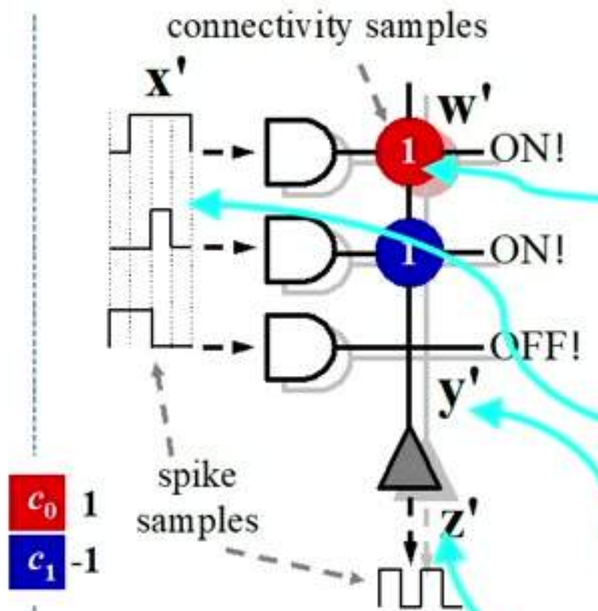
$$var\{\Delta y\} = \sum_{i=0}^{n-1} var\{w_i' x_i'\}$$

Unbiased approximation w/ variance is affected by both synaptic and spiking randomness.

Deploy a Network on TrueNorth



(a) Tea learning



(b) Tea deploying

Traditional floating-point precision

Binary/low integer precision *sampled by float-point probability*

$$p_i = w_i / c^{(i)}, c^{(i)} \in \{c_0, c_1\}$$

$$\begin{cases} P(w_i' = c^{(i)}) = p_i \\ P(w_i' = 0) = 1 - p_i \end{cases}$$

$$\begin{cases} P(x_i' = 1) = x_i \\ P(x_i' = 0) = 1 - x_i \end{cases}$$

$$y' = \sum_{i=0}^{n-1} w_i' x_i'$$

$$E\{y'\} = E\left\{\sum_{i=0}^{n-1} w_i' x_i'\right\} = \sum_{i=0}^{n-1} E\{w_i'\} E\{x_i'\}$$

$$= \sum_{i=0}^{n-1} p_i c^{(i)} x_i = \sum_{i=0}^{n-1} w_i x_i = y$$

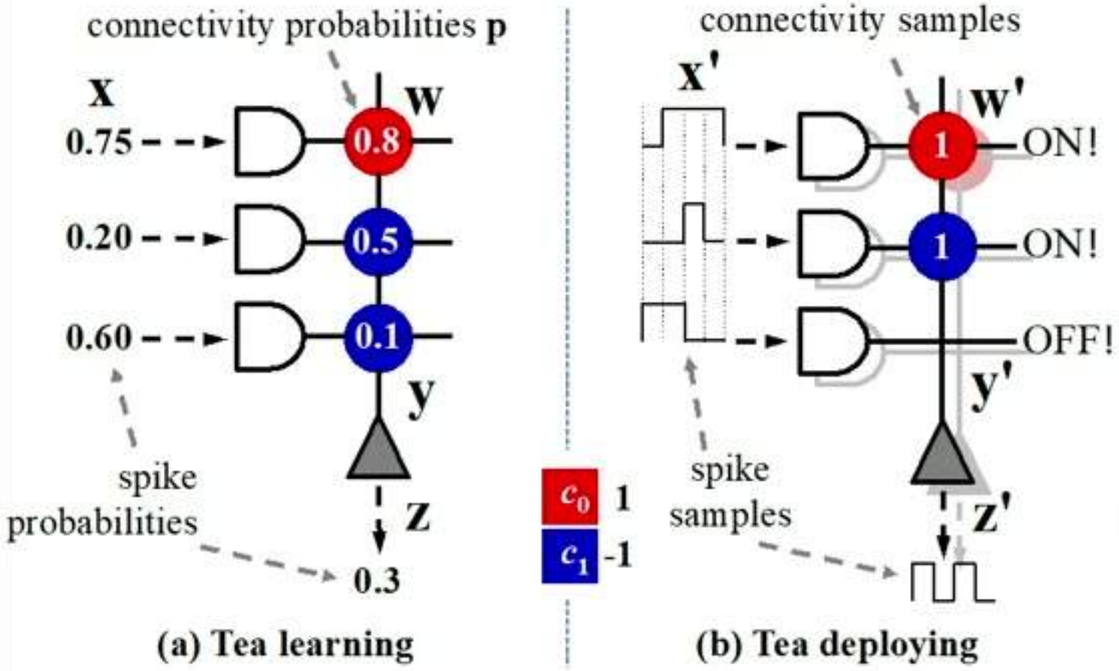
$$E\{z'\} = P(y' \geq 0) = \frac{1}{2}\left[1 + erf\left(\frac{-\mu_{y'}}{\sqrt{2}\sigma_{y'}}\right)\right]$$

MNIST Accuracy:

- **95.27% in Caffe**
- **90.04% @1 NN copy & 1 *spf* in TrueNorth**
- **92.74% @ 1 NN copy & 4 *spf* in TrueNorth**
- **94.63% @16 NN copies & 1 *spf* in TrueNorth**

(NN - neural networks; *spf* – spikes per frame)

Minimizing Deployment Variance



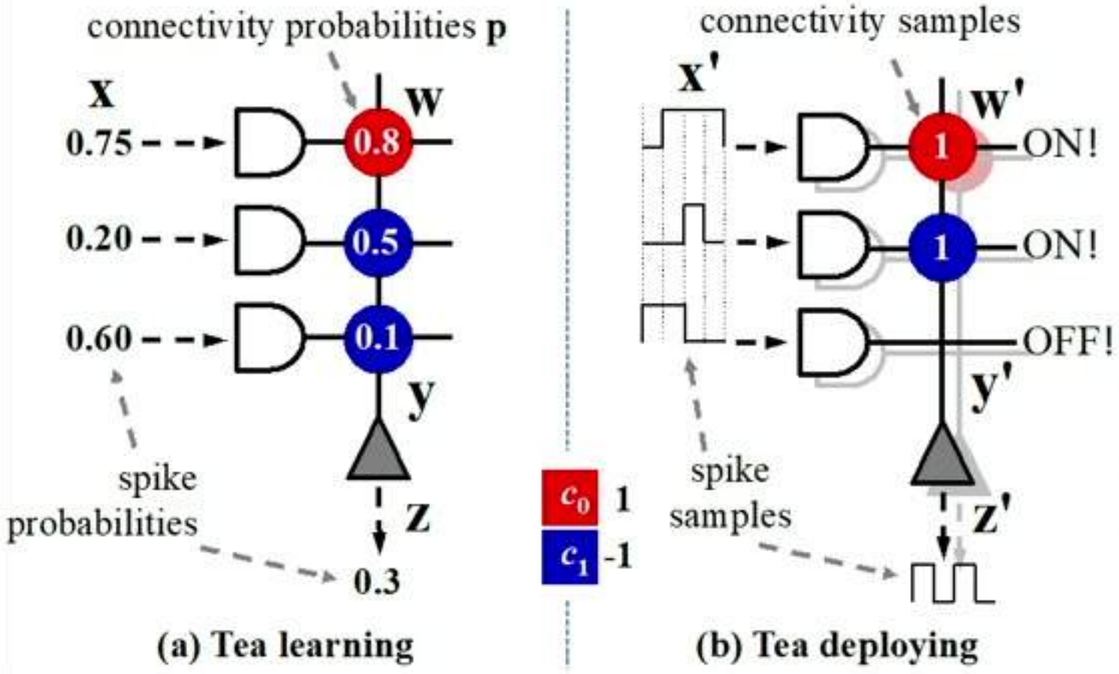
$$\Delta y = y' - y = \sum_{i=0}^{n-1} w_i' x_i' - \sum_{i=0}^{n-1} w_i x_i$$

$$E\{\Delta y\} = 0$$

$$var\{\Delta y\} = \sum_{i=0}^{n-1} var\{w_i' x_i'\}$$

Unbiased approximation w/ variance is affected by both synaptic and spiking randomness.

Minimizing Deployment Variance



$$\Delta y = y' - y = \sum_{i=0}^{n-1} w_i' x_i' - \sum_{i=0}^{n-1} w_i x_i$$

$$E\{\Delta y\} = 0$$

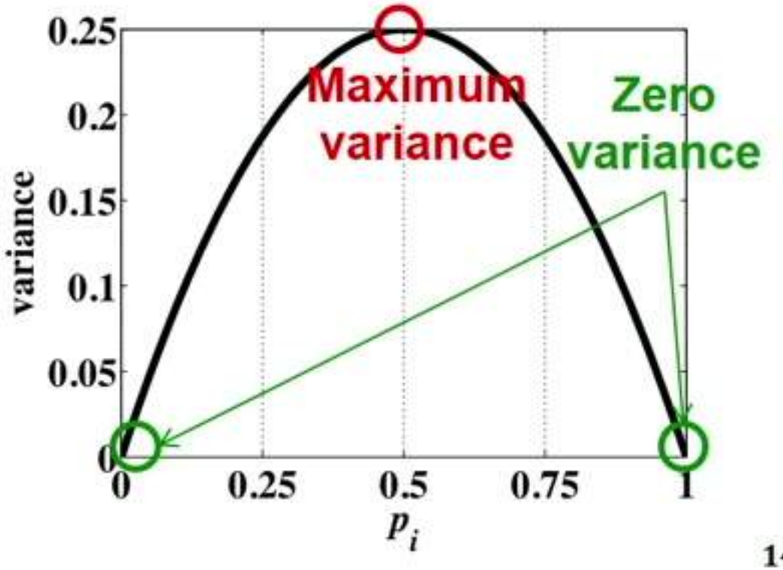
$$var\{\Delta y\} = \sum_{i=0}^{n-1} var\{w_i' x_i'\}$$

Unbiased approximation w/ variance is affected by both synaptic and spiking randomness.

