Compressing Neural Networks Pruning and Quantization

May 15, 2023

Sungjoo Yoo

Computing Memory Architecture Lab.

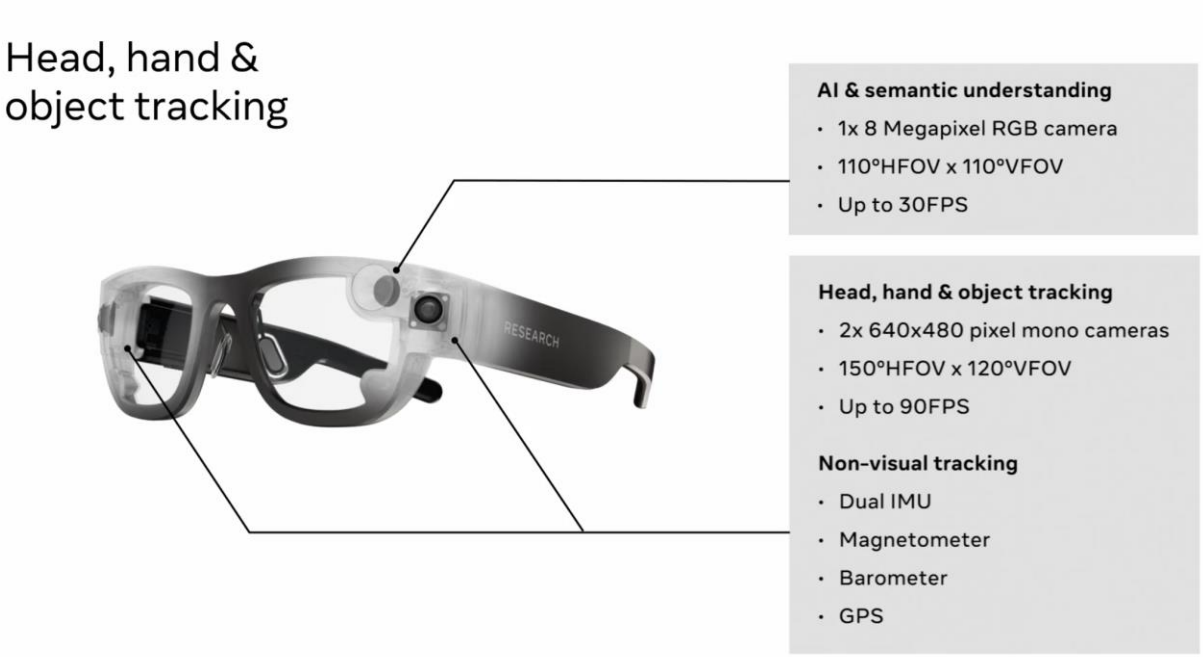
CSE, SNU

http://cmalab.snu.ac.kr

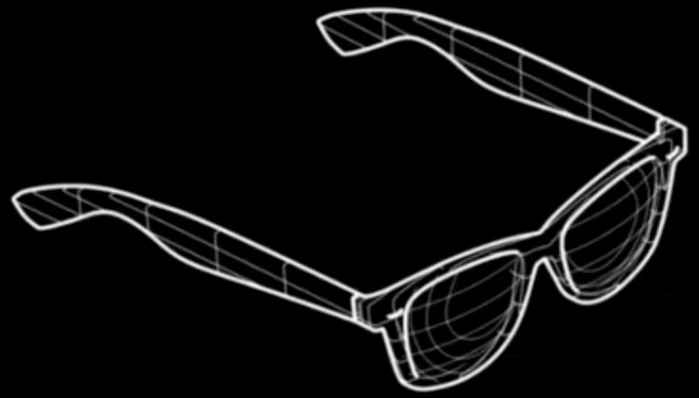
[Nreal][Facebook]

Improving Energy Efficiency is Key to Future AI Devices, e.g., AR Glasses

- Dr. Vikas Chandra, On-Device AI, Facebook Reality Lab.
- “In order to realize mobile AR devices, 100X energy efficiency is needed compared with the current ASIC NPU based solutions”



Mar 22–26
www.tinyML.org

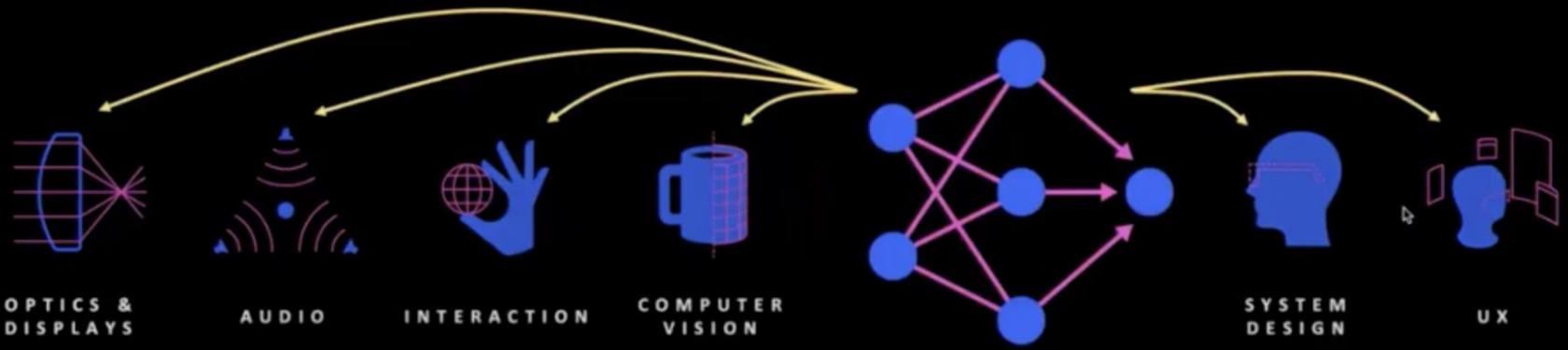


SOCIALLY ACCEPTABLE

LIGHTWEIGHT

ALL-DAY BATTERY LIFE

THERMALLY CONSTRAINED



Thank You to our 26 sponsors:

Challenges for On-device AI in AR

From the Applications …

- Support for heterogeneous applications
- No user perceivable latency
- Support for processing high resolution data

From the System …

- 10× – 100× better energy efficiency
- Limited compute and storage resources
- Limited and shared bandwidth to system memory

Weekly Lecture / Lab Schedule

- W1 (March 6) Class introduction / (March 8) Orientation & team formation
- W2 13 Verilog 1 Combinational circuits / 15 Verilog 1 (tool installation, adder & multiplier combinational logic)
- W3 20 Verilog 2 Sequential circuits / 22 Verilog 2 (memory i/o, FSM sequential logic)
- W4 27 AI application introduction 1, Amaranth introduction 1 / 29 Amaranth (tool installation, MAC, adder tree)
- W5 4/3 AI application introduction 2, Amaranth introduction 2 (memory i/o, FSM sequential logic) / 5 Amaranth (PE)
- W6 10 AI application introduction 3, Neural network accelerator 1 / 12 Amaranth (PE controller)
- W7 17 Neural network accelerator 2 / 19 Q&A
- W8 24 **Mid-term exam** / 26 Amaranth (outer product accelerator)
- W9 5/1 Reading data from memory 1 (VA2PA, interconnect) / 3 PyTorch (simple CNN on MNIST)
- W10 8 Reading data from memory 2 (DRAM main memory) / 10 Quantization aware training (QAT)
- W11 15 Compressing networks (pruning, low precision) / 17 Convolution lowering & tiling
- W12 22 Zero-skipping & low-precision hardware accelerator / 24 PyTorch – Amaranth communication
- W13 29 Invited talk (1h online Google Edge TPU, 1h offline Furiosa Date to be determined soon) / 31 Zero skipping
- W14 6/5 **Final exam** / 7 Project Q&A
- W15 12 (online) Claim & Project Q&A / 14 (online) Project Q&A, submission

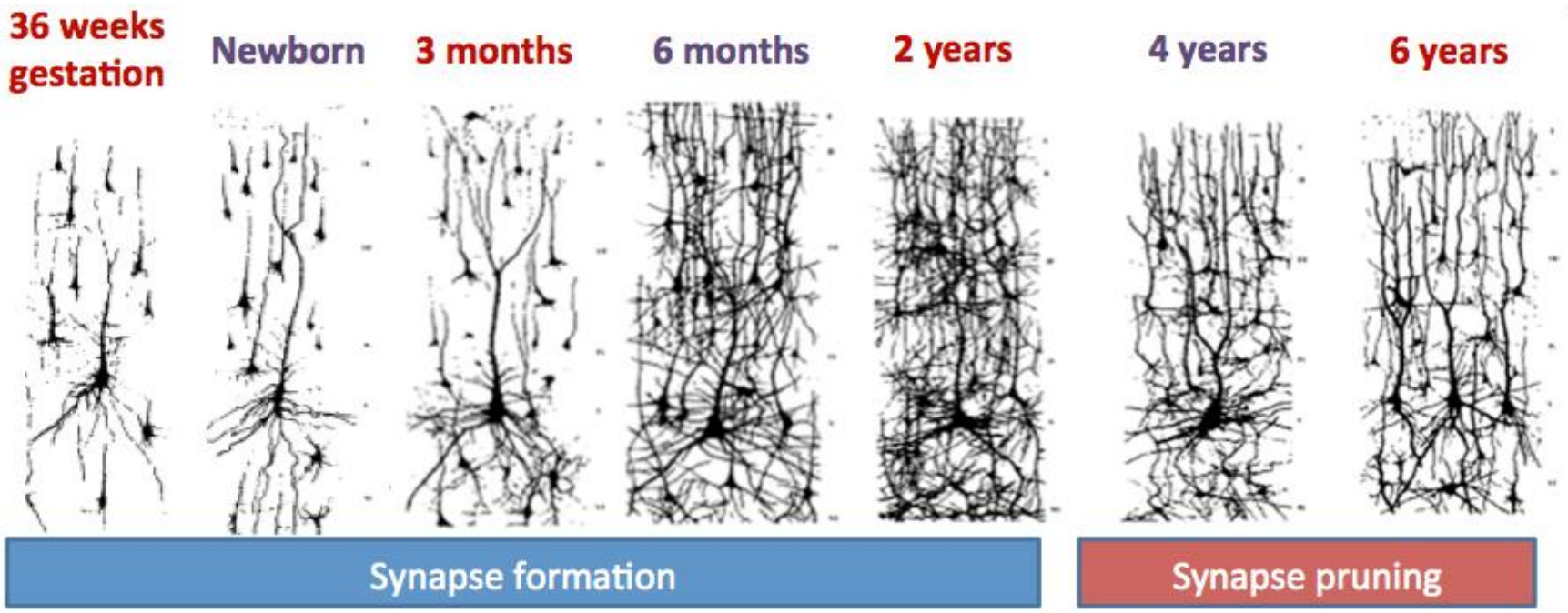
Pruning: Agenda

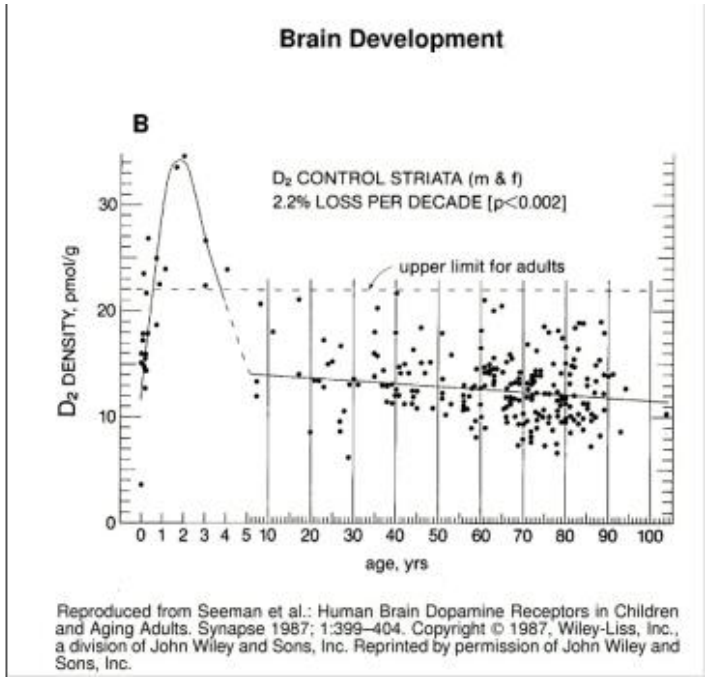
- Introduction
- Magnitude-based pruning
- Structured pruning
  - Groupwise brain damage
  - NVIDIA’s 2:4 rule pruning
- Lottery ticket pruning

[https://universe-review.ca/R10-16-ANS12.htm]

Neuron Pruning is Natural in Biological System

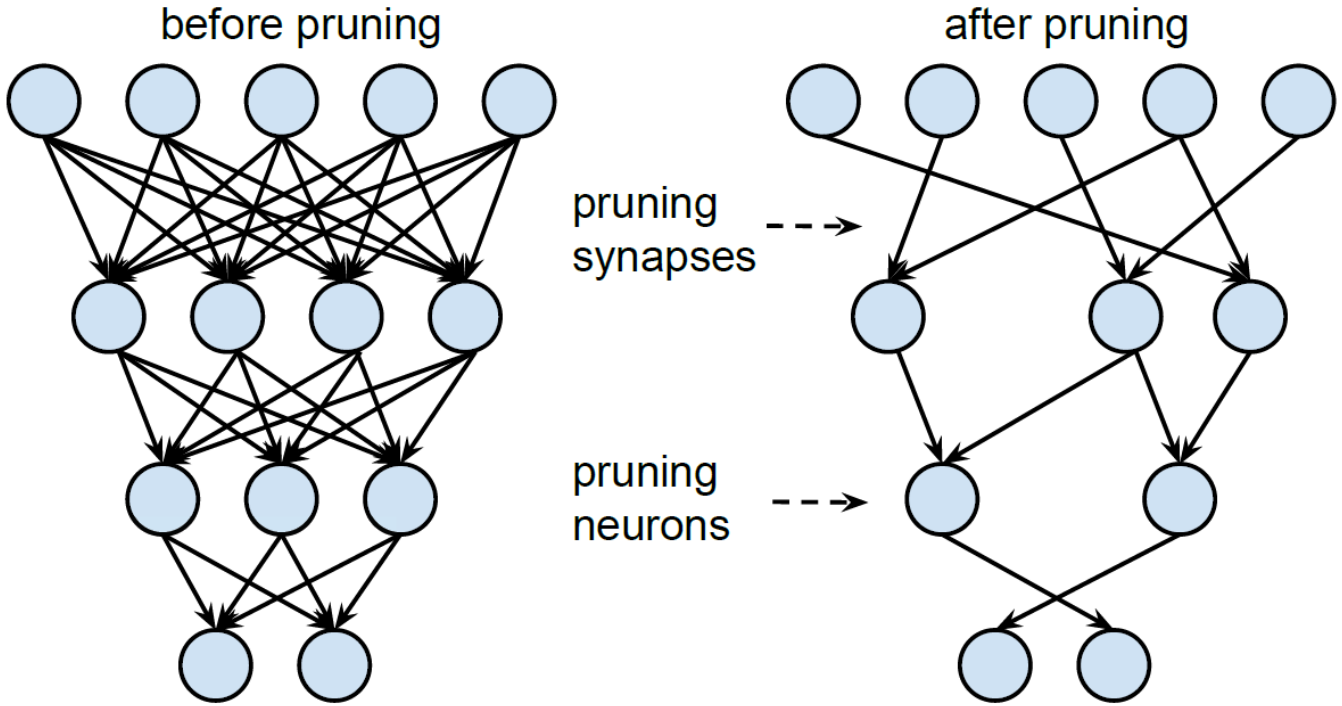
- # synapses increases before 2 years old and, then decreases due to pruning, possibly to reduce resource (e.g., energy) usage

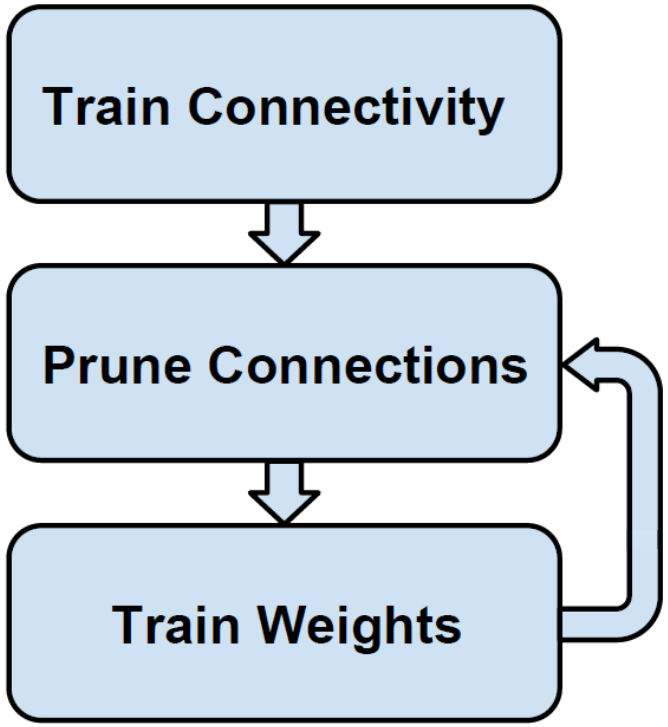




[Han, 2015]

Magnitude-based Pruning: NIPS 2015





Goal and Benefits

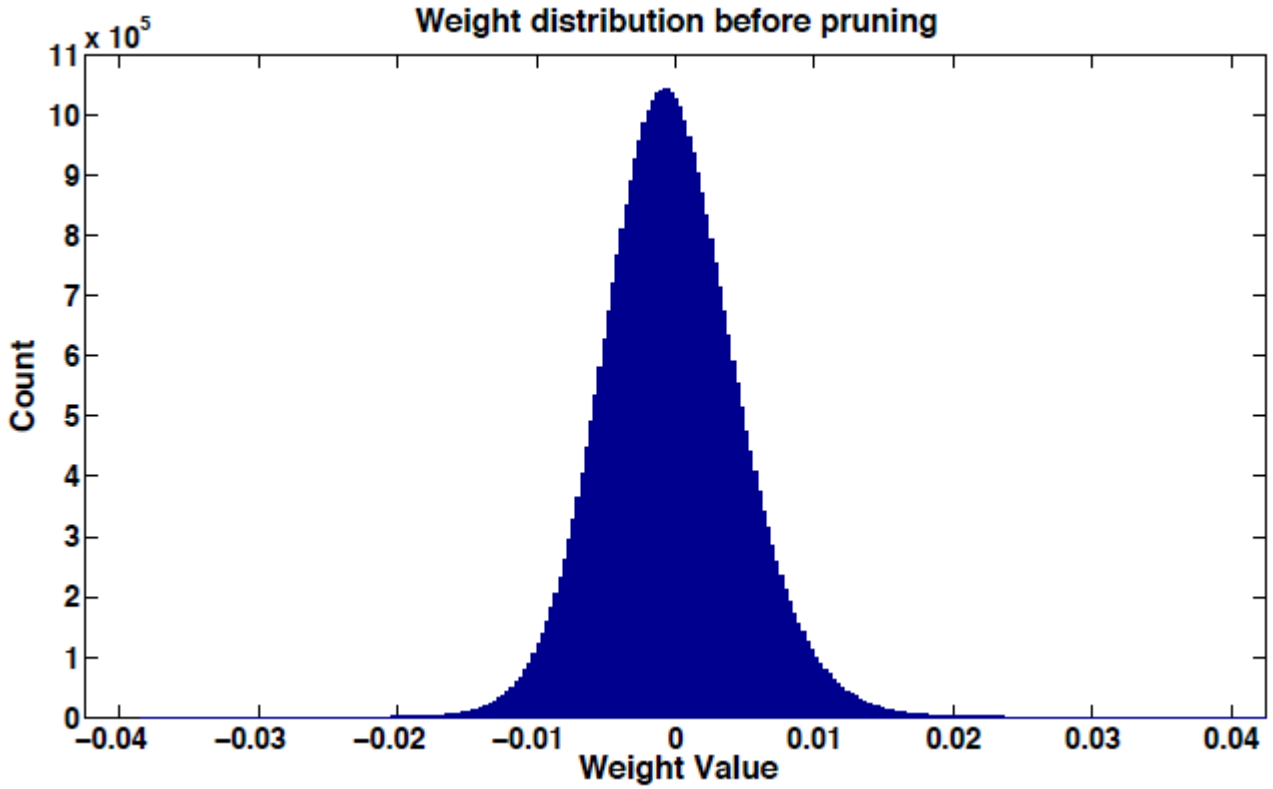
- “Our goal is to reduce the storage and energy required to run inference on such large networks so they can be deployed on mobile devices.”

- Benefits of small models for mobile devices
- Faster download
- Lower energy consumption in computation and data transfer inside of computing system (e.g., silicon chip)

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**

Illustration of Pruning

- Weight distribution before pruning



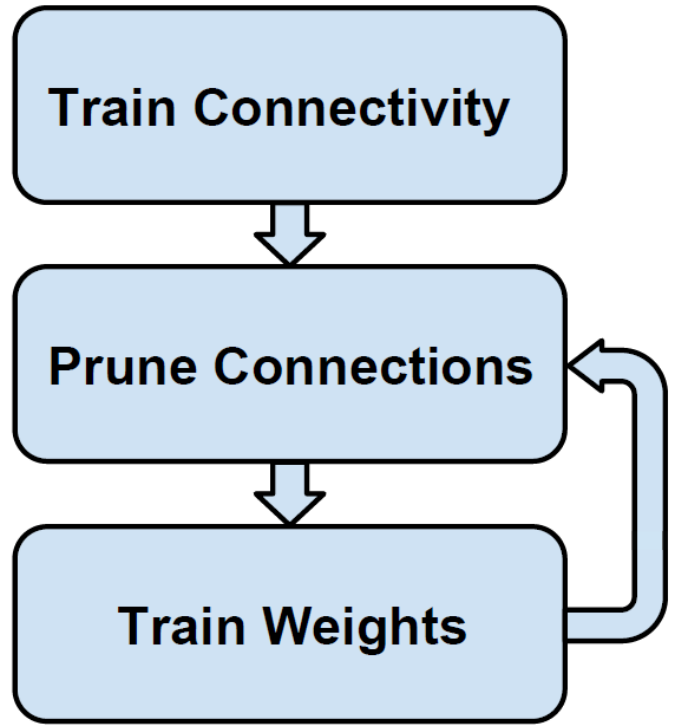
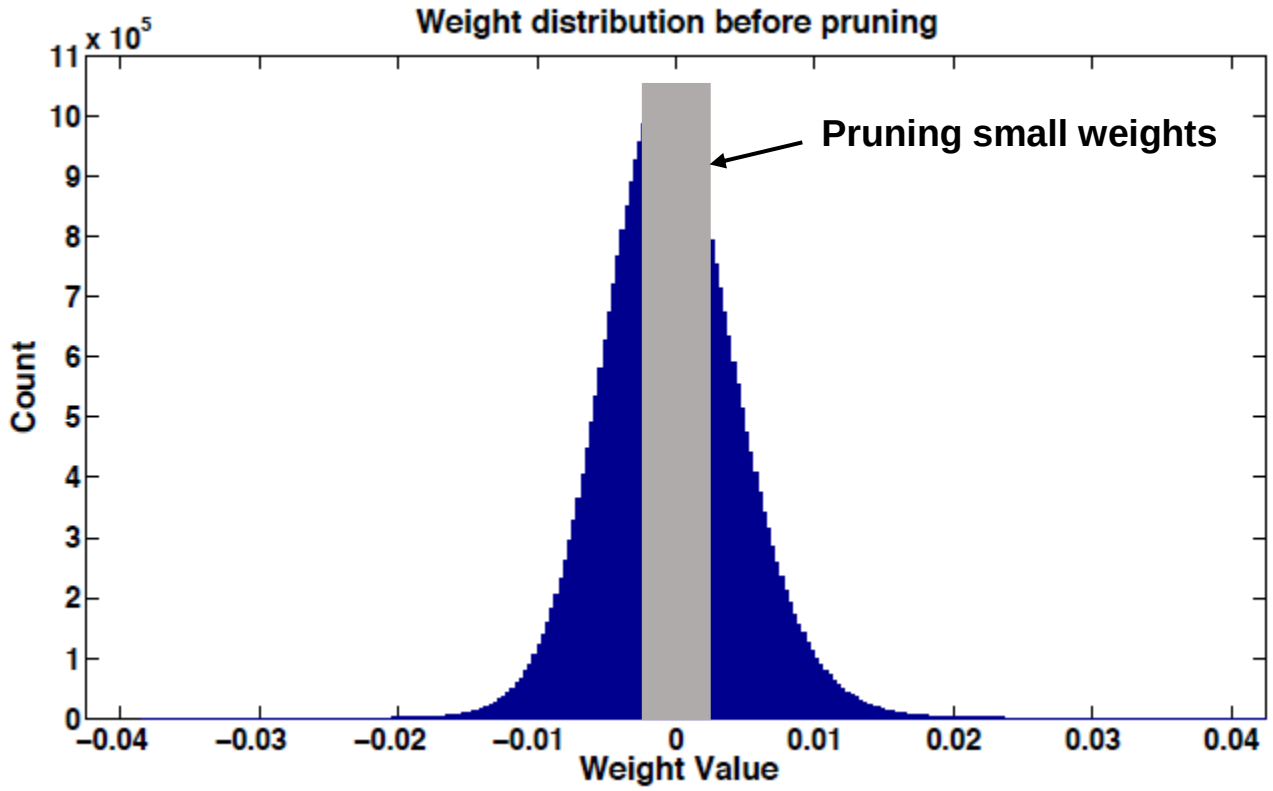


Illustration of Pruning

- Small weights are pruned, i.e., set to zero



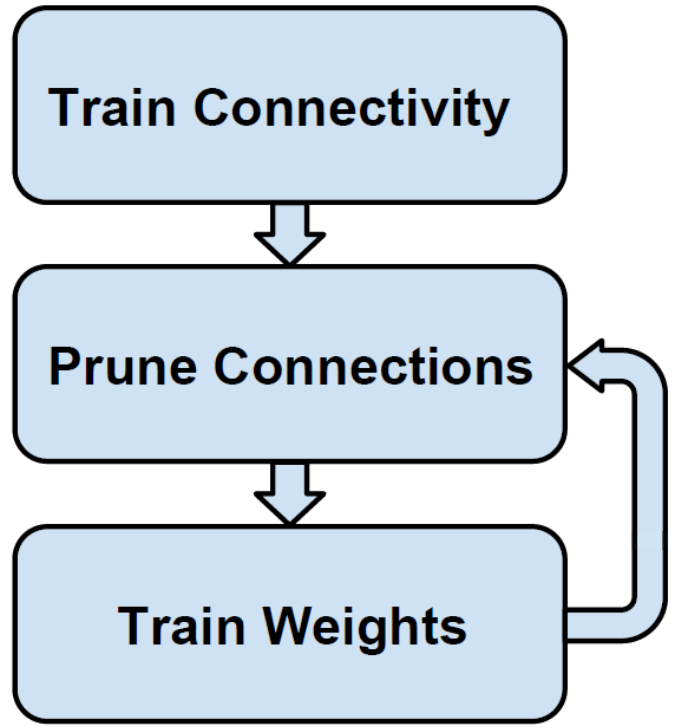
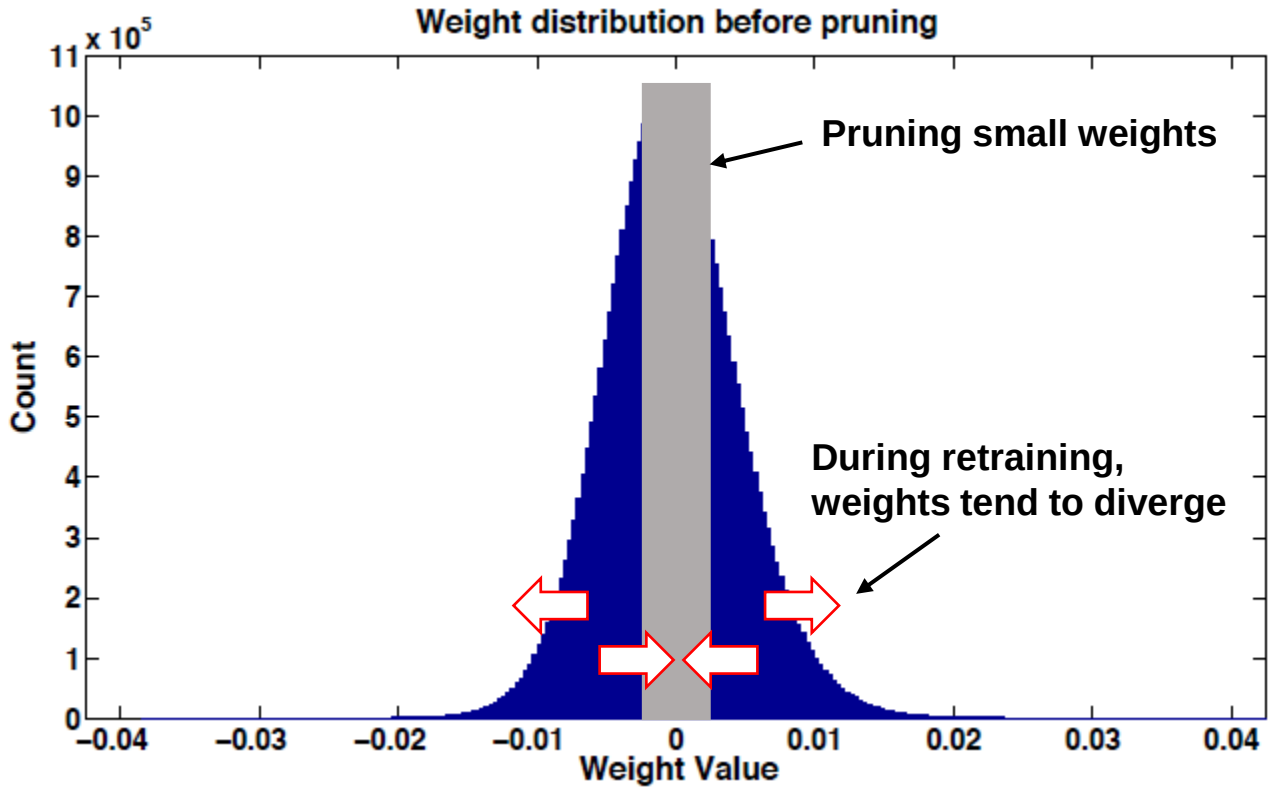