Spiking randomness:
determined by external database

Synaptic randomness:

$$var\{w_i'\} = E\{(w_i')^2\} - E\{w_i'\}^2 = p_i(1-p_i)$$

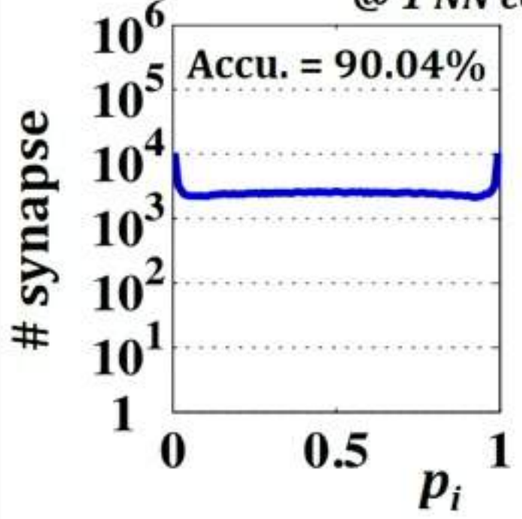


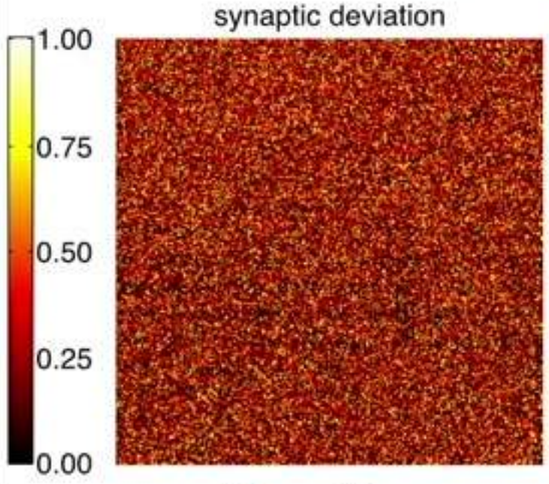
Minimizing Deployment Variance

Minimization target: $\hat{E}(\mathbf{w}) = E_D(\mathbf{w})$

Data loss

@ 1 NN copy & 1 spf





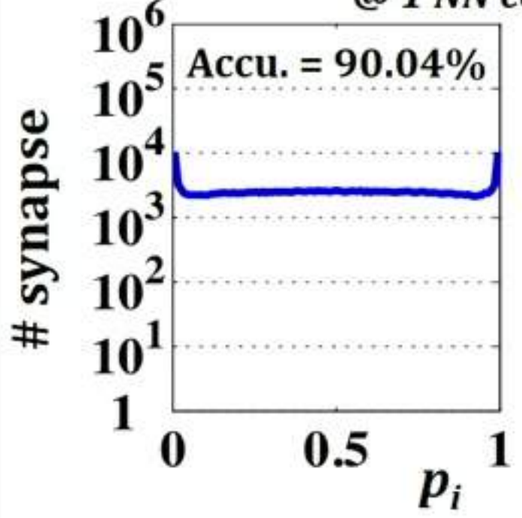
Baseline

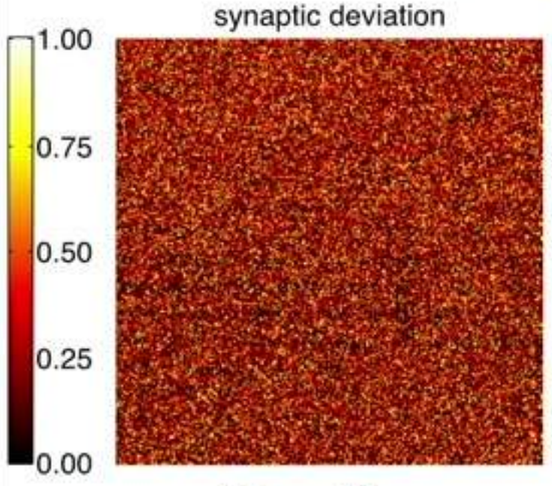
Minimizing Deployment Variance

Minimization target: $\hat{E}(\mathbf{w}) = E_D(\mathbf{w}) + \lambda \cdot E_P(\mathbf{p})$

Data loss **Probability regularization**

@ 1 NN copy & 1 spf

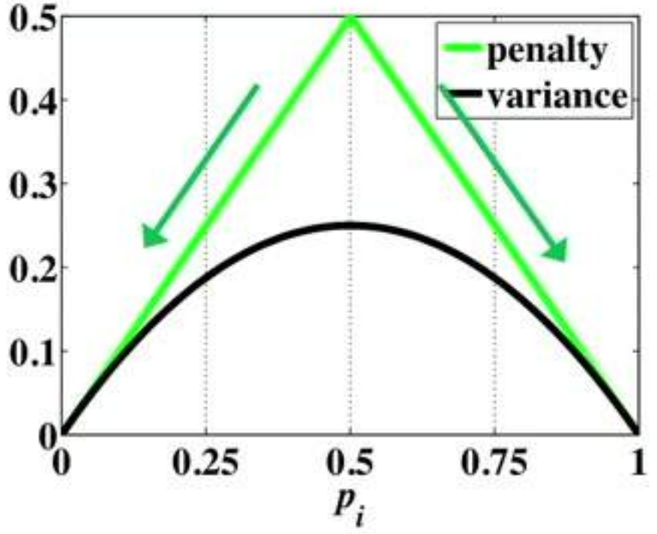




Baseline

$$E_P(\mathbf{p}) = \left\| \left| \mathbf{p} - a \right| - b \right\|$$
$$= \sum_{i=1}^{M} \left| \left| p_i - a \right| - b \right|$$

$a = b = 0.5$

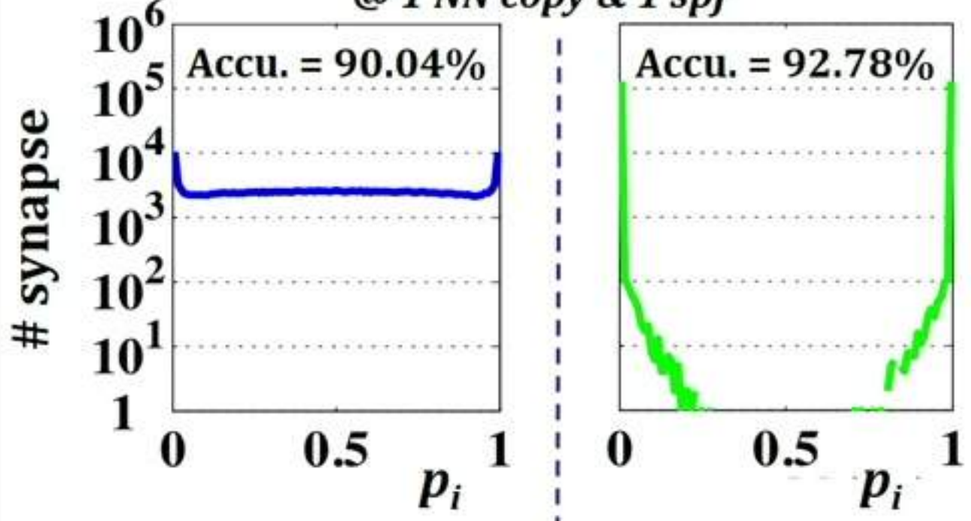


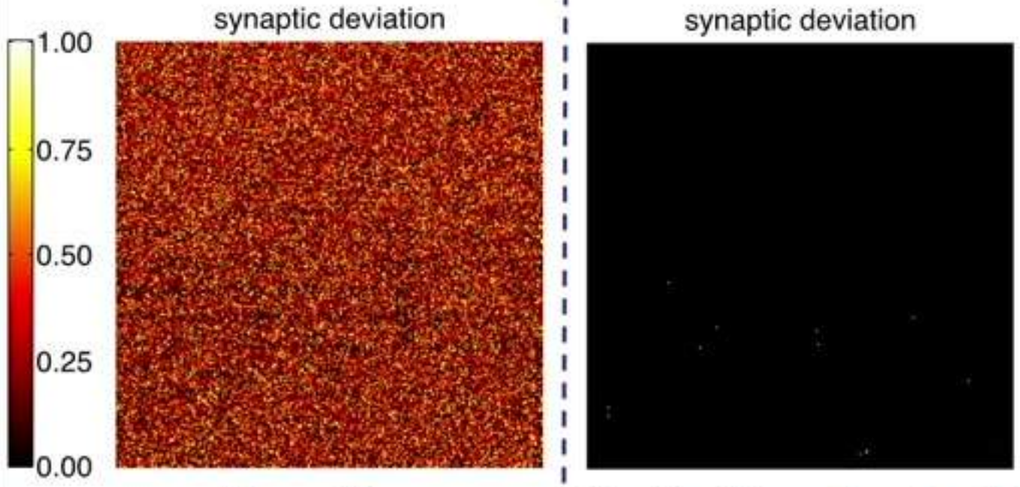
Minimizing Deployment Variance

Minimization target: $\hat{E}(\mathbf{w}) = E_D(\mathbf{w}) + \lambda \cdot E_P(\mathbf{p})$

Data loss **Probability regularization**

@ 1 NN copy & 1 spf

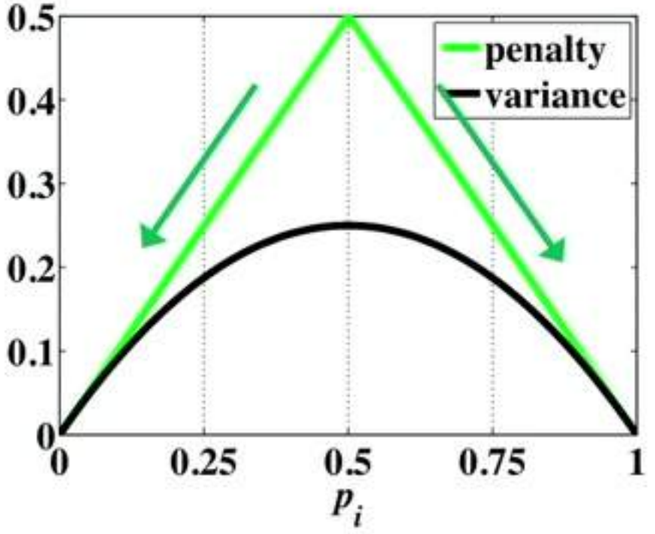




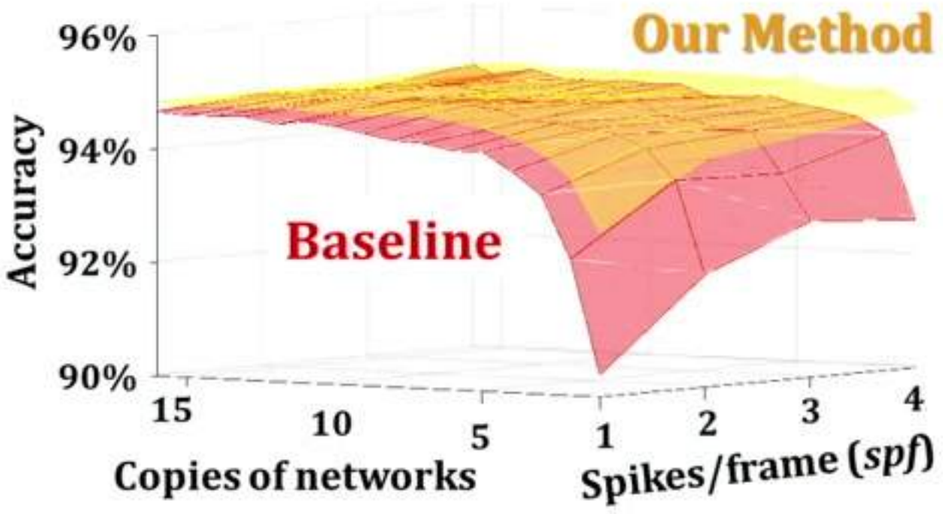
Baseline **Prob. Regularization**

$$E_P(\mathbf{p}) = \left\| \left| \mathbf{p} - a \right| - b \right\| = \sum_{i=1}^{M} \left| \left| p_i - a \right| - b \right|$$

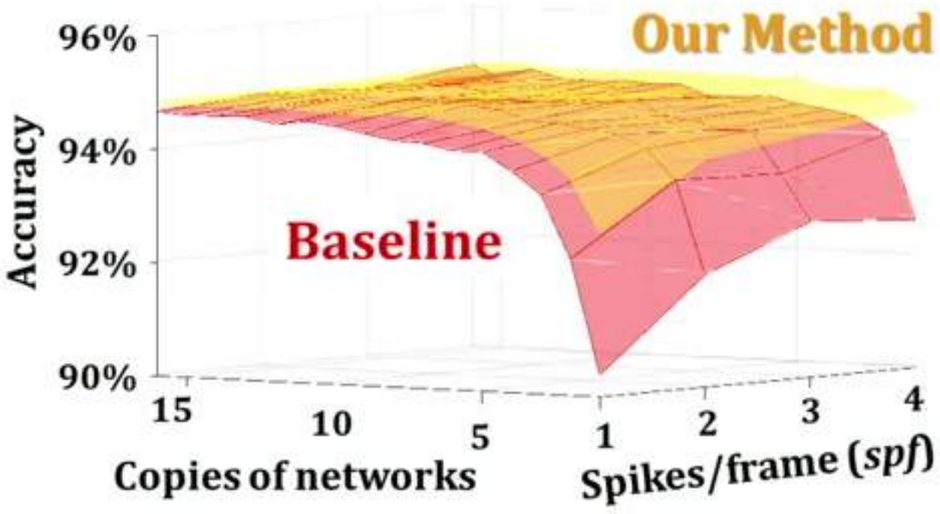
$a = b = 0.5$

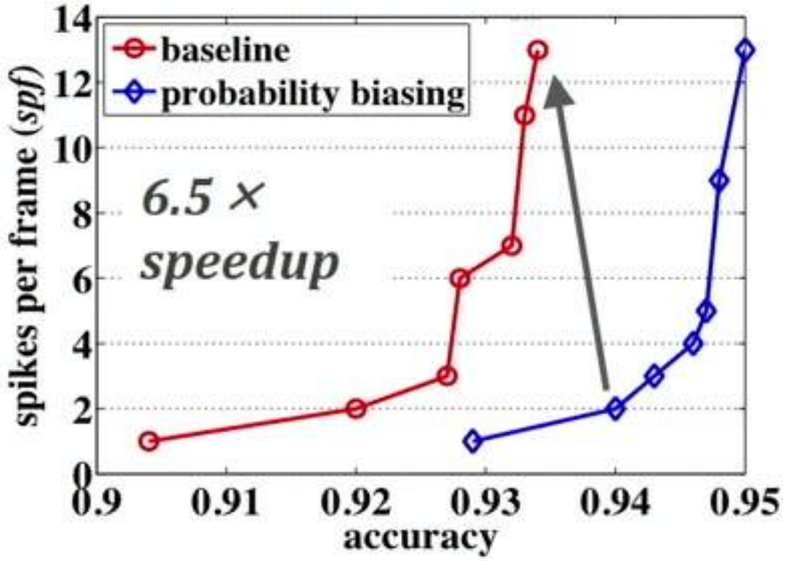


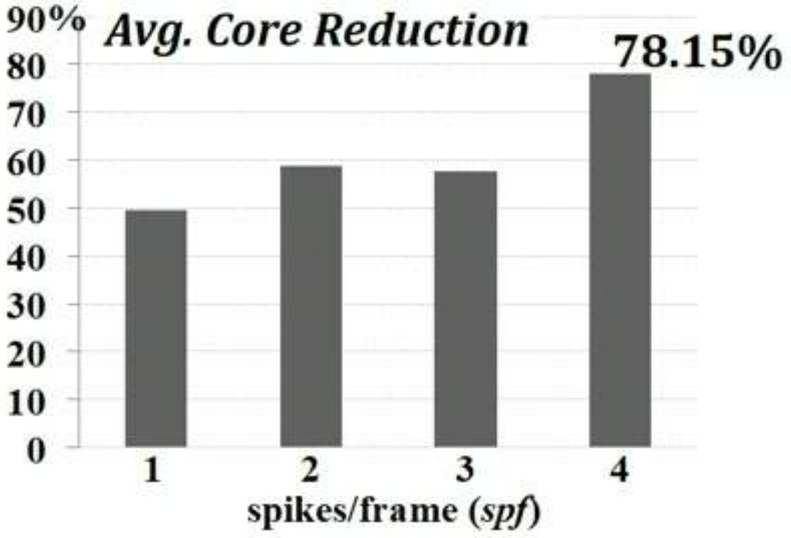
Experiments

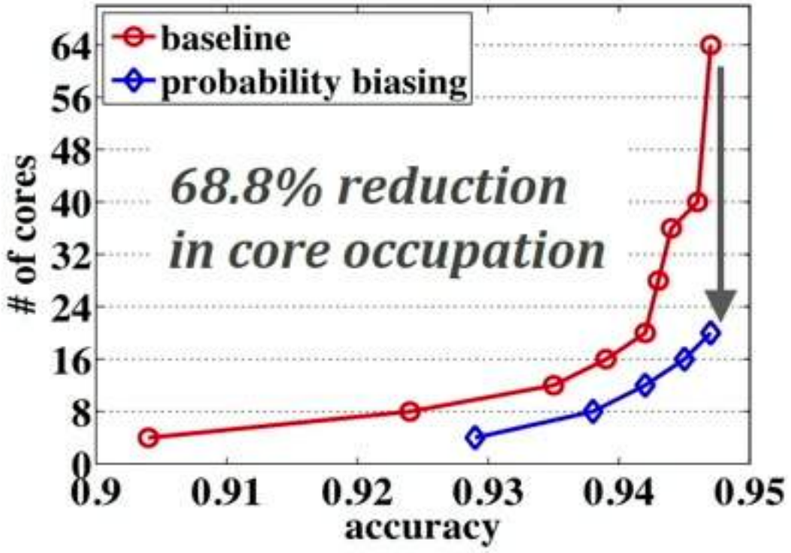


Experiments





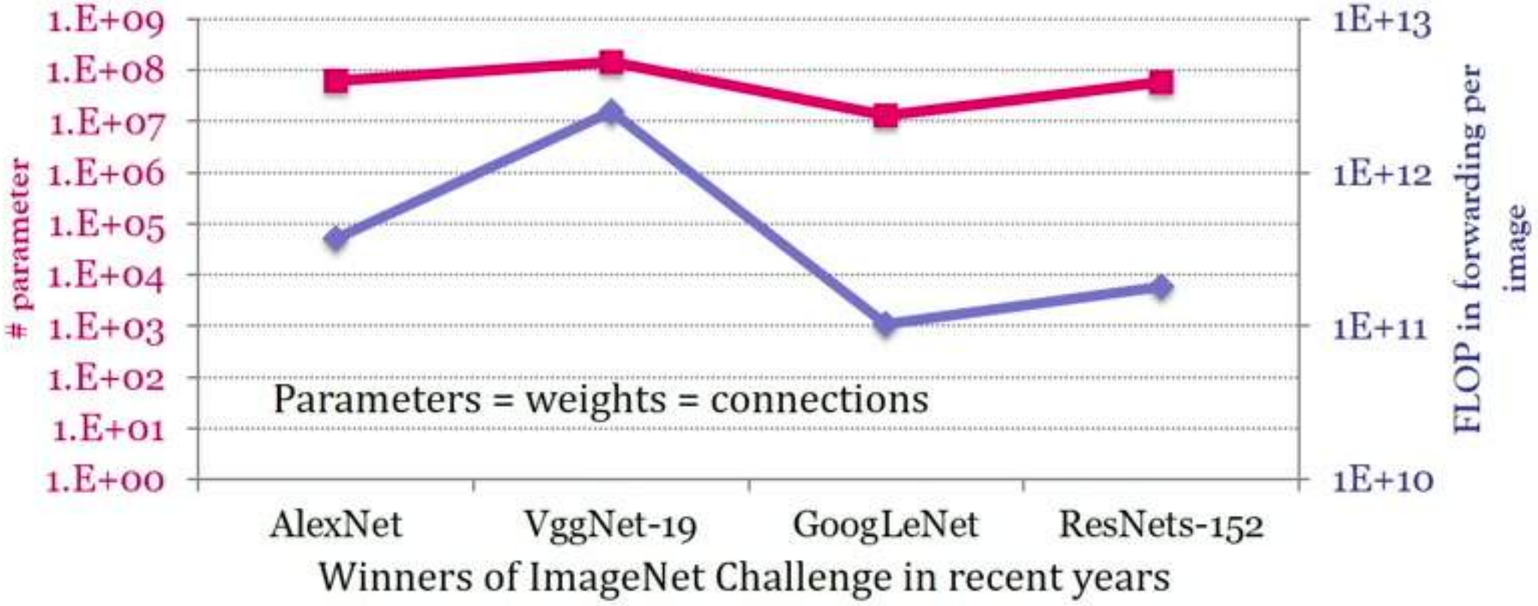




Learning Structured Sparsity in Deep Neural Networks

Duke
UNIVERSITY

Complexity of Deep Neural Networks



Fewer parameters, fewer computation (FLOP: Floating Point Operation)

How to reduce the number of parameters in DNN so as to reduce FLOP, meanwhile maintain the classification accuracy?

Non-structurally Sparse DNNs

- State-of-the-art methods to reduce the number of parameters
  - Weight regularization (L1-norm)

Sparsity: the ratio of zeros

AlexNet, B. Liu, *et al.*, CVPR 2015

Layer	conv1	conv2	conv3	conv4	conv5
Sparsity%	0.927	0.95	0.951	0.942	0.938
Theoretical speedup	2.61	7.14	16.12	12.42	10.77

Non-structurally Sparse DNNs

- State-of-the-art methods to reduce the number of parameters
  - Weight regularization (L1-norm)
  - Connection pruning