Illustration of Pruning

- During training, weights are re-distributed



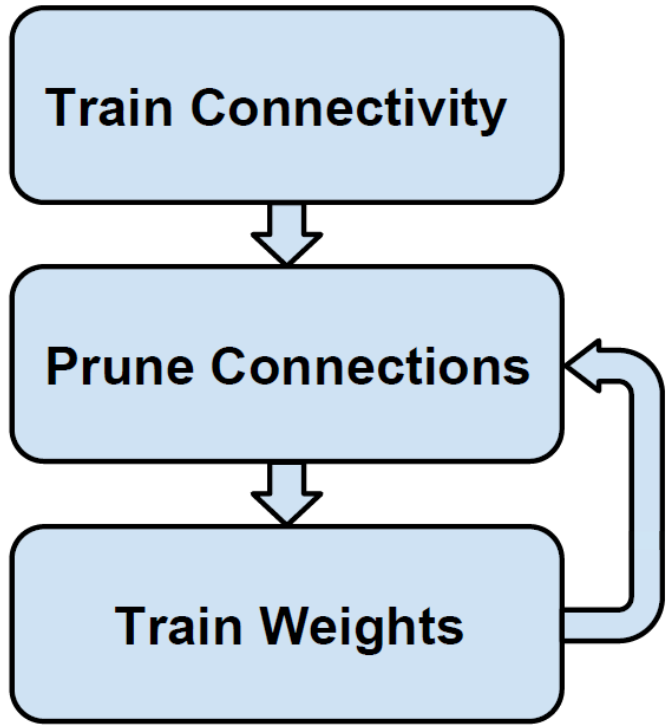
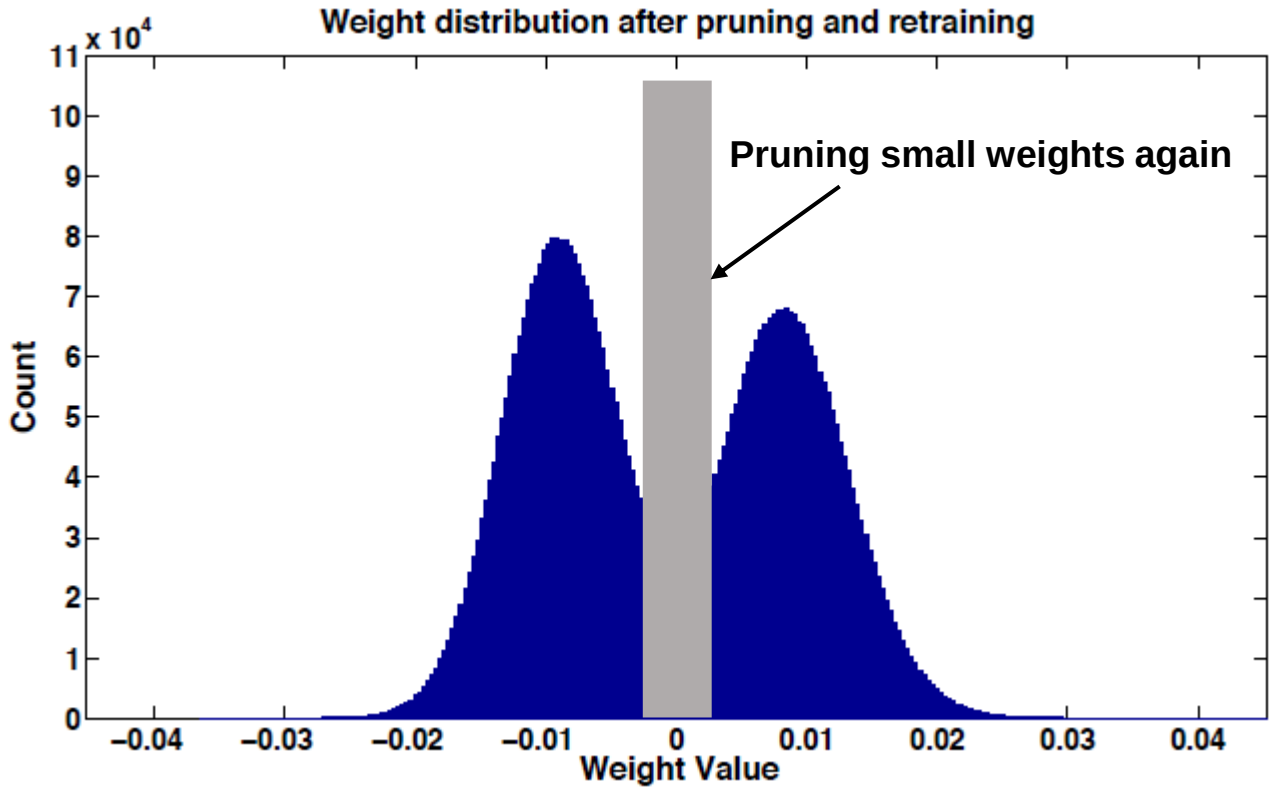
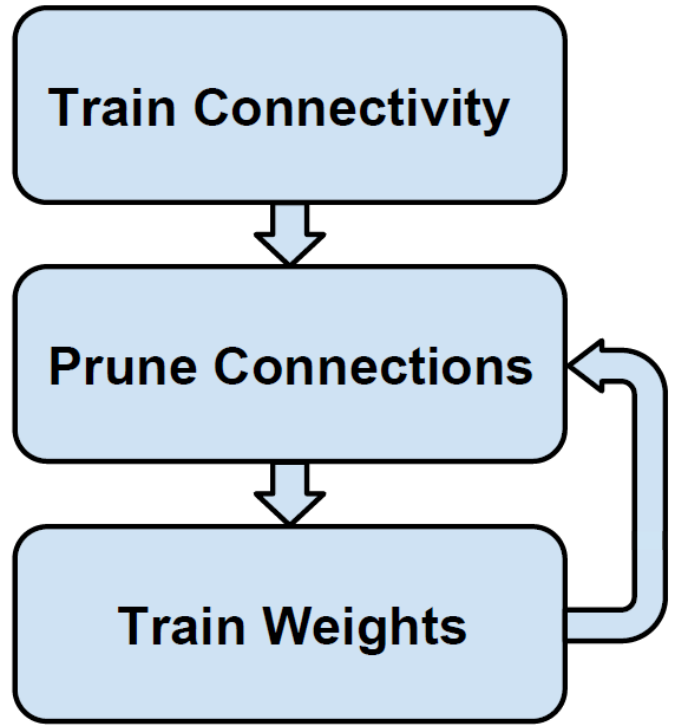


Illustration of Pruning

- After a step of training, new small weights are obtained

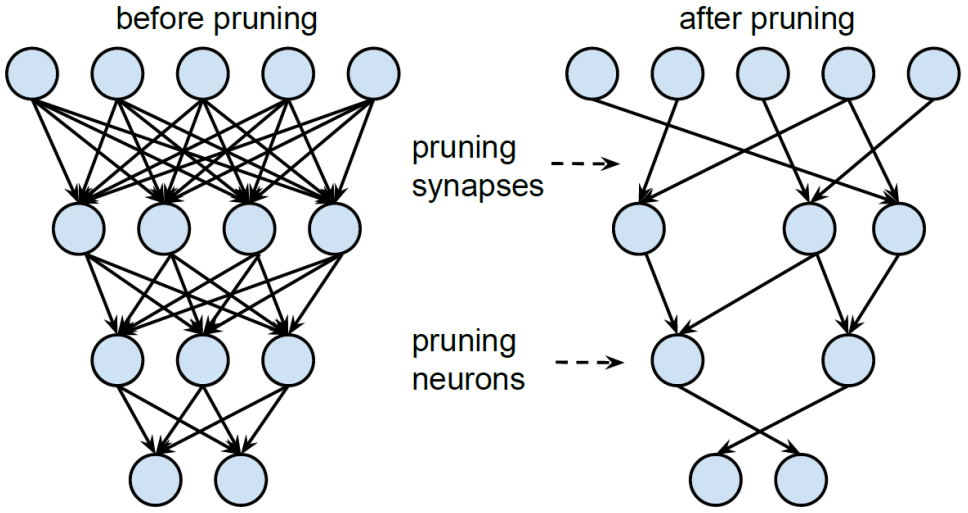


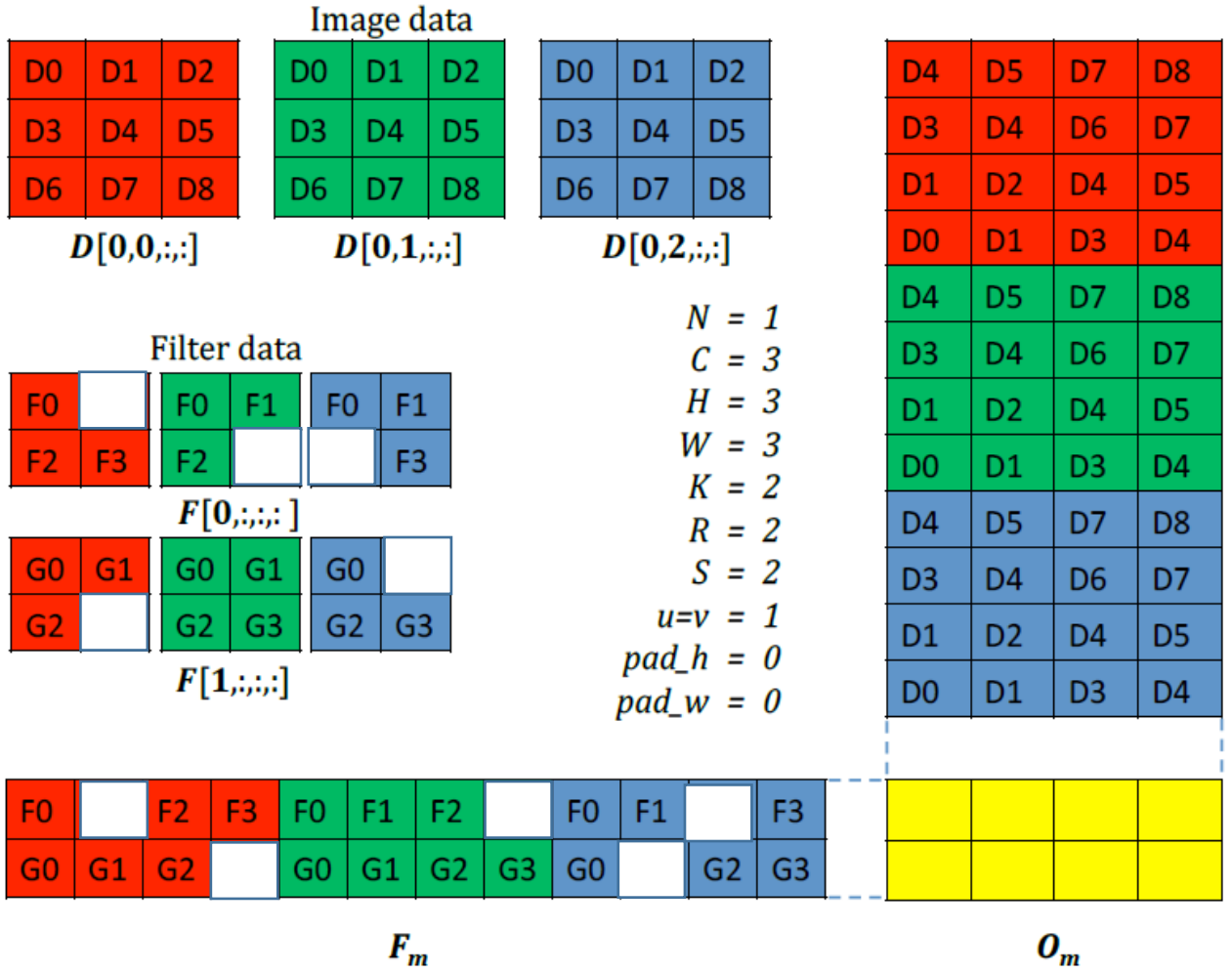
10X smaller scale



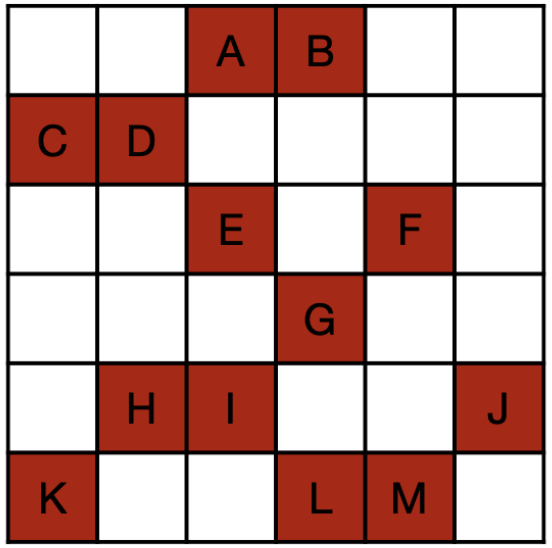
[Han, 2015][Chetlur, 2014]

How to Efficiently Store Sparse Weight Matrix? Compressed Sparse Row (CSR) or Column (CSC)





CSR Format for Sparse Matrices



0	2	4	6	7	10	13

Row Pointer

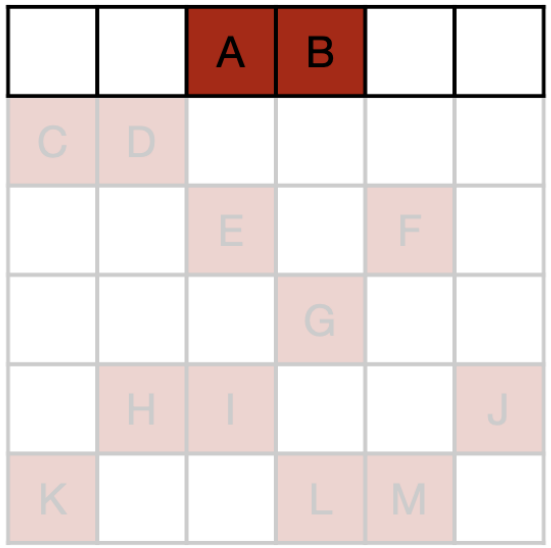
2	3	0	1	2	4	3	...