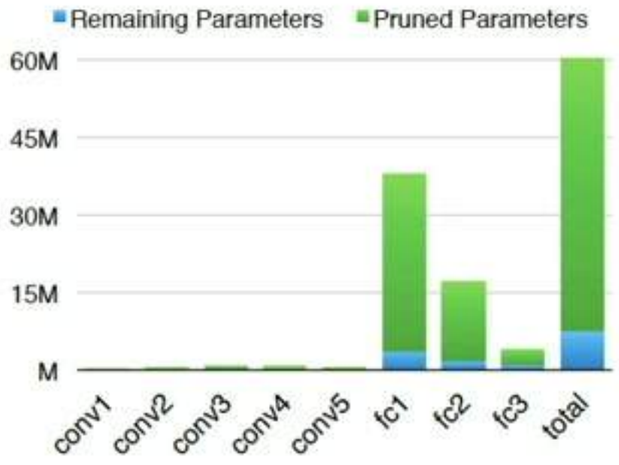
Sparsity: the ratio of zeros

AlexNet, B. Liu, *et al.*, CVPR 2015

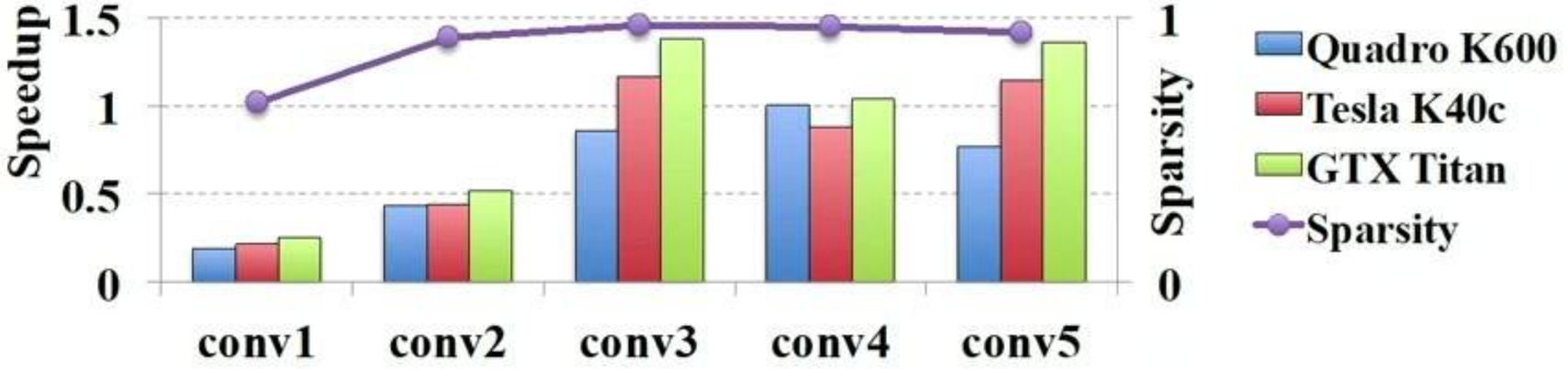
Layer	conv1	conv2	conv3	conv4	conv5
Sparsity%	0.927	0.95	0.951	0.942	0.938
Theoretical speedup	2.61	7.14	16.12	12.42	10.77

AlexNet, S. Han, *et al.*, NIPS 2015

Layer	Weights	FLOP	Act%	Remained Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10%
Total	61M	1.5B	54%	**11%**	**30%**

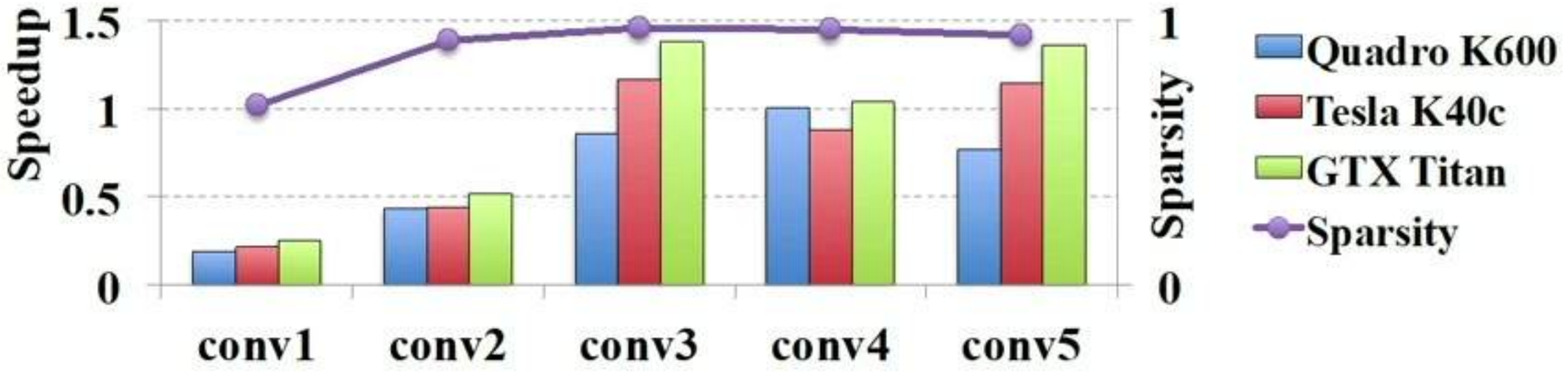


Theoretical Speedup ≠ Practical Speedup

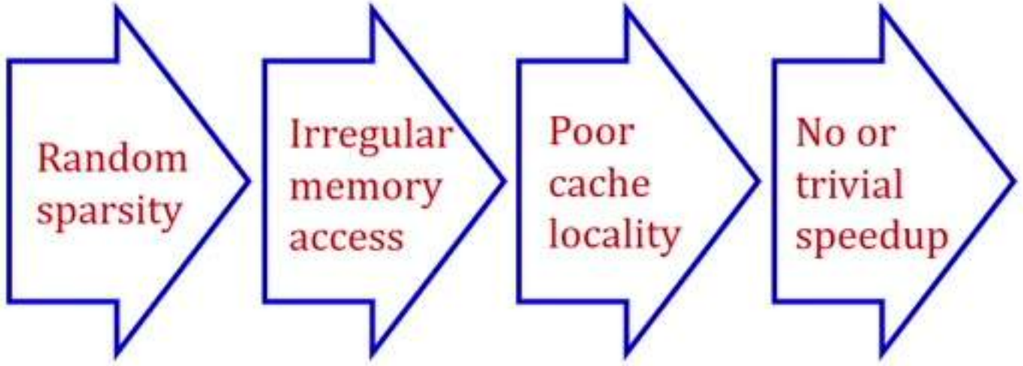


Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.

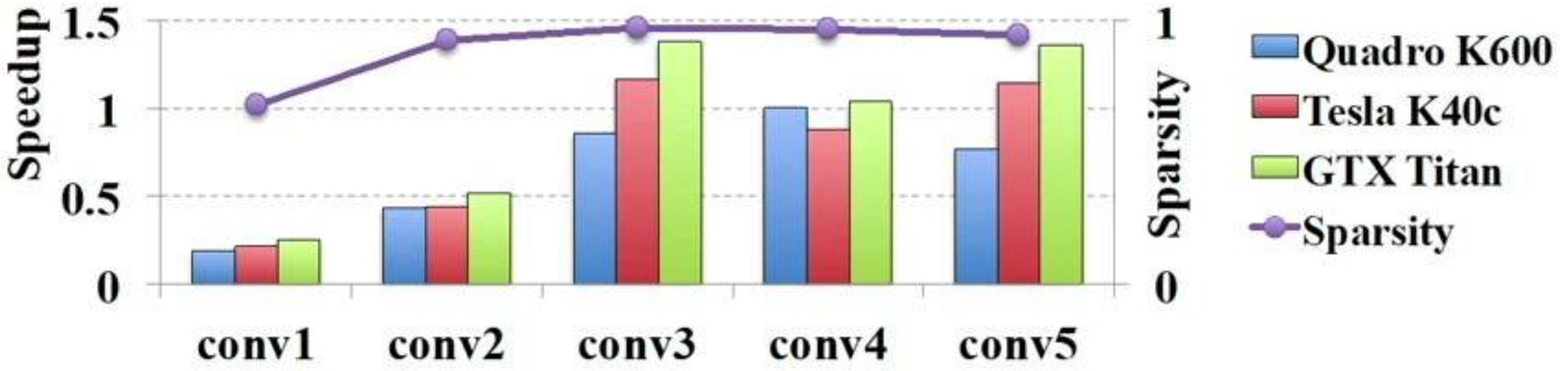
Theoretical Speedup ≠ Practical Speedup



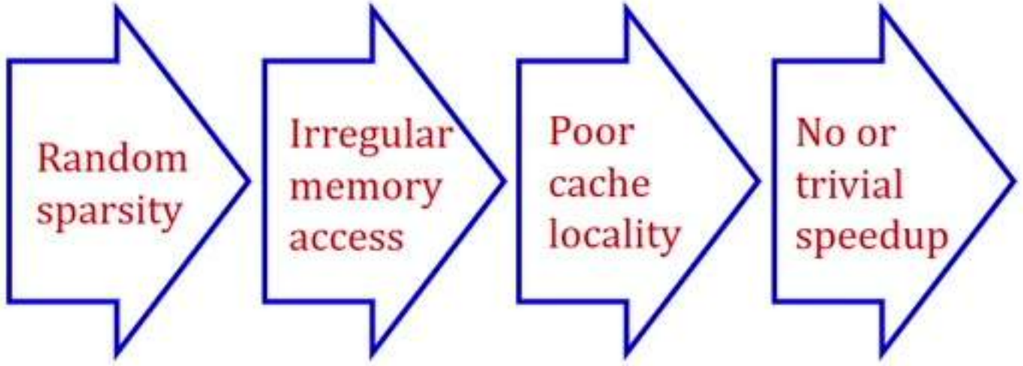
Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.



Theoretical Speedup ≠ Practical Speedup



Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.



Hardcoding nonzero weights in source code in B. Liu, etc., CVPR 2015

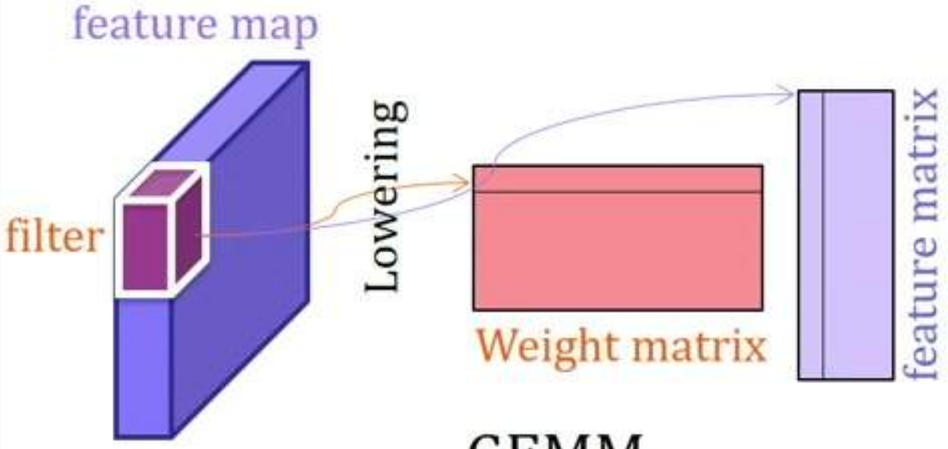
Software customization

Hardware customization

Customizing an EIE chip accelerator for compressed DNN in S. Han ISCA 2017

Theoretical Speedup ≈ Practical Speedup

Example: Removing rows/columns in GEMM (row/column-wise sparsity)



GEMM

GEneral Matrix Matrix Multiplication

Non-structured sparsity

conv2_1: weight sparsity (col:8.7% row:19.5% elem:94.6%)

Structured sparsity

conv2_1: weight sparsity (col:75.2% row:21.9% elem:91.5%)

5.17X speedup

Structured sparsity → Regular memory access → Good cache locality → Great speedup → Higher speedup with software or hardware customization