Column Indices

A	B	C	D	E	F	G	...

Values

CSR Format for Sparse Matrices



`A[0].nonzeros = [2, 3]`

Because there is no element before the current row.

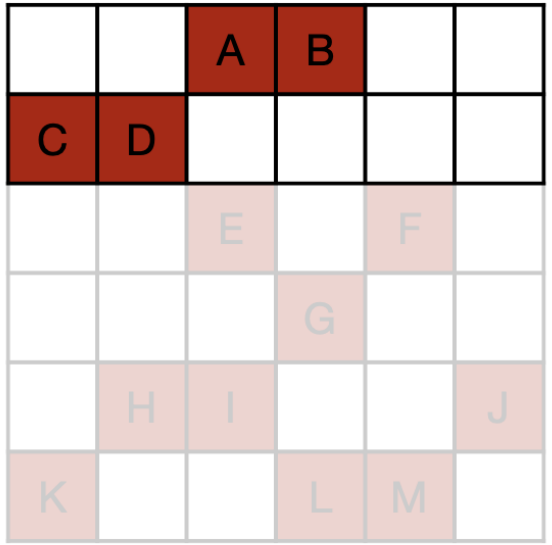
Row Pointer

Column Indices

Values

Nonzero values for row 0

CSR Format for Sparse Matrices



`A[1].nonzeros = [0, 1]`

Because there are two elements before the current row.

0	2	4	6	7	10	13

Row Pointer

2	3	0	1	2	4	3	…

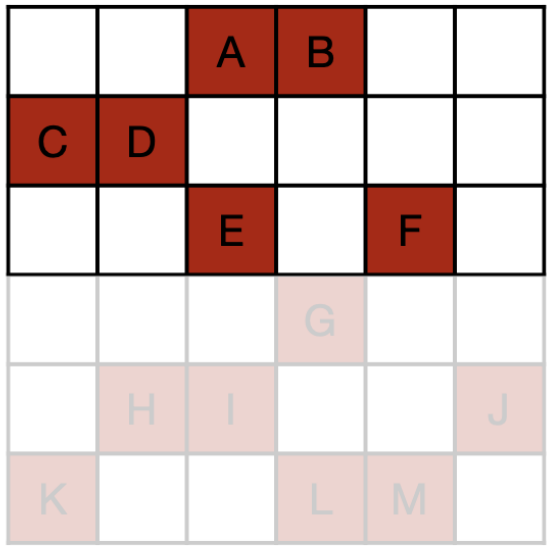
Column Indices

A	B	C	D	E	F	G	…

Values

Nonzero values for row 1

CSR Format for Sparse Matrices



Because there are four elements before the current row.

0	2	4	6	7	10	13

Row Pointer

2	3	0	1	2	4	3	...

Column Indices

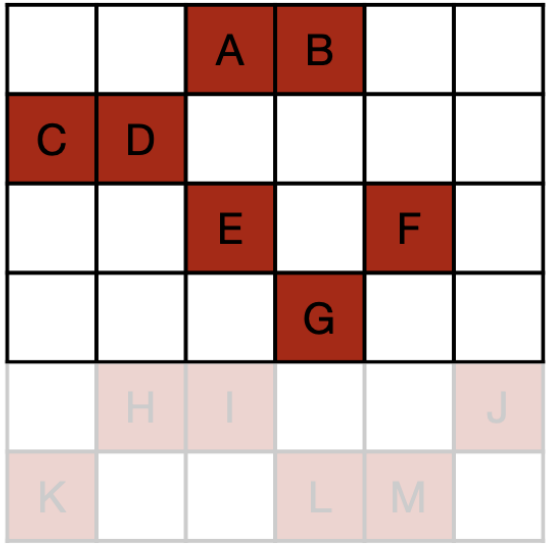
`A[2].nonzeros = [2, 4]`

A	B	C	D	E	F	G	...

Values

Nonzero values for row 2

CSR Format for Sparse Matrices



Because there are six elements before the current row.

0	2	4	6	7	10	13

Row Pointer

2	3	0	1	2	4	3	...

Column Indices

A[3].nonzeros = [3]

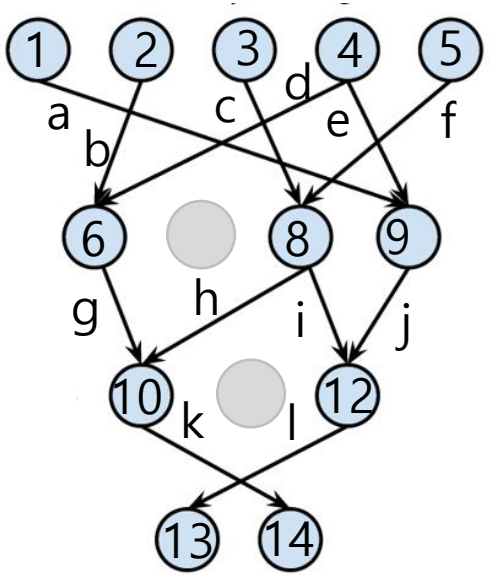
A	B	C	D	E	F	G	...

Values

Nonzero values for row 3

Compressed Sparse Row or Column (CSR, CSC) Format to Locate Non-Zero Weights

- “We store the sparse structure that results from pruning using compressed sparse row (CSR) or compressed sparse column (CSC) format, which requires 2a+n+1 numbers, where a is the number of non-zero elements and n is the number of rows or columns.”



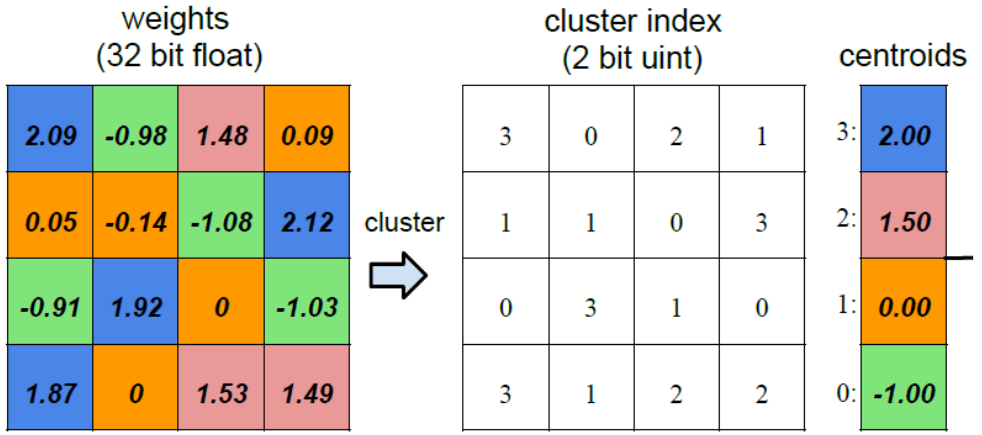
CSC-like example

Non-zero weight indices	4 5 11 13 16 19 / 1 7 9 12 / 2 5
Weight values	a b c d e f g h i j k l

Weight Clustering to Reduce # Bits to Represent Each Weight

- “We limit the number of effective weights we need to store by having multiple connections share the same weight, and then fine-tune those shared weights.”

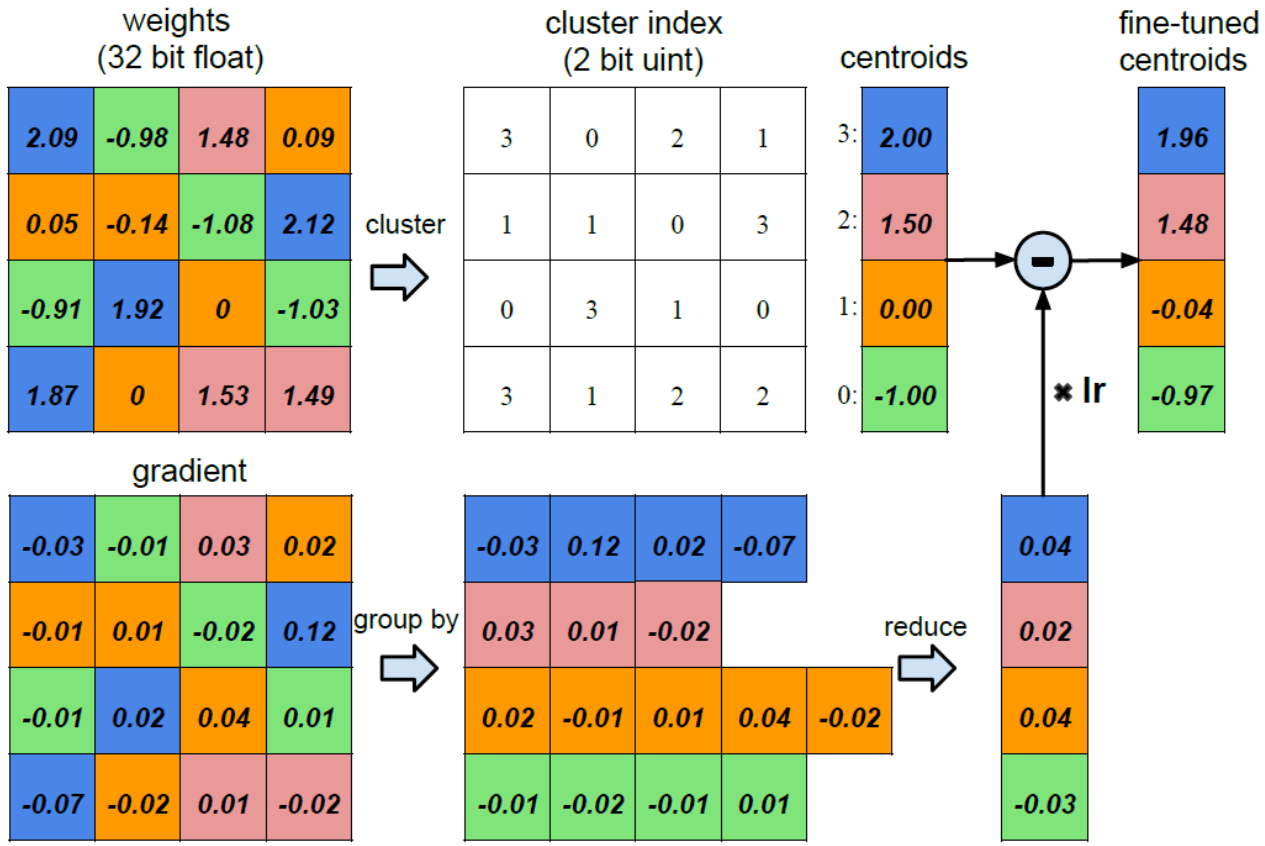
“The weights are quantized to 4 bins (denoted with 4 colors), all the weights in the same bin share the same value, thus for each weight, we then need to store only a small index into a table of shared weights.



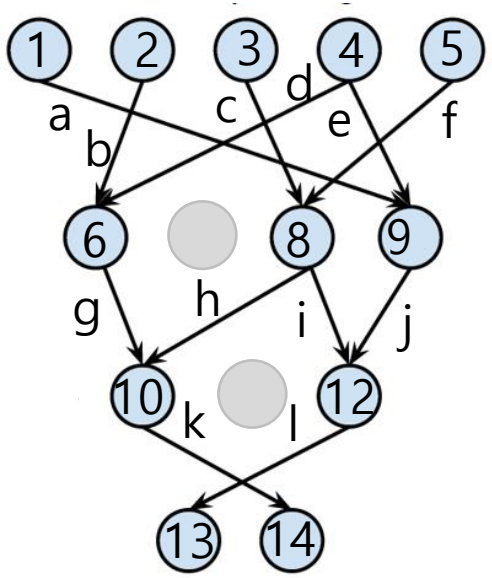
Weight Clustering to Reduce # Bits to Represent Each Weight

- "We limit the number of effective weights we need to store by having multiple connections share the same weight, and then fine-tune those shared weights."

"The weights are quantized to 4 bins (denoted with 4 colors), all the weights in the same bin share the same value, thus for each weight, we then need to store only a small index into a table of shared weights. During update, all the gradients are grouped by the color and summed together, multiplied by the learning rate and subtracted from the shared centroids from last iteration. For pruned AlexNet, we are able to quantize to 8-bits (256 shared weights) for each CONV layers, and 5-bits (32 shared weights) for each FC layer without any loss of accuracy."