Structured Sparsity Regularization

- Group Lasso regularization in ML model

$$\arg\min_{\mathbf{w}}\left\{E(\mathbf{w})\right\} = \arg\min_{\mathbf{w}}\left\{E_D(\mathbf{w}) + \lambda_g \cdot R_g(\mathbf{w})\right\}$$

$$\Updownarrow$$

$$\arg\min_{\mathbf{w}}\left\{E(\mathbf{w})\right\} = \arg\min_{\mathbf{w}}\left\{E_D(\mathbf{w})\right\}$$

$$s.t.\ R_g(\mathbf{w}) \le \eta_g$$

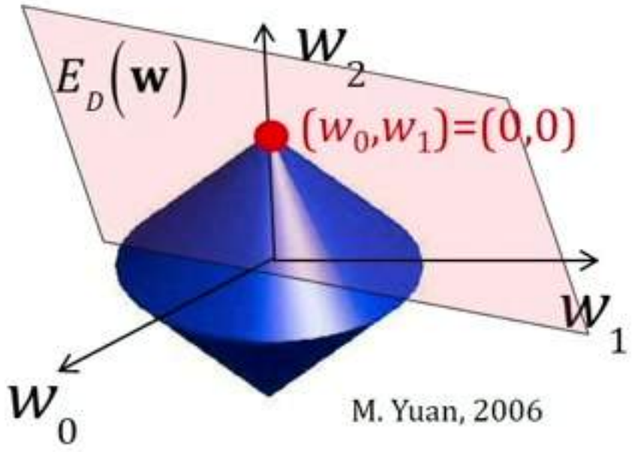
Many groups will be zeros

$$R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g,$$

$$||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$$

Example:

$$R_g\left(\underbrace{w_0, w_1}_{\text{group 1}}, \underbrace{w_2}_{\text{group 2}}\right) = \sqrt{w_0^2 + w_1^2} + \sqrt{w_2^2} \le \eta_g$$



M. Yuan, 2006

SSL: Structured Sparsity Learning

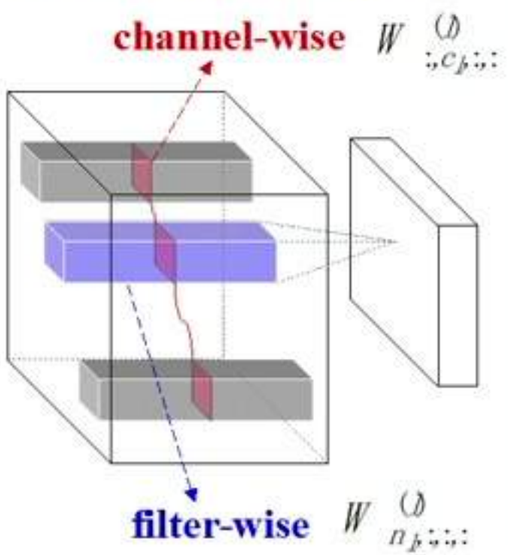
- Group Lasso regularization in DNNs:

$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda \cdot R(\boldsymbol{W}) + \lambda_g \cdot \sum_{l=1}^{L} R_g\left(\boldsymbol{W}^{(l)}\right)$$

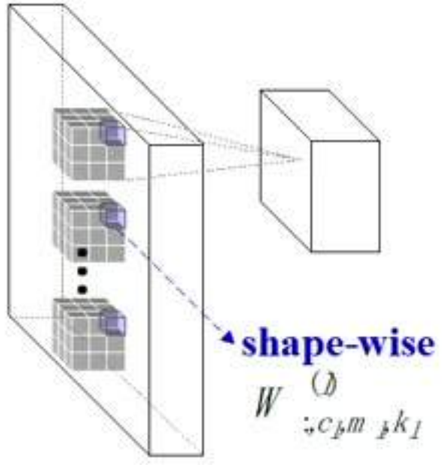
$$\boxed{R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g}$$

Learned structured sparsity is determined by the way of splitting groups

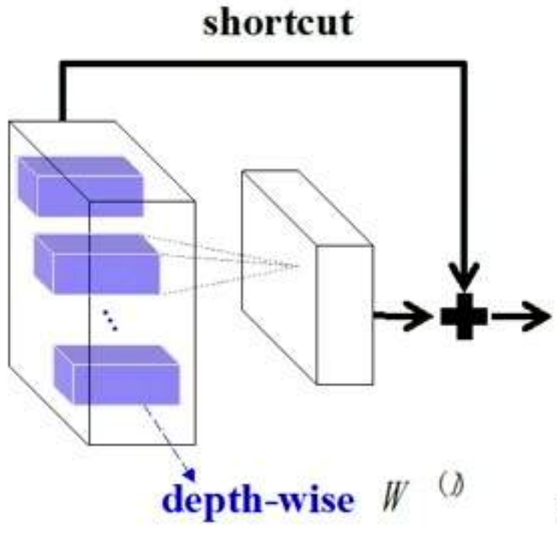
Penalize unimportant filters and channels



Learn filter shapes



Learn the depth of layers



Experiments – Penalizing unimportant filters and channels

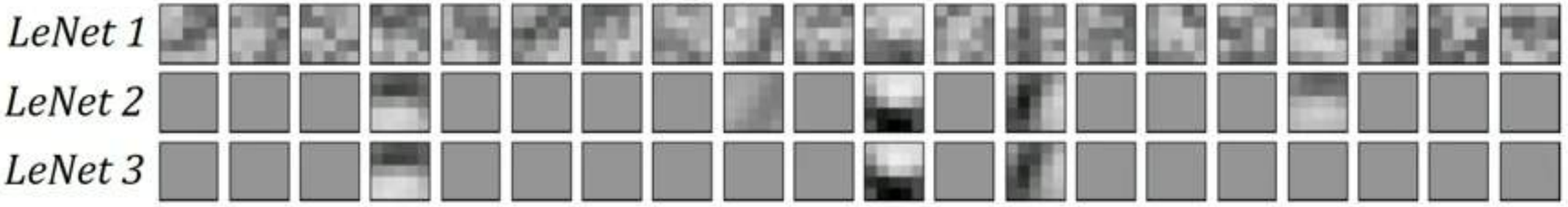
LeNet on MNIST

Table 1: Results after penalizing unimportant filters and channels in *LeNet*

LeNet #	Error	Filter # §	Channel # §	FLOP §	Speedup §
1 (*baseline*)	0.9%	20—50	1—20	100%—100%	1.00×—1.00×
2	0.8%	5—19	1—4	25%—7.6%	1.64×—5.23×
3	1.0%	3—12	1—3	15%—3.6%	1.99×—7.44×

§In the order of *conv1*—*conv2*

conv1 filters (gray level 128 represents zero)



Fewer but more natural patterns

Experiments – Learning smaller filter shapes

Table 2: Results after learning filter shapes in *LeNet*

LeNet #	Error	Filter size §	Channel #	FLOP	Speedup
1 (*baseline*)	0.9%	25—500	1—20	100%—100%	1.00×—1.00×
4	0.8%	21—41	1—2	8.4%—8.2%	2.33×—6.93×
5	1.0%	7—14	1—1	1.4%—2.8%	5.19×—10.82×

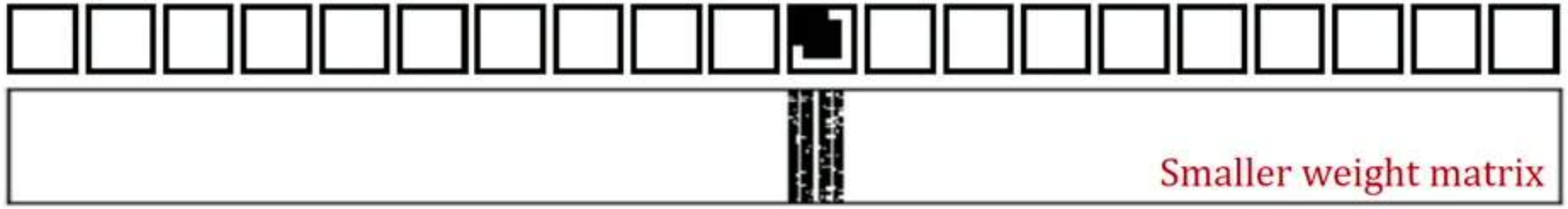
§ The sizes of filters after removing zero shape fibers, in the order of *conv1—conv2*

Learned shapes of conv1 filters:

LeNet 1 *LeNet 4* *LeNet 5*

Learned shape of conv2 filters @ *LeNet 5*

3D 20x5x5 filters is regularized to 2D filters!



SSL can efficiently learn DNNs with smaller filters without accuracy loss

Experiments – Learning smaller dense weight matrix

Filter-wise sparsity = row-wise sparsity
Shape-wise sparsity = column-wise sparsity
⟶ Smaller dense weight matrix

Table 3: Learning row-wise and column-wise sparsity of *ConvNet* on CIFAR-10

ConvNet #	Error	Row sparsity [§]	Column sparsity [§]	Speedup [§]
1 (*baseline*)	17.9%	12.5%–0%–0%	0%–0%–0%	1.00×–1.00×–1.00×
2	17.9%	50.0%–28.1%–1.6%	0%–59.3%–35.1%	1.43×–3.05×–1.57×
3	16.9%	31.3%–0%–1.6%	0%–42.8%–9.8%	1.25×–2.01×–1.18×

[§]in the order of *conv1–conv2–conv3*

Figure 5: Learned *conv1* filters in *ConvNet 1* (top), *ConvNet 2* (middle) and *ConvNet 3* (bottom)

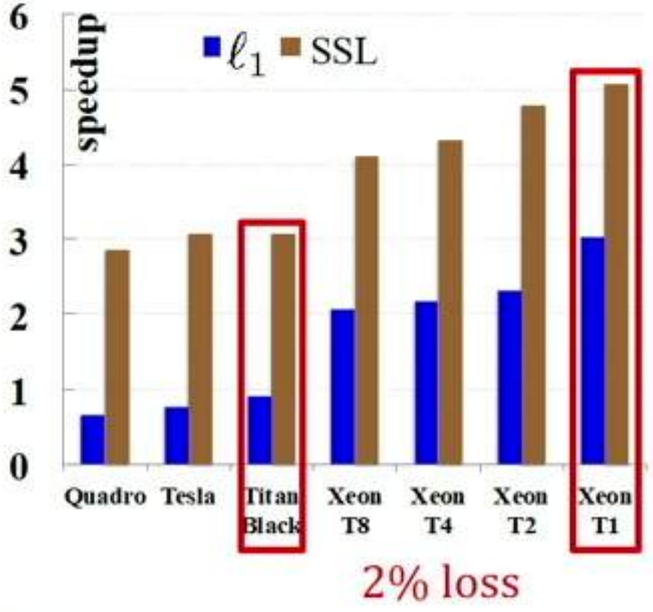
SSL can efficiently learn DNNs with smaller but dense weight matrix which has good locality

Experiments – AlexNet@ImageNet

Learning row-wise and column-wise sparsity:

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

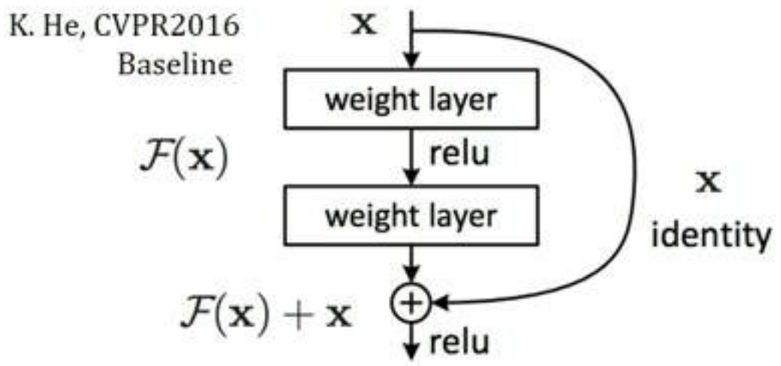
#	Method	Top1 err.	Statistics	conv1	conv2	conv3	conv4	conv5	
1	ℓ_1	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%	2% loss
			CPU ×	0.80	2.91	4.84	3.83	2.76	
			GPU ×	0.25	0.52	1.38	1.04	1.36	
2	SSL	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%	
			row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
			CPU ×	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	ℓ_1	42.51%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%	No loss
			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	

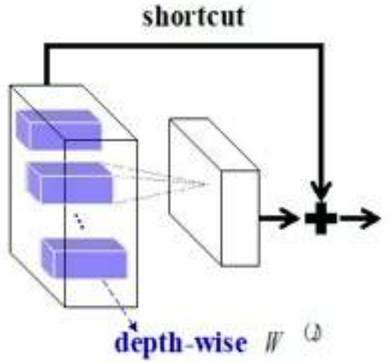


1. Non-structured sparsity method even slows down in some layers
2. layer-wise 5.1X /3.1X on CPU/GPU with 2% accuracy loss
3. layer-wise 1.4X on both CPU and GPU w/o accuracy loss
4. Higher speedups than non-structured speedups