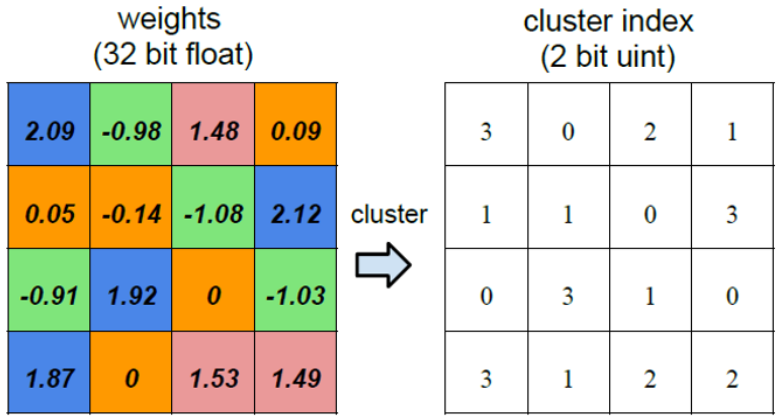


Replacing Weight Values with Bin Indices

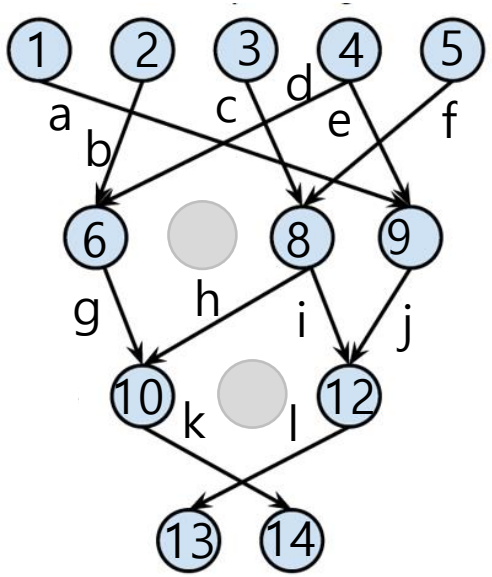


CSC-like example

Non-zero weight indices	4 5 11 13 16 19 / 1 7 9 12 / 2 5	
	4 1 6 2 3 3 / 1 6 2 3 / 2 3	4 bits
Weight values	a b c d e f g h i j k l	
Weight value bin indices	3 0 2 1 1 0 3 0 3 1 0 3	8 bits (CONV) 5 bits (FC)

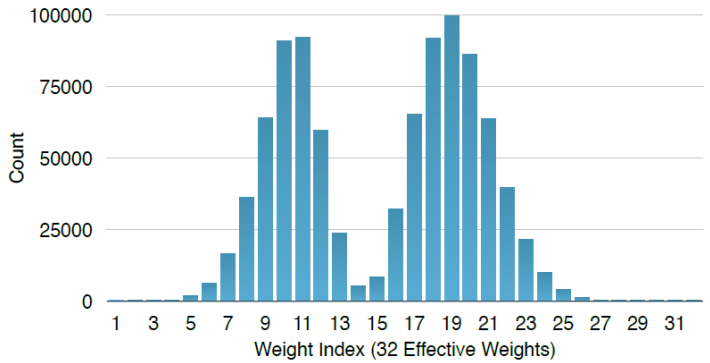


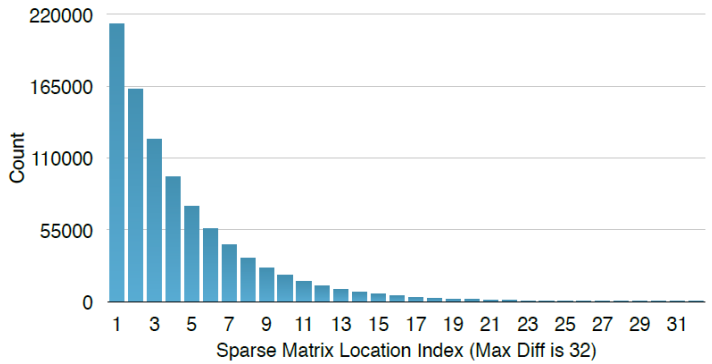
Huffman Coding of Bin Indices and Non-Zero Weight Indices



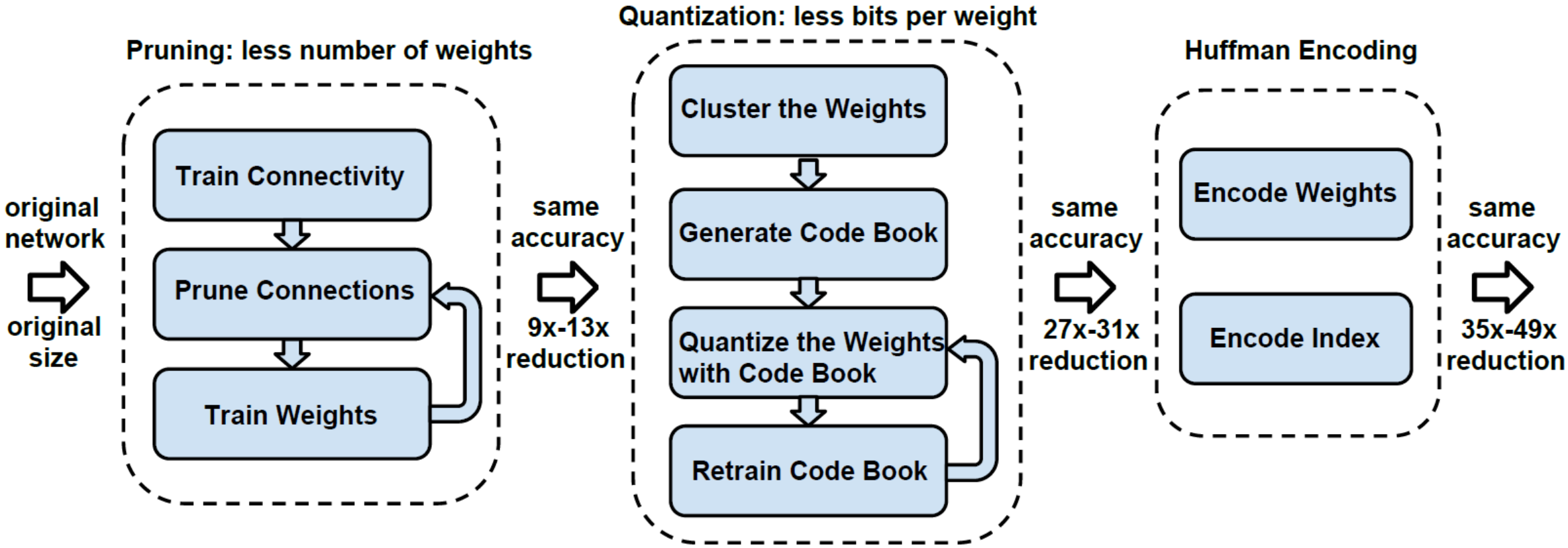
CSC-like example

Non-zero weight indices	4 5 11 13 16 19 / 1 7 9 12 / 2 5	
	4 1 6 2 3 3 / 1 6 2 3 / 2 3	4 bits
Weight values	a b c d e f g h i j k l	
Weight value bin indices	3 0 2 1 1 0 3 0 3 1 0 3	8 bits (CONV) 5 bits (FC)





Pruning, Quantization and Compression: ICLR 2016



Accuracy, Compression Ratio and # Bits (Levels)

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	**40**×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	**39**×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	**35**×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	**49**×

AlexNet

#CONV bits / #FC bits	Top-1 Error	Top-5 Error	Top-1 Error Increase	Top-5 Error Increase
32bits / 32bits	42.78%	19.73%	-	-
8 bits / 5 bits	42.78%	19.70%	0.00%	-0.03%
8 bits / 4 bits	42.79%	19.73%	0.01%	0.00%
4 bits / 2 bits	44.77%	22.33%	1.99%	2.60%

AlexNet

Non-zero weight indices: 4 5 11 13 16 19 / 1 7 9 12 / 2 5

4 1 6 2 3 3 / 1 6 2 3 / 2 3 — 4 bits

Weight values: a b c d e f g h i j k l

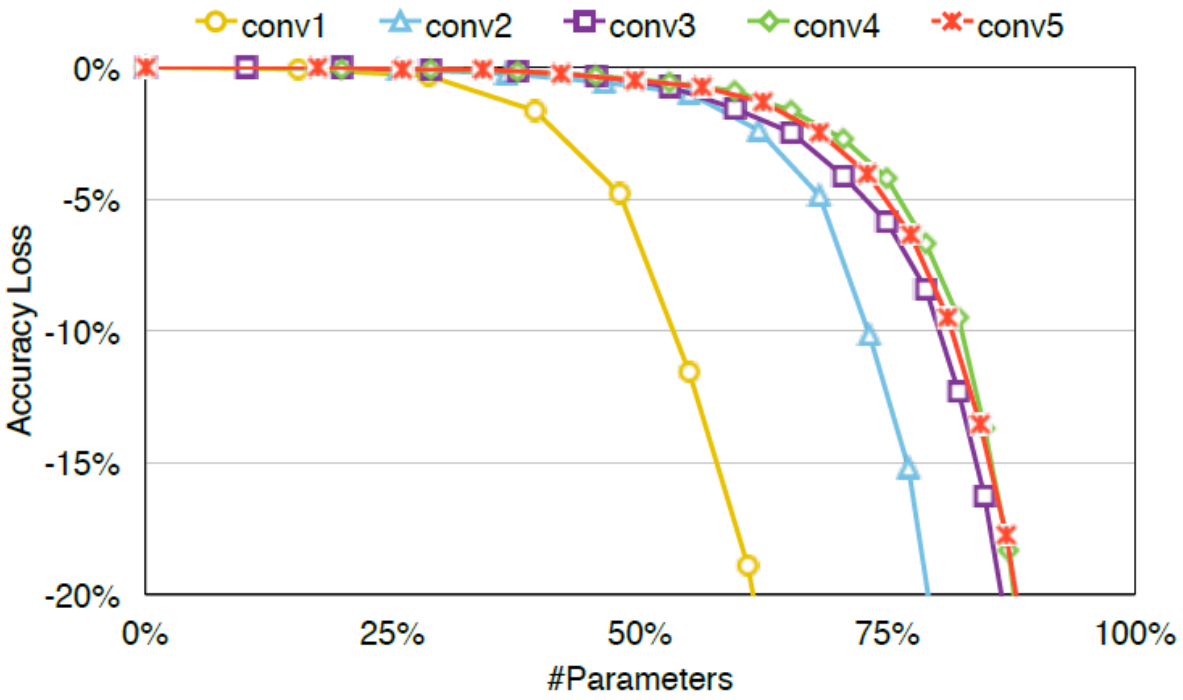
Weight value bin indices: 3 0 2 1 1 0 3 0 3 1 0 3 — 8 bits (CONV), 5 bits (FC)

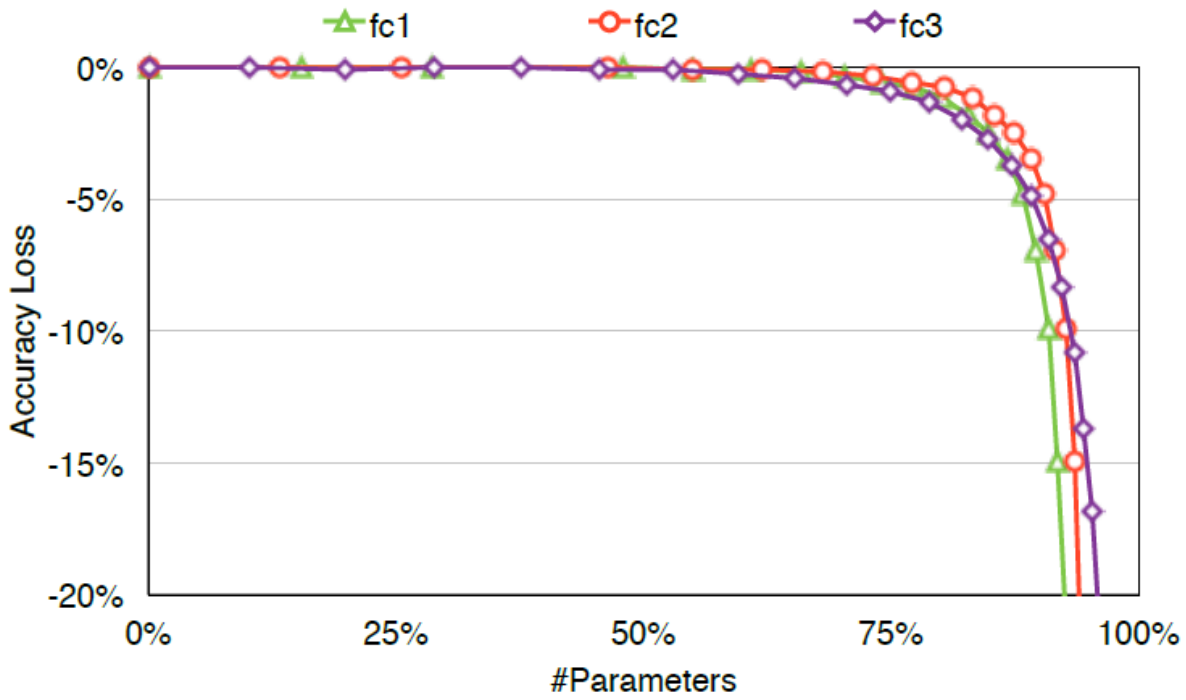
- 8b/5b case: 256 levels for CONV and 32 levels for FC layers
- Finally, weights can be represented by ~2.5 bits (CONV) and ~3.5 bits (FC)

Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
Total	61M	11%(9×)	5.4	4	4	3.2	3.7% (**27**×)	2.88% (**35**×)

Per-Layer Sensitivity

- CONV1 (near the input) is the most sensitive
- FC layers (near the output) is much less sensitive





After Deep Compression Paper …

They Started A Startup, DeePhi and Sold It to Xilinx

About DeePhi Tech

"DeePhi Tech"

DeePhi Tech focuses on deep learning, pushing the frontier of AI applications. Supported and invested by GSR Venture and Banyan Capital, we successfully developed a complete automation flow of co
acceleration which achieves joint optimizatio
hardware. A smaller, faster and more efficier
(DPU) is released to public.
Deephi has built deep collaboration with lead
security surveillance and cloud service. The
achieves an order of magnitude higher energ
recognition and speech detection. Deephi be
algorithm, software and hardware would be t
Innovation and high quality research is our d
between products and customers makes us

Xilinx Announces the Acquisition of DeePhi Tech

Deal to Accelerate Data Center and Intelligent Edge Applications

Jul 17, 2018

BEIJING and SAN JOSE, Calif., July 17, 2018 – Xilinx, Inc. (NASDAQ: XLNX) the leader in adaptive and intelliger
Technology Co., Ltd (DeePhi Tech), a Beijing-based privately held start-up with industry-leading capabilities i
system-level optimization for neural networks.