Experiments – Regularizing the depth of DNNs

Experiments of ResNets on CIFAR-10

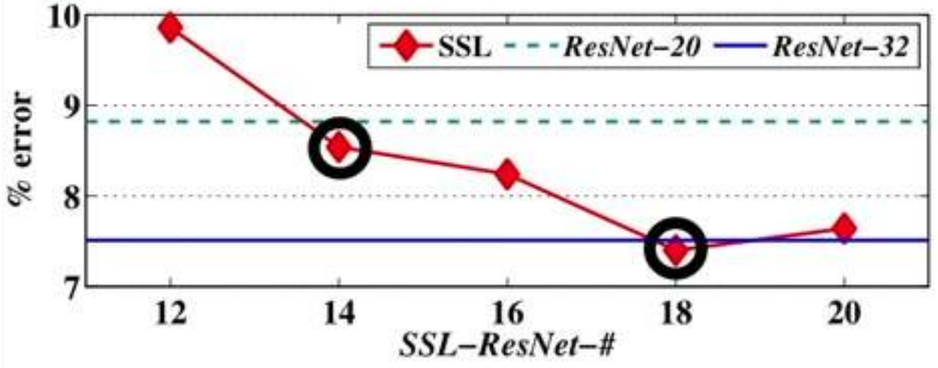


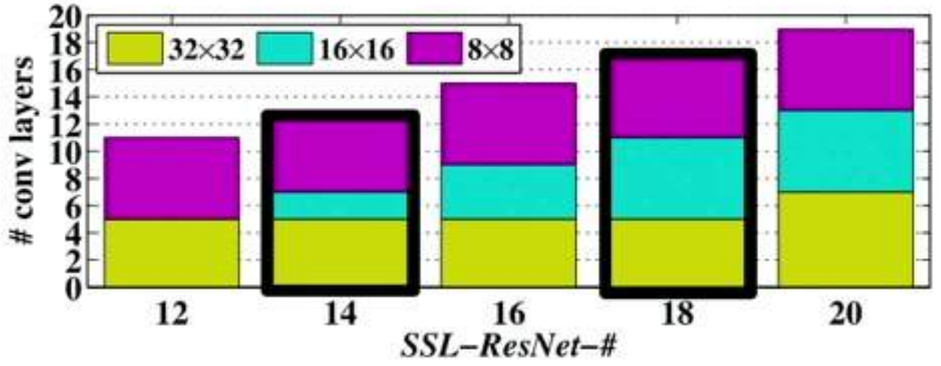


ResNet-20/32: baseline with 20/32 layers

SSL-ResNet-#: Ours with # layers after learning depth of ResNet-20

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



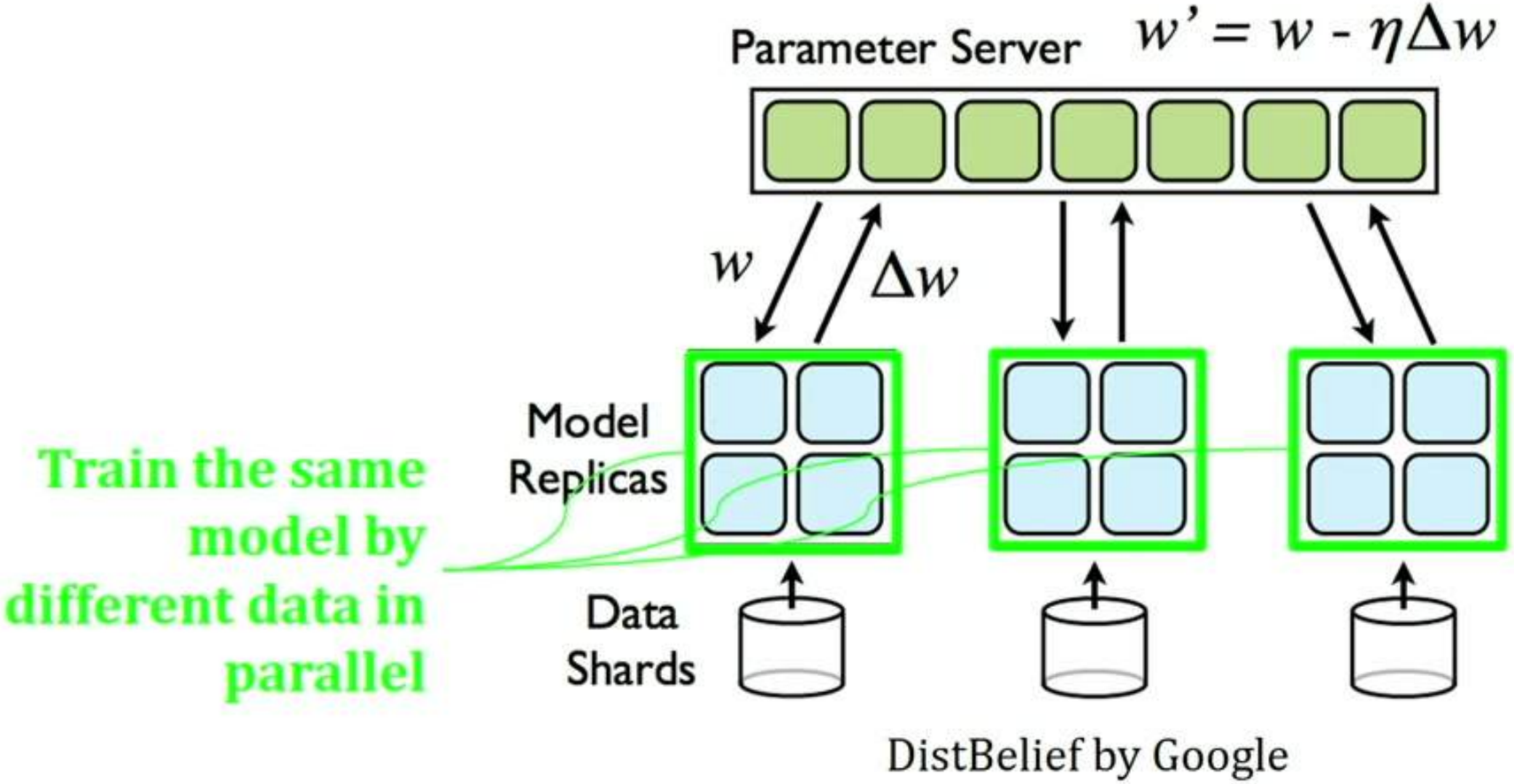


TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning

Duke UNIVERSITY

EI

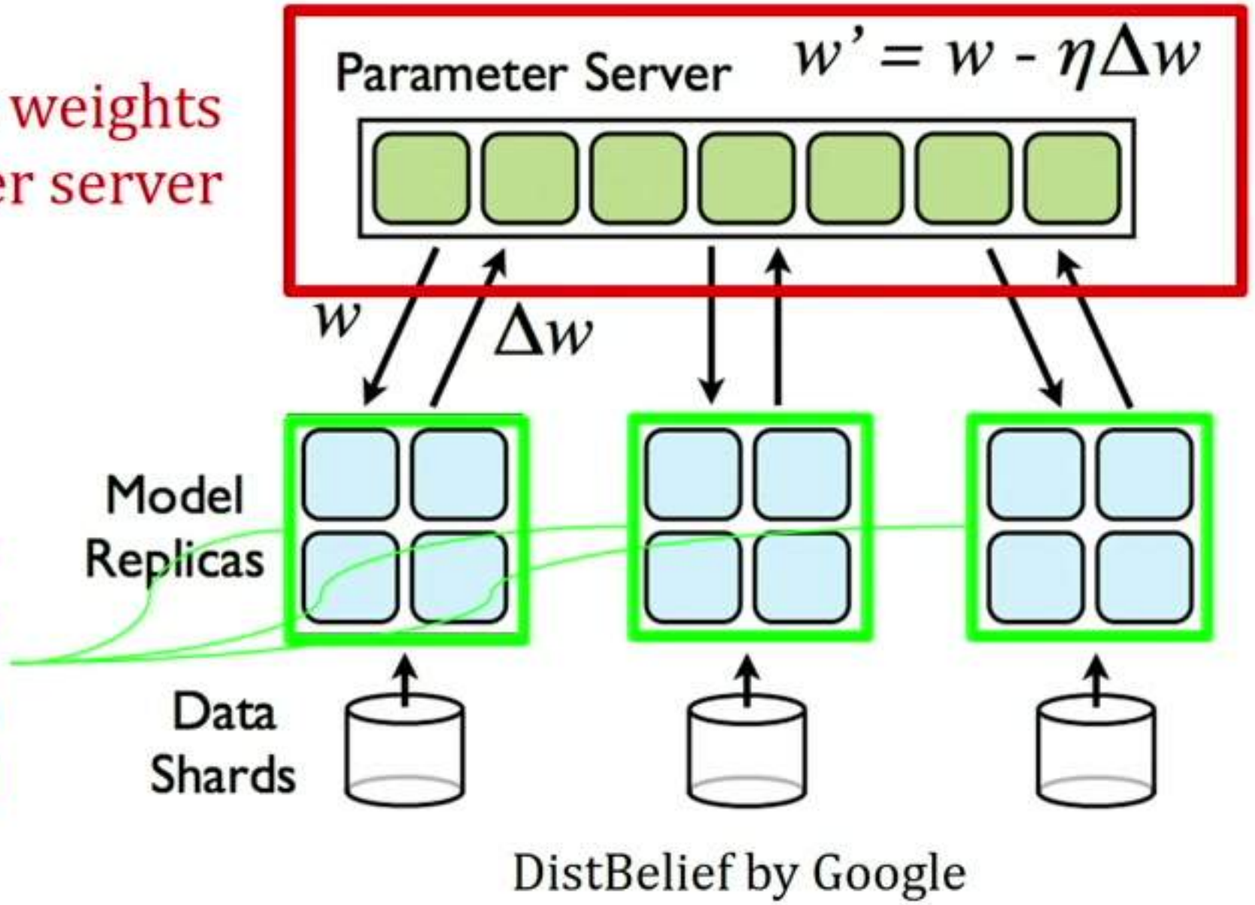
Distributed Deep Learning



Distributed Deep Learning

When parallelism increase, communication is the bottleneck

Synchronize weights in parameter server



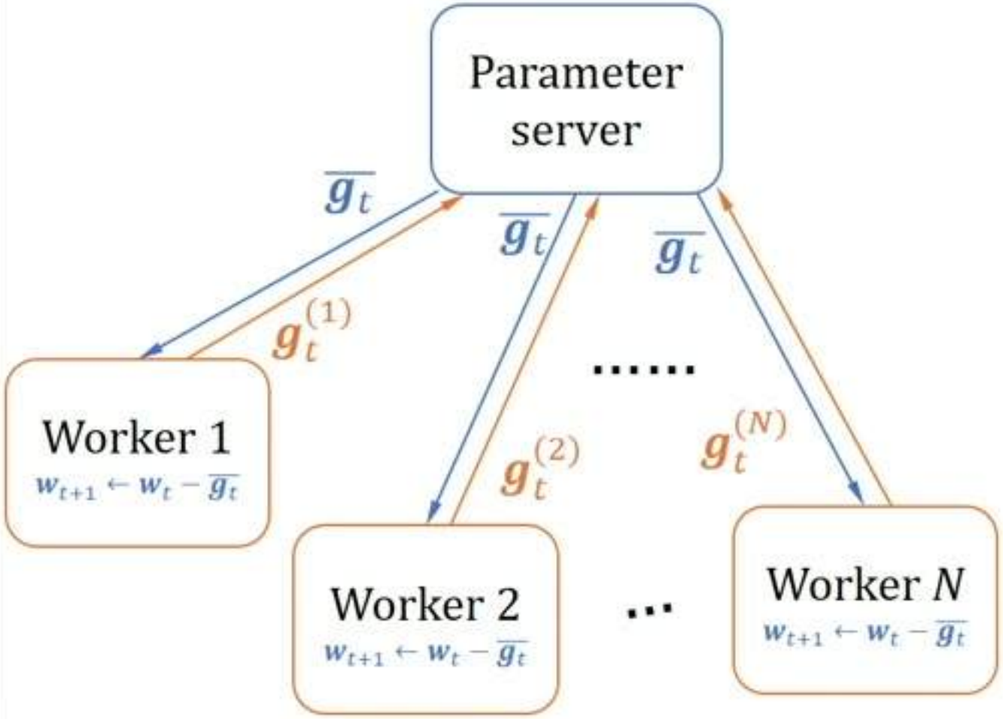
Train the same model by different data in parallel

DistBelief by Google

TernGrad – distributed training with ternary gradients

Key ideas to reduce communication:

1. Randomly quantize gradients to only three levels (0, ±1)
2. The expectations of quantized gradients equal original values
3. Exchange quantized gradients instead of floating weights



Algorithm 1 *TernGrad*: distributed SGD training using ternary gradients.

Worker : $i = 1, ..., N$

1 Input $\boldsymbol{z}_t^{(i)}$, a part of a mini-batch of training samples $\boldsymbol{z}_t$

2 Compute gradients $\boldsymbol{g}_t^{(i)}$ under $\boldsymbol{z}_t^{(i)}$

3 Ternarize gradients to $\tilde{\boldsymbol{g}}_t^{(i)} = ternarize(\boldsymbol{g}_t^{(i)})$

4 Push ternary $\tilde{\boldsymbol{g}}_t^{(i)}$ to the server

5 Pull averaged gradients $\overline{\boldsymbol{g}_t}$ from the server

6 Update parameters $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \cdot \overline{\boldsymbol{g}_t}$

Server :

7 Average ternary gradients $\overline{\boldsymbol{g}_t} = \sum_i \tilde{\boldsymbol{g}}_t^{(i)} / N$

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

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

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

TernGrad – Convergence

Mathematically guarantee the convergence of *TernGrad*

L. Bottou 1998

Assumption 1. *$C(\boldsymbol{w})$ has a single minimum $\boldsymbol{w}^*$ and gradient $-\nabla_{\boldsymbol{w}} C(\boldsymbol{w})$ always points to $\boldsymbol{w}^*$, i.e.,*

$$\forall \epsilon > 0, \inf_{||\boldsymbol{w}-\boldsymbol{w}^*||^2>\epsilon} (\boldsymbol{w}-\boldsymbol{w}^*)^T \nabla_{\boldsymbol{w}} C(\boldsymbol{w}) > 0. \quad (8)$$

Convexity is a subset of Assumption 1, and we can easily find non-convex functions satisfying it.

Assumption 2. *Learning rate γ_t is positive and constrained as $\sum_{t=0}^{+\infty} \gamma_t^2 < +\infty$ and $\sum_{t=0}^{+\infty} \gamma_t = +\infty$, which ensures γ_t decreases neither very fast nor very slow respectively.*

We assume the gradient is bounded as

Assumption 3 (Gradient Bound)**.** *The gradient $\boldsymbol{g}$ is bounded as $\mathbf{E}\{max(abs(\boldsymbol{g})) \cdot ||\boldsymbol{g}||_1\} \leq A + B||\boldsymbol{w}-\boldsymbol{w}^*||^2$, where $A, B > 0$ and $||\cdot||_1$ is ℓ_1 norm.*

Theorem 1. *When online learning systems update as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma_t (s_t \cdot sign(\boldsymbol{g}_t) \circ \boldsymbol{b}_t)$ using stochastic ternary gradients, they converge* ***almost surely*** *toward minimum $\boldsymbol{w}^*$, i.e., $P(\lim_{t\to+\infty} \boldsymbol{w}_t = \boldsymbol{w}^*) = 1$.*

TernGrad – Gradients Histograms

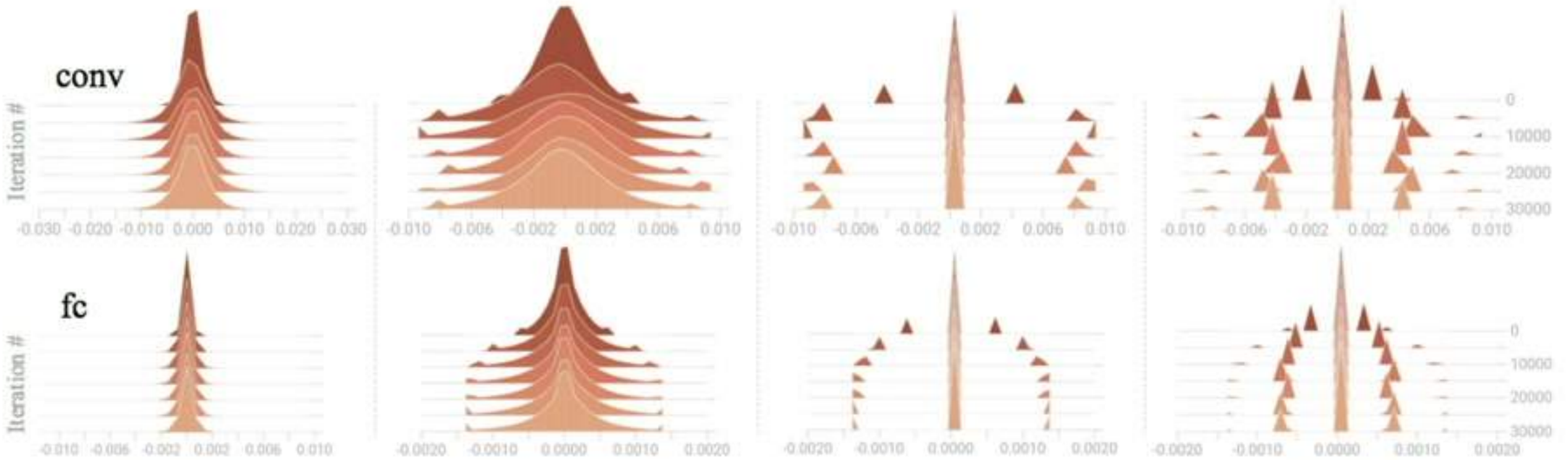
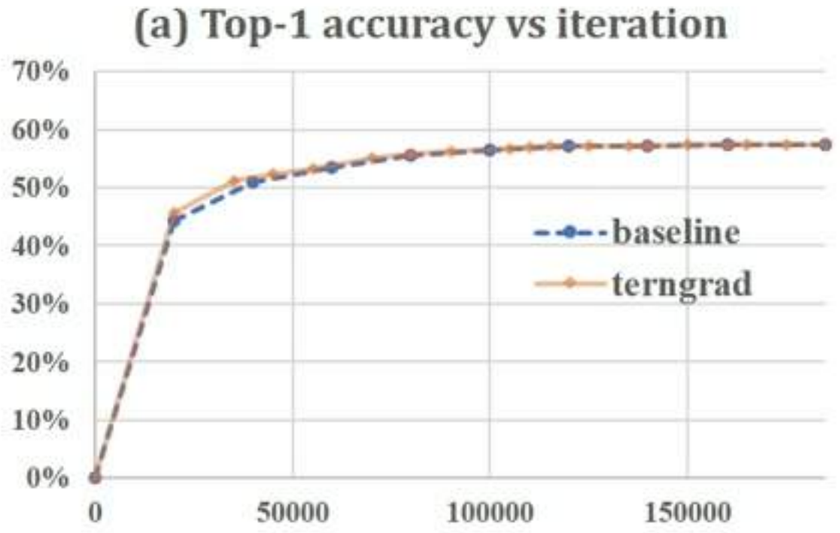


Figure 2: Histograms of (a) original floating gradients, (b) clipped gradients, (c) ternary gradients and (d) final averaged gradients. Visualization by TensorBoard. The DNN is *AlexNet* distributed on two workers, and vertical axis is the training iteration. Top row visualizes the third convolutional layer and bottom one visualizes the first fully-connected layer.

TernGrad – AlexNet



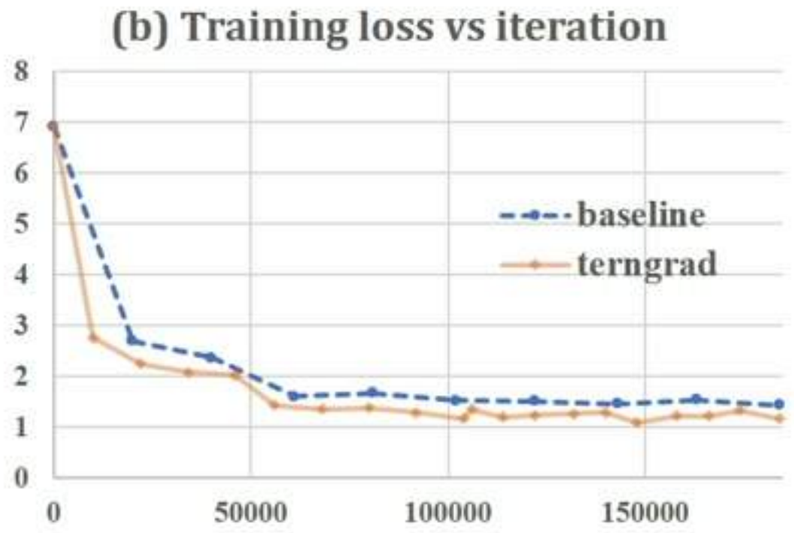


Table 2: Accuracy comparison for *AlexNet*.

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.

TernGrad – GoogLeNet

Table 3: Accuracy comparison for *GoogLeNet*.

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%

TernGrad – Speedup

A performance model to evaluate the speed distributed training.

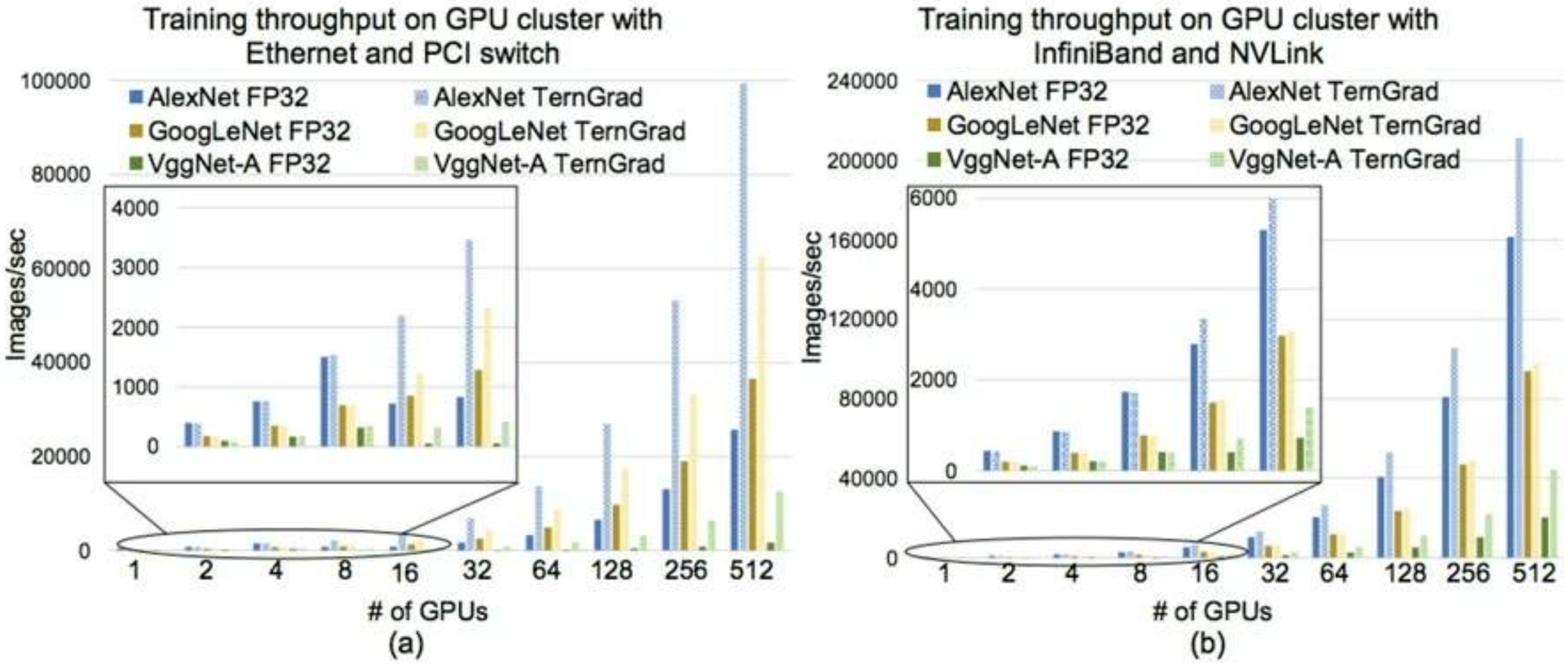


Figure 5: Training throughput on two different GPUs clusters: (a) 128-node GPU cluster with 1Gbps Ethernet, each node has 4 NVIDIA GTX 1080 GPUs and one PCI switch; (b) 128-node GPU cluster with 100 Gbps InfiniBand network connections, each node has 4 NVIDIA Tesla P100 GPUs connected via NVLink. Mini-batch size per GPU of *AlexNet*, *GoogLeNet* and *VggNet-A* is 128, 64 and 32, respectively

This is not the end …

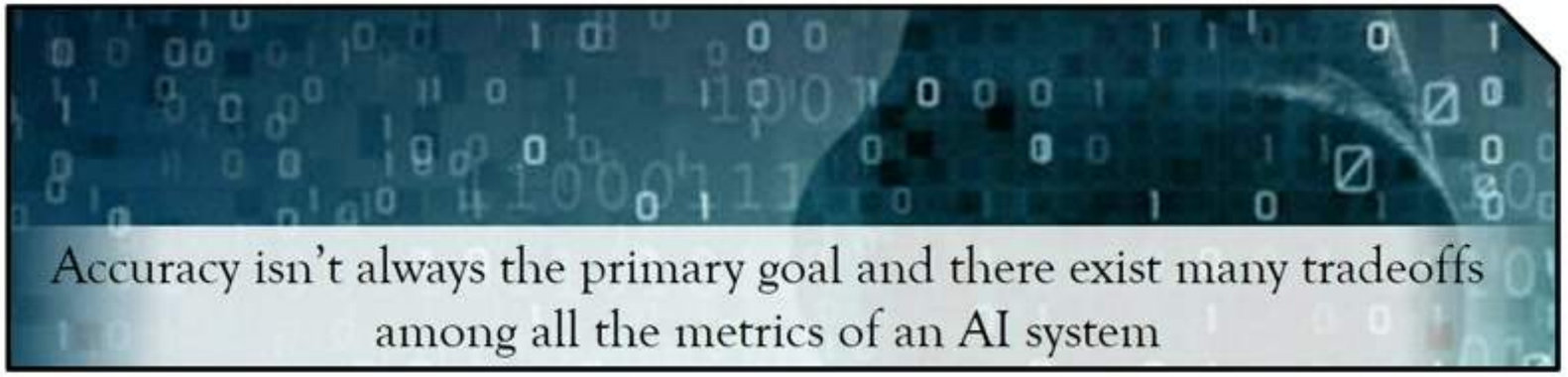
- Our 1-level quantization method (ASP-DAC'17 and DAC'16) is included in the latest PDK of IBM TrueNorth Chip.
- Our structural pruning technique (NIPS'16)
  - is supported by the library of Intel Nervana Neural Network Processors.
  - is adopted by Intel's newest NLP accelerator (ICLR'18).
  - is adopted by SF-Technology, achieving 2X performance improvement in their datacenter.
- Our TenGrad technique (NIPS'17) is supported by Facebook Caffe2 and HP parameter server product for distributed learning.

Our Perspective

1

2

3



NSF IUCRC ASIC Center

What is ASIC?

The **A**lternative **S**ustainable and **I**ntelligent **C**omputing (**ASIC**) Center is a multi-site, multidisciplinary consortium that explores research frontiers in emerging computing platforms for cognitive applications

Industry partners:

Coherus, IBM, NVIDIA, SRC, intel, Kneron, forté media, THALES, LG, Midea, Technergetics, LEONARDO, CTI, CDI Corporation, arm, ITRI, DEEPHI, hp, SF TECHNOLOGY, QUALCOMM, TERADEET, TEXAS INSTRUMENTS, EMERSON, SAAB, NOVUMIND, OAK RIDGE, Google

Members include influential faculty across the three research sites:


Yiran Chen
Center/Site Director


Hai "Helen" Li
Center/Site Co-Director


Benjamin Lee
Center/Site Co-Director


Robert Calderbank


Krish Chakrabarty


Xin Li


Miroslav Pajic

Duke UNIVERSITY


Qinru Qiu
Site Director


Pramod Vashney
Site Co-Director


Yanzhi Wang
Site Co-Director


Jian Tang


Amit Sanyal


Senem Velipasalar

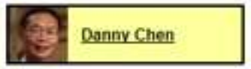

Bei Yu

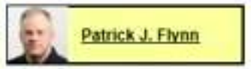
Syracuse University


Yiyu Shi
Site Director

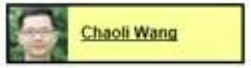

Sharon Hu
Site Co-Director


Michael Niemier
Site Co-Director


Danny Chen


Patrick J. Flynn


Walter Scheirer


Chaoli Wang

UNIVERSITY OF NOTRE DAME

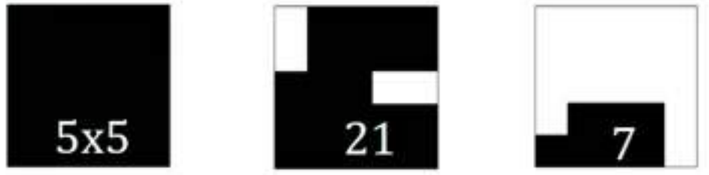
Experiments – Learning smaller filter shapes

Table 2: Results after learning filter shapes in *LeNet*

LeNet #	Error	Filter size §	Channel #	FLOP	Speedup
1 (*baseline*)	0.9%	25—500	1—20	100%—100%	1.00×—1.00×
4	0.8%	21—41	1—2	8.4%—8.2%	2.33×—6.93×
5	1.0%	7—14	1—1	1.4%—2.8%	5.19×—10.82×

§ The sizes of filters after removing zero shape fibers, in the order of *conv1*—*conv2*

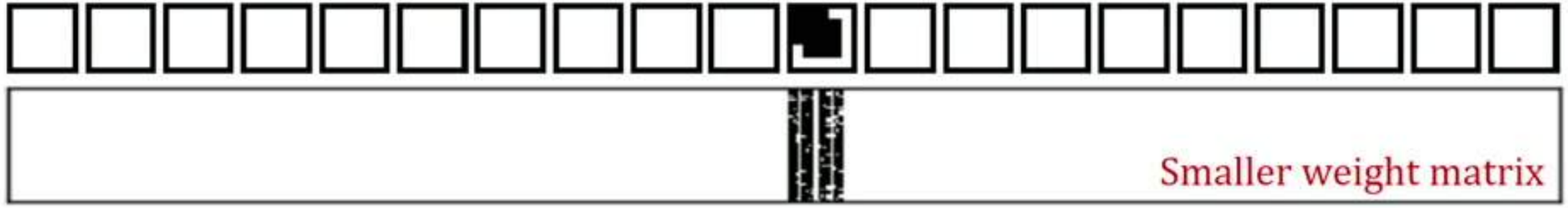
Learned shapes of conv1 filters:



LeNet 1 *LeNet 4* *LeNet 5*

Learned shape of conv2 filters @ *LeNet 5*

3D 20x5x5 filters is regularized to 2D filters!



SSL can efficiently learn DNNs with smaller filters without accuracy loss