DeePhi Tech is at the leading position in multiple disciplines including neural network compression, neural network complier, processor design and FPGA programming. Relative works have been reported top conferences including NIPS, ICLR, FPGA and ISCA. The neural network processors have already been in actual deployment.

Team Introduction

Team members in DeePhi Tech include the one of the first tenures in Tsinghua University Yu Wang, Song Han from Stanford University and other experts previously worked in famous companies such as Baidu, Siemens and Nokia.

Song Han, Founder & Chief Scientist
Graduate from Tsinghua
Ph.D at Stanford
Lead deep learning model compression and acceleration research

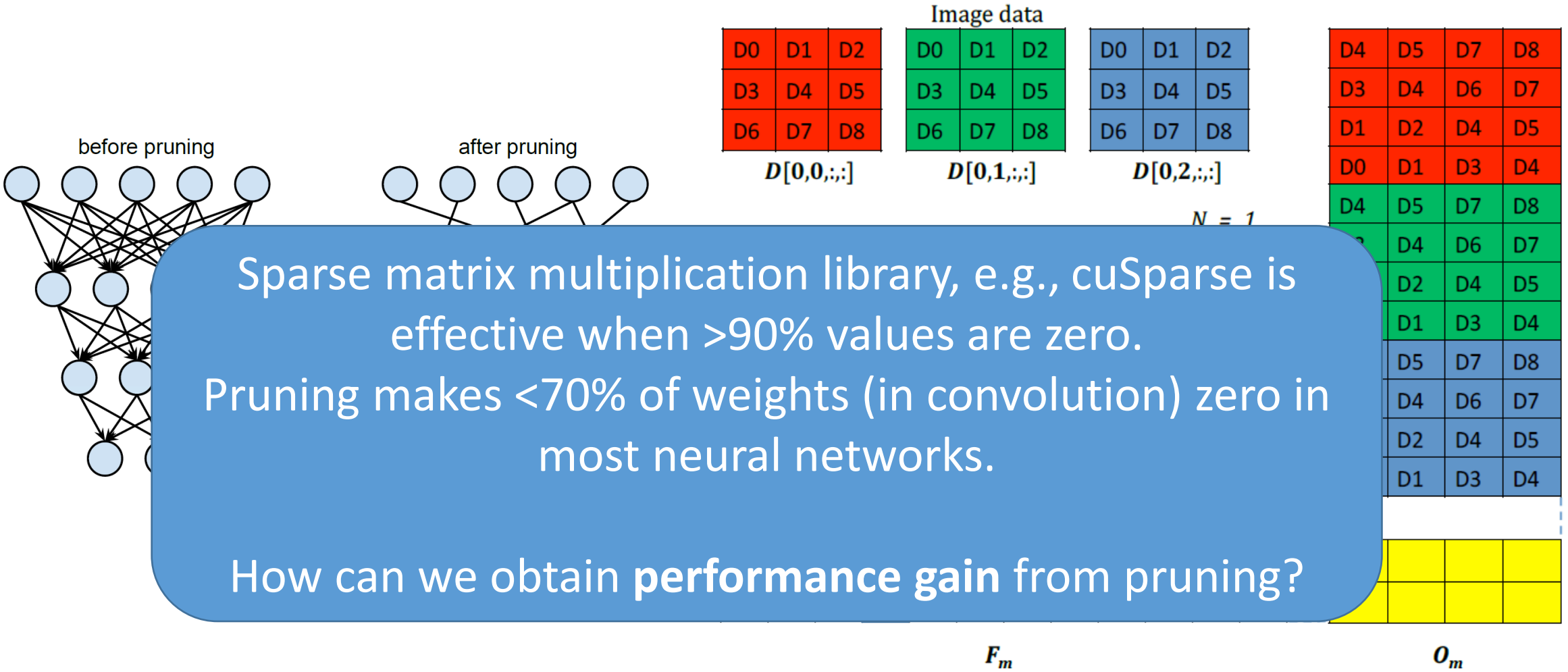
From idea to startup, it took 1 year.
The startup was founded in 2016 and sold to Xilinx in 2018.

Pruning: Agenda

- Introduction
- Magnitude-based pruning
- Structured pruning
  - Groupwise brain damage
  - NVIDIA’s 2:4 rule pruning
- Lottery ticket pruning

[Han, 2015][Chetlur, 2014]

Pruning [Han 2015] Hardly Reduces the Runtime of Convolution on GPU

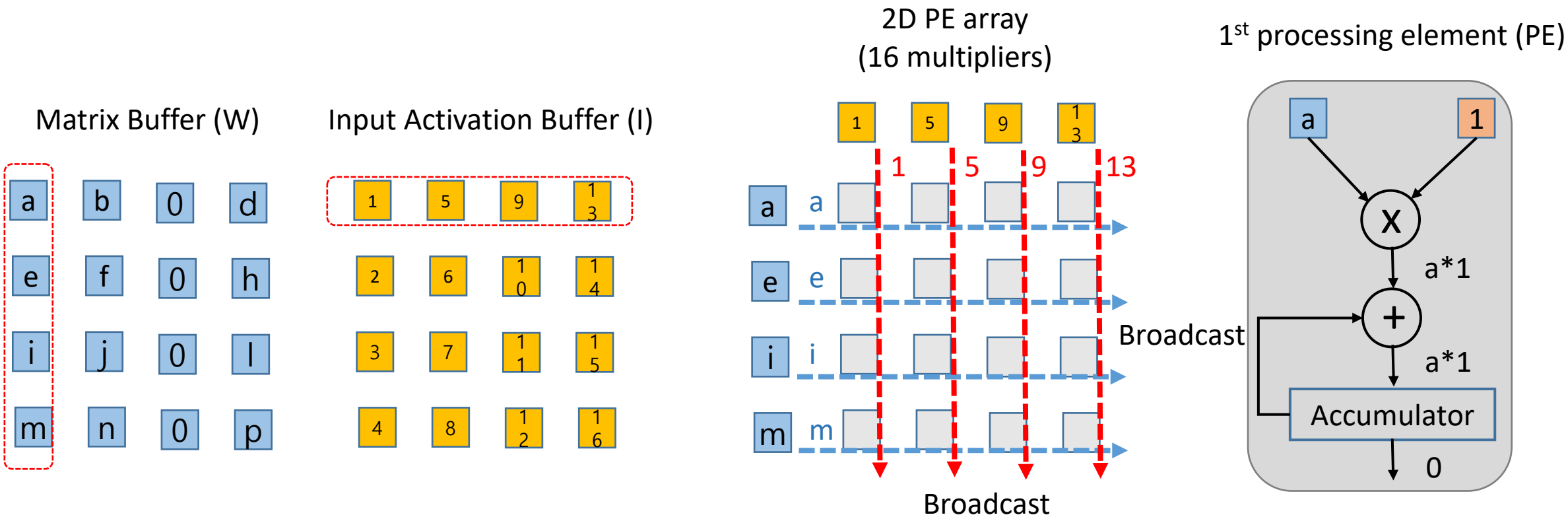


Sparse matrix multiplication library, e.g., cuSparse is effective when >90% values are zero.
Pruning makes <70% of weights (in convolution) zero in most neural networks.

How can we obtain **performance gain** from pruning?

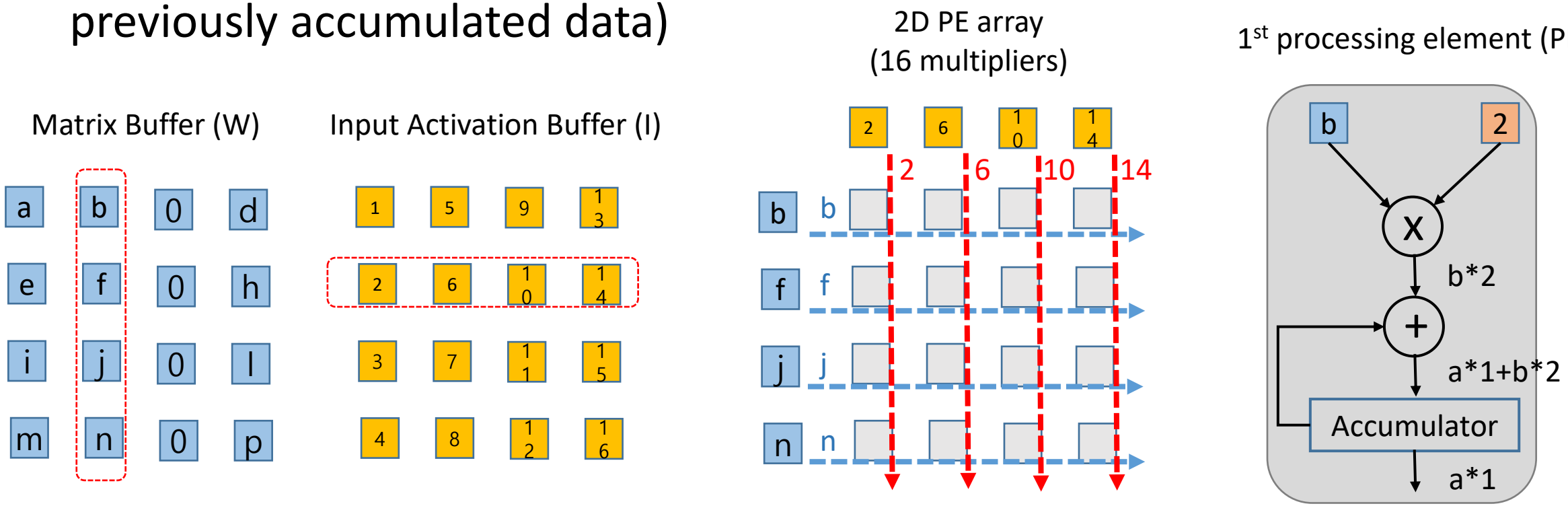
How to Exploit Zero-Skipping in Our Accelerator?

- Column level pruning in weight matrix will enable zero skipping
- Assume 3rd column in weight matrix is pruned



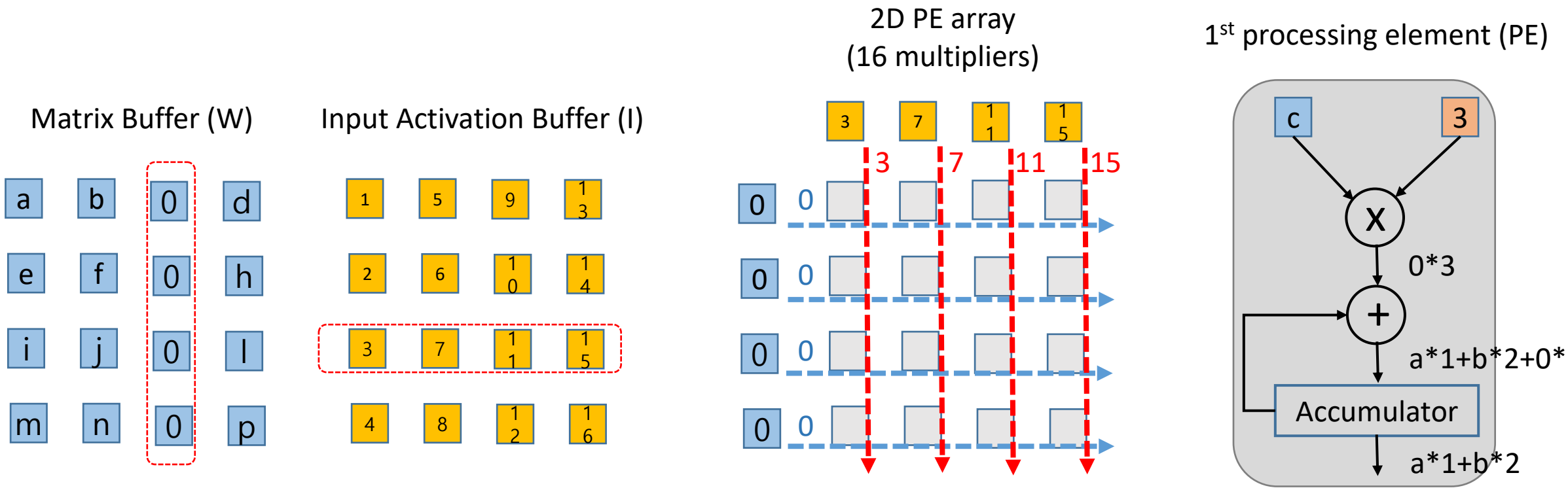
We Have Only to Do Computation with Non-Zero Columns

- 2nd clock cycle: Two new 4x1 and 1x4 vectors are read from the matrices to the line buffers of 2D PE array and each PE performs a multiplication and accumulation (of new multiplication result + previously accumulated data)



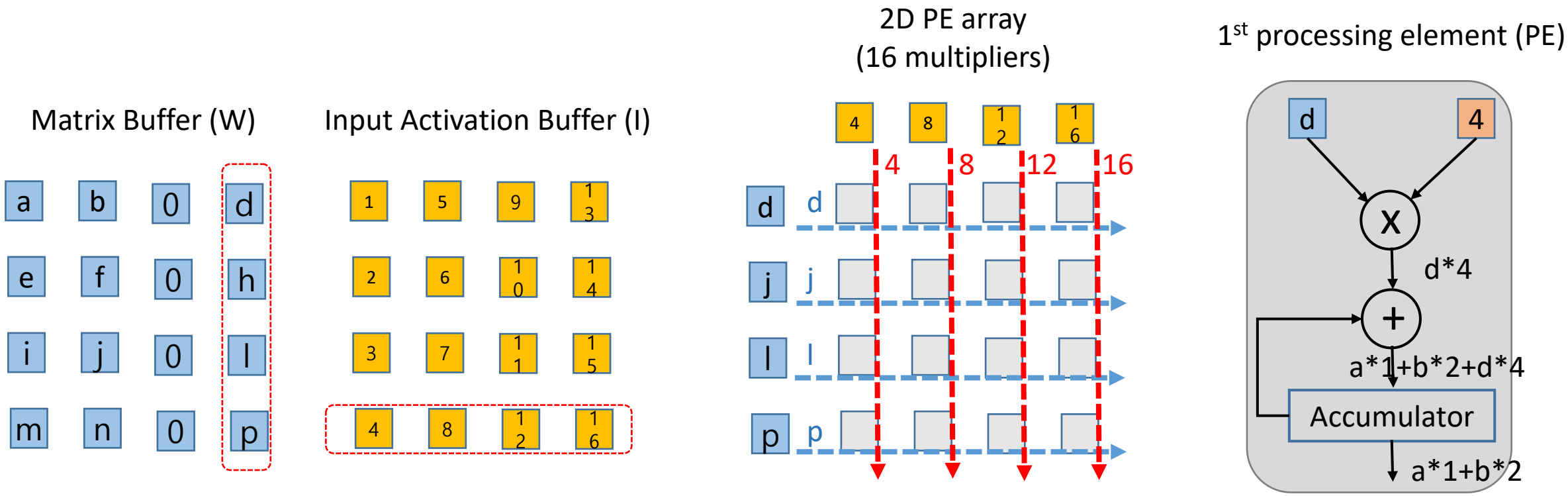
In 3rd Cycle, We Do Not Spend Cycle for Zero Column Input

- 3rd clock cycle: We skip the read operations for 3rd column and 3rd row
- As in the case of Sparse Tensor Core, we need metadata to represent zero columns



Instead, We Read 4th Column and Do Computation thereby Saving One Cycle

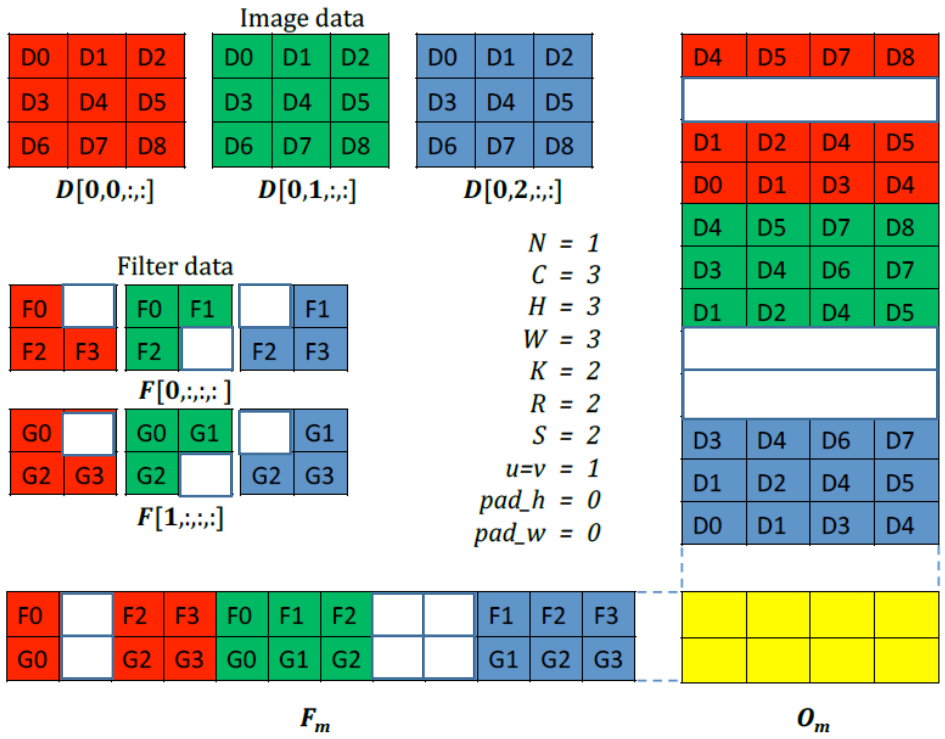
- **3rd clock cycle**: The 4th column and rows are read from the matrices
- Zero skipping enables a reduction in total cycle from 4 to 3 cycles



[Lebedev, 2016]

How to Realize Column-wise Pruning?

- Similar to NVIDIA's pruning
- Step 1: Train the network
- Step 2: Prune columns
  - Prune columns whose sum of absolute weights < threshold
- Step 3: Fine-tune the pruned network
- Repeat Steps 2 and 3 as far as the quality of network output is maintained



[Lebedev, 2016]

AlexNet Experiments

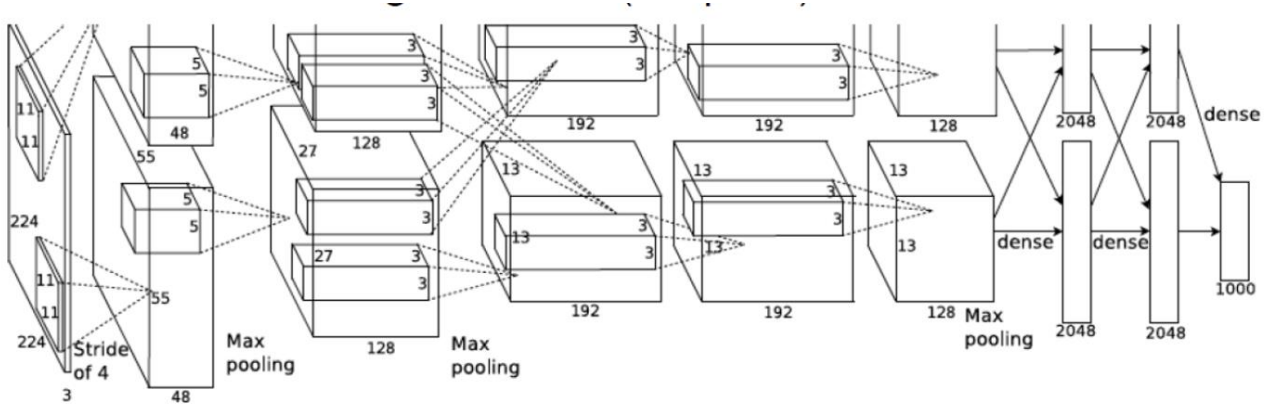
method	density	speed-up	accuracy drop
Accelerating the second convolutional layer of AlexNet			
Denton et al. [15]: Tensor decomposition + Fine-tuning		2.7x	$\sim 1\%$
Lebedev et al. [29]: CP-decomposition + Fine-tuning		4.5x	$\sim 1\%$
Jaderberg et al. [21]: Tensor decomposition + Fine-tuning		6.6x	$\sim 1\%$
Training with fixed sparsity patterns	0.12	8.33	0.82%
Training with fixed sparsity patterns	0.2	5x	0.16%
Group-wise sparsification + Fine-tuning	0.1	10x	1.13%
Group-wise sparsification + Fine-tuning	0.2	5x	0.43%
Group-wise sparsification + Fine-tuning	0.3	3.33x	0.11%
Group-wise sparsification + Fine-tuning	0.4	2.5x	-0.09%
Gradual group-wise sparsification	0.11	9.0x	0.28%
Gradual group-wise sparsification	0.05	20x	1.07%
Accelerating the second and the third convolutional layers of AlexNet			
Training with fixed sparsity patterns	0.12	8.7x	1.54%
Training with fixed sparsity patterns	0.35	2.9x	0.36%
Training with fixed sparsity patterns	0.54	1.9x	-0.53%
Group-wise sparsification + Fine-tuning	0.2	5x	1.50%
Group-wise sparsification + Fine-tuning	0.3	3.33x	1.17%
Group-wise sparsification + Fine-tuning	0.5	2x	0.57%
Gradual group-wise sparsification	0.12	8.5x	1.04%
Accelerating all five convolutional layers of AlexNet			
Training with fixed sparsity patterns	0.34	3.0x	1.34%
Gradual group-wise sparsification	0.31	3.2x	1.43%

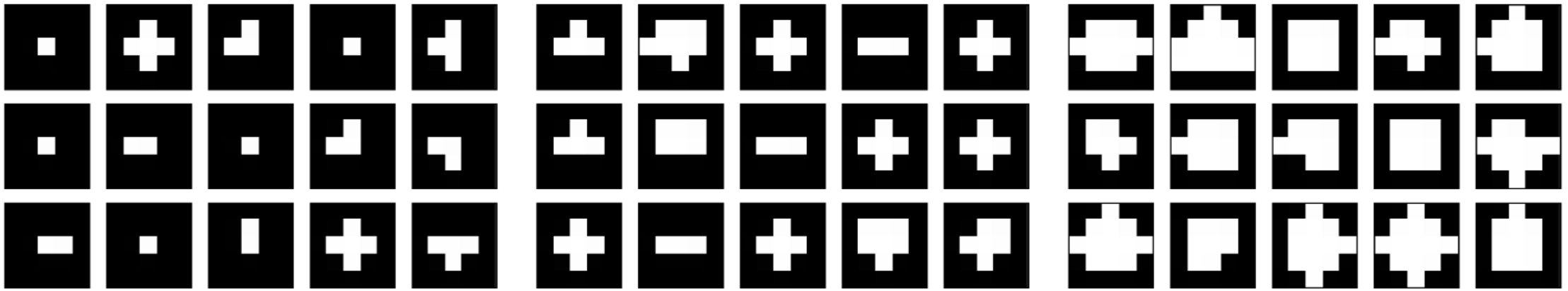
[Lebedev, 2016]

AlexNet Experiments (2nd Layer)

$$\Omega_{2,1}^{T}(K) = \lambda \sum_{i,j,s} \min(\|\Gamma_{ijs}\|, \theta)$$

- For each τ (i.e., sparsity level), we pick λ that results in the minimal accuracy drop after sparsification before fine-tuning. After picking the optimal λ, we perform fine-tuning.





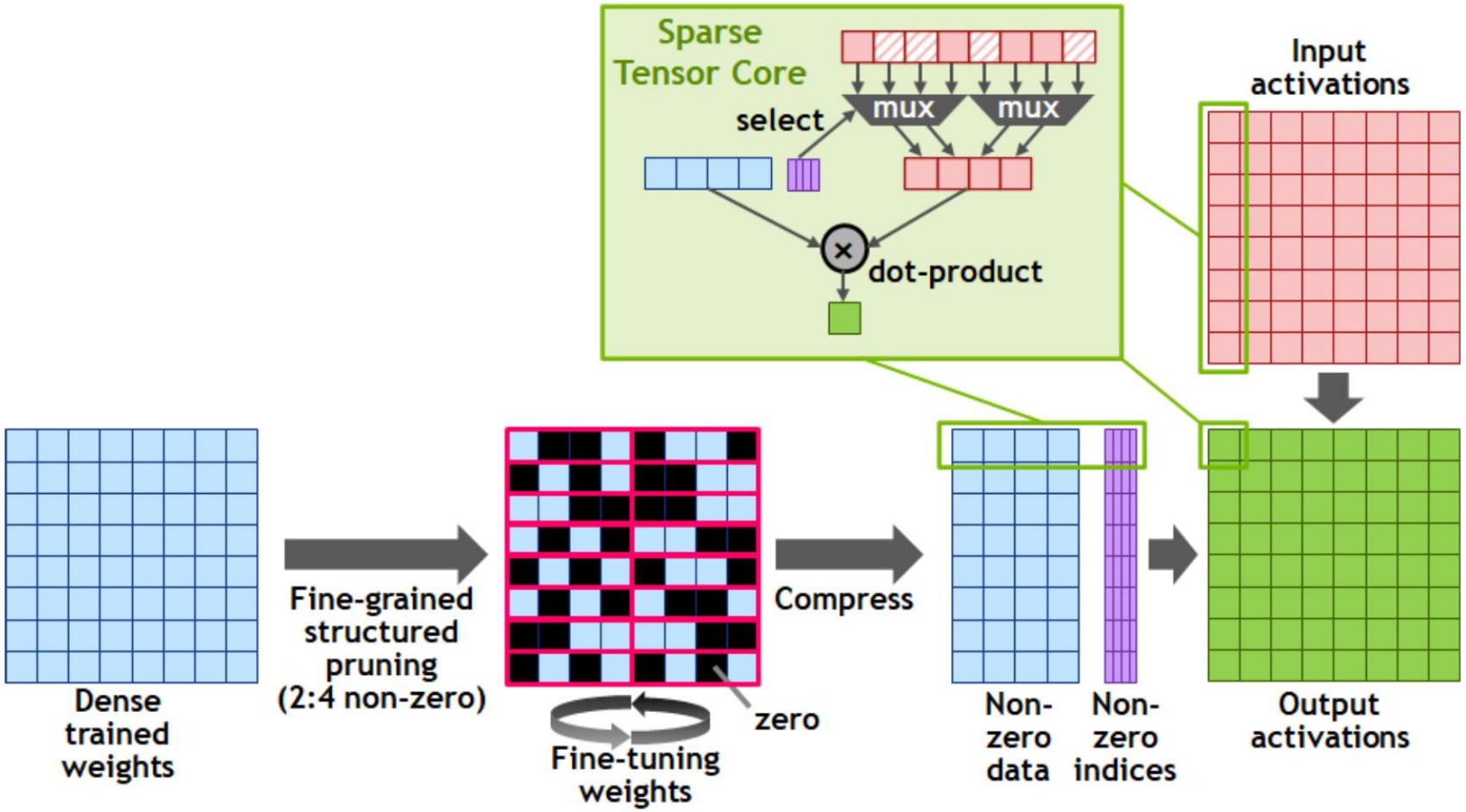
(a) sparsity $1 - \tau = 0.9$ (b) sparsity $1 - \tau = 0.8$ (c) sparsity $1 - \tau = 0.6$

Pruning: Agenda

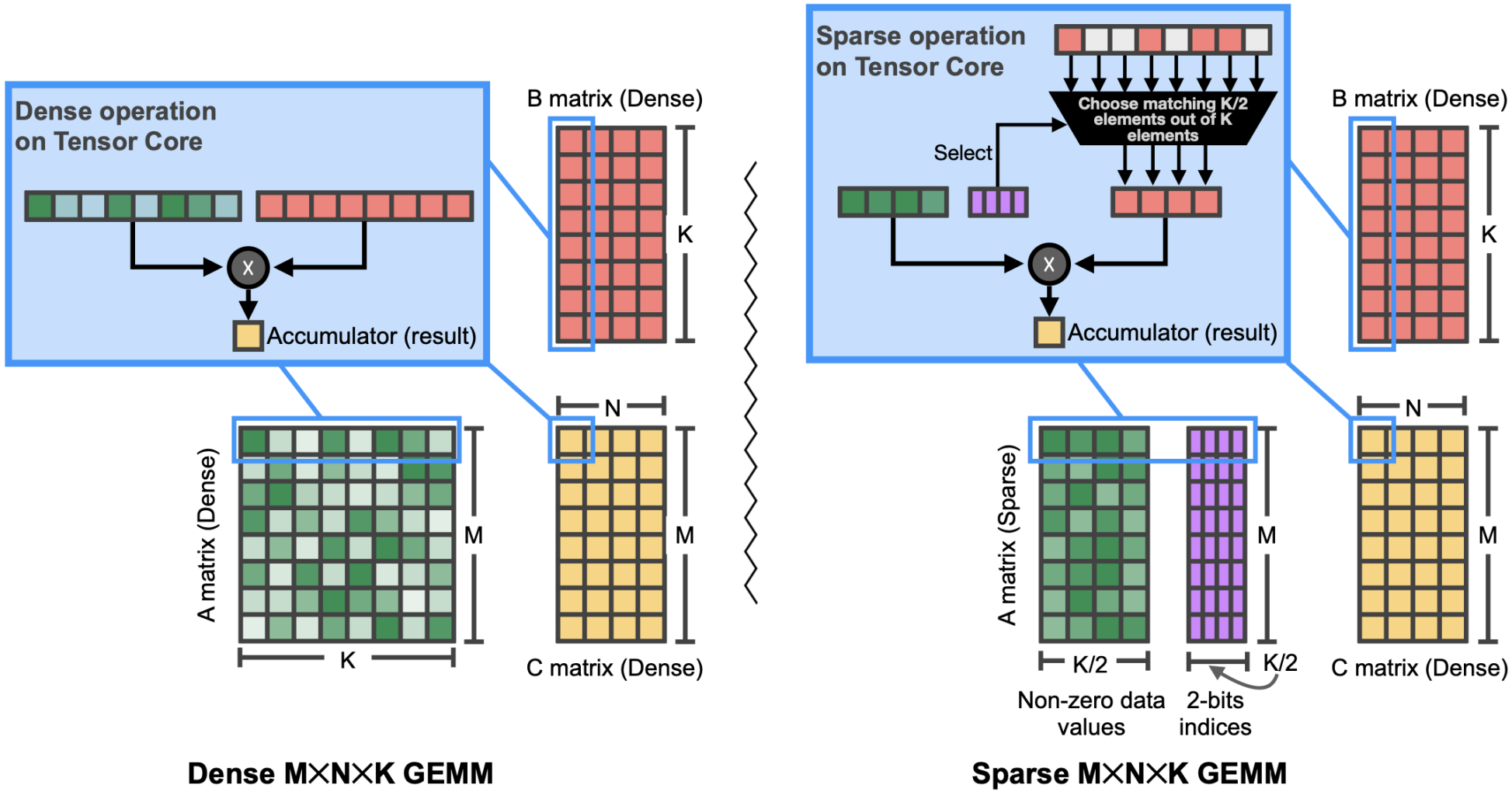
- Introduction
- Magnitude-based pruning
- Structured pruning
  - Groupwise brain damage
  - NVIDIA’s 2:4 rule pruning
- Lottery ticket pruning

[https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf]

Zero Skipping in Nvidia A100



Dense vs. Sparse Tensor Core



[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

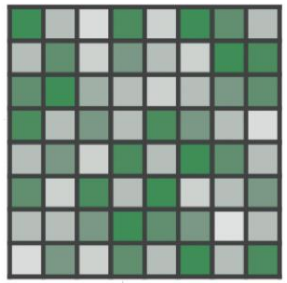
RECIPE FOR 2:4 SPARSE NETWORK TRAINING

1) Train (or obtain) a dense network

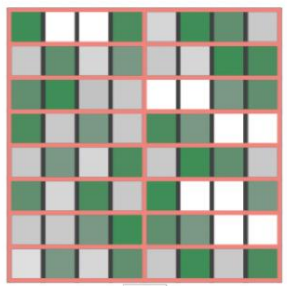
2) Prune for 2:4 sparsity

3) Repeat the original training procedure

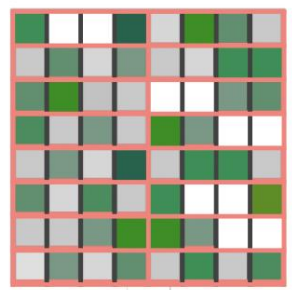
- Same hyper-parameters as in step-1
- Initialize to weights from step-2
- Maintain the 0 pattern from step-2: no need to recompute the mask



Dense weights



2:4 sparse weights

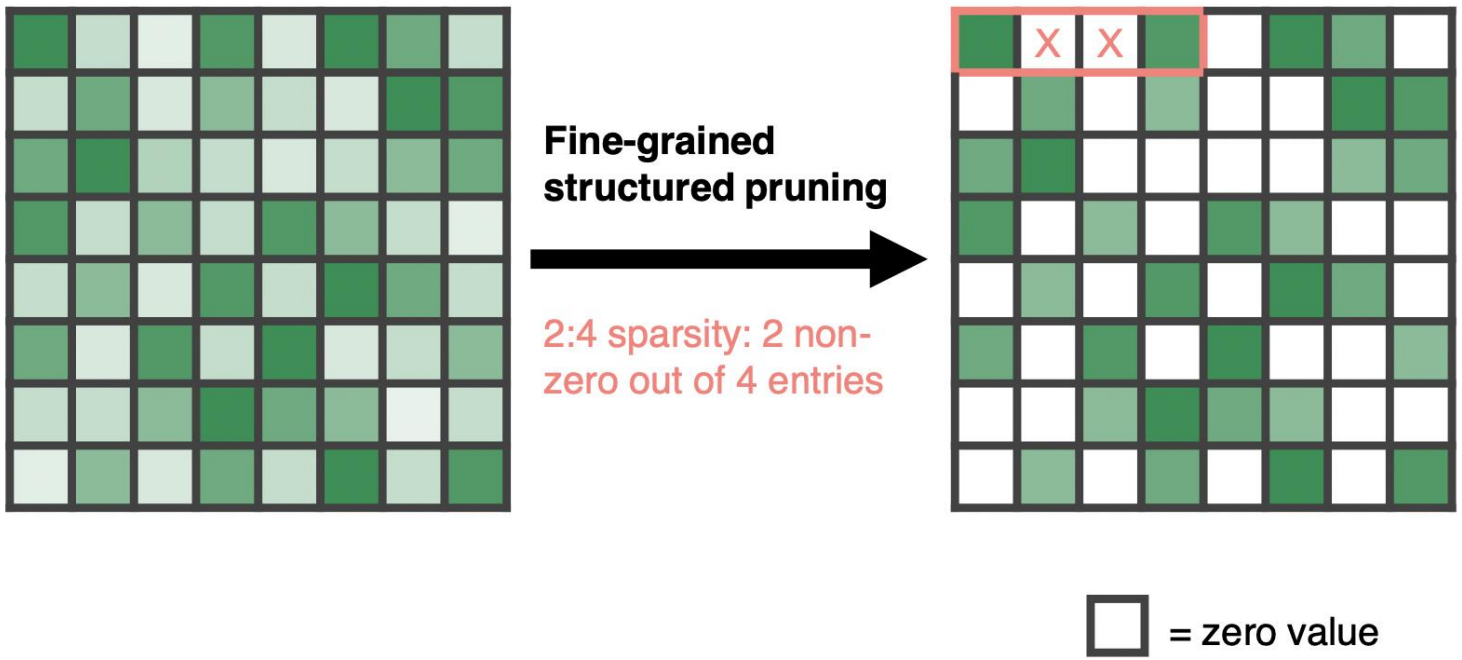


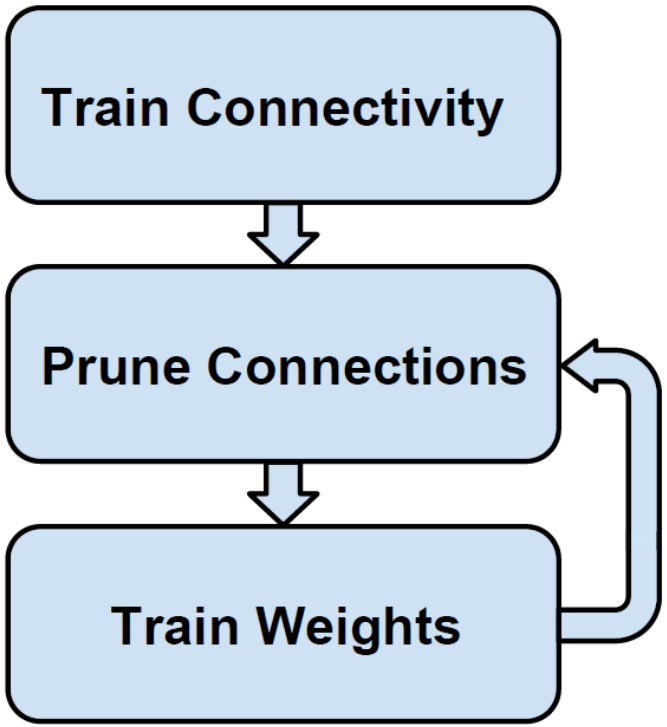
Retrained 2:4 sparse weights

[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

RECIPE STEP 2: PRUNE WEIGHTS

At Most 2 Non-zeros in Every Contiguous Group of 4 Values

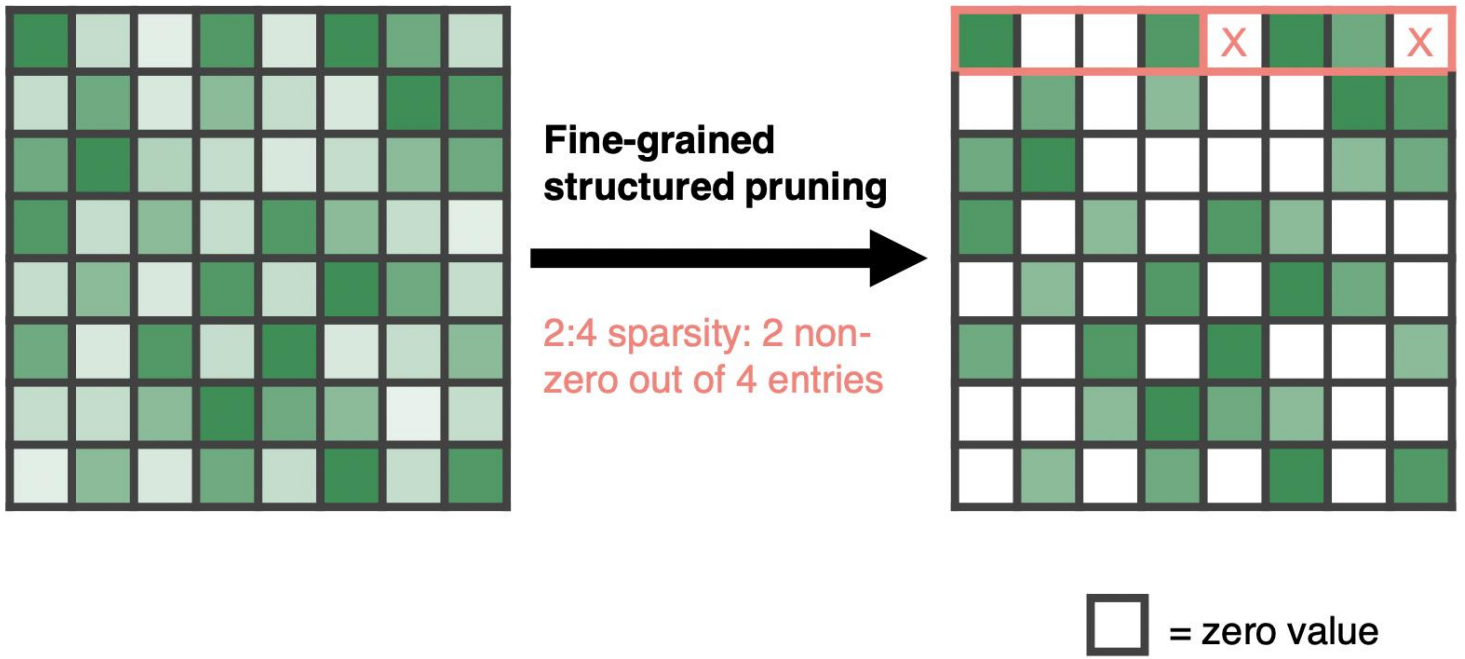


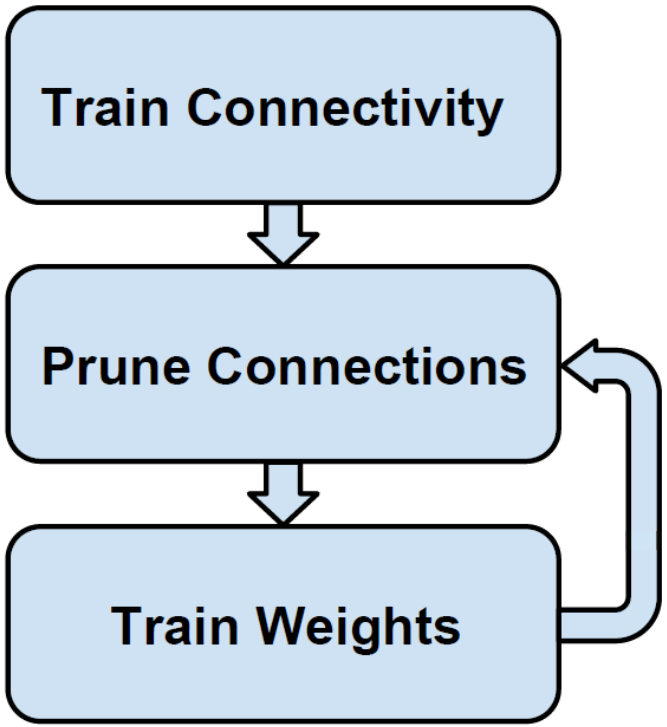


[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

RECIPE STEP 2: PRUNE WEIGHTS

At Most 2 Non-zeros in Every Contiguous Group of 4 Values





Before Step 2, Permutation Can be Applied to Avoid Excessive Pruning under 2:4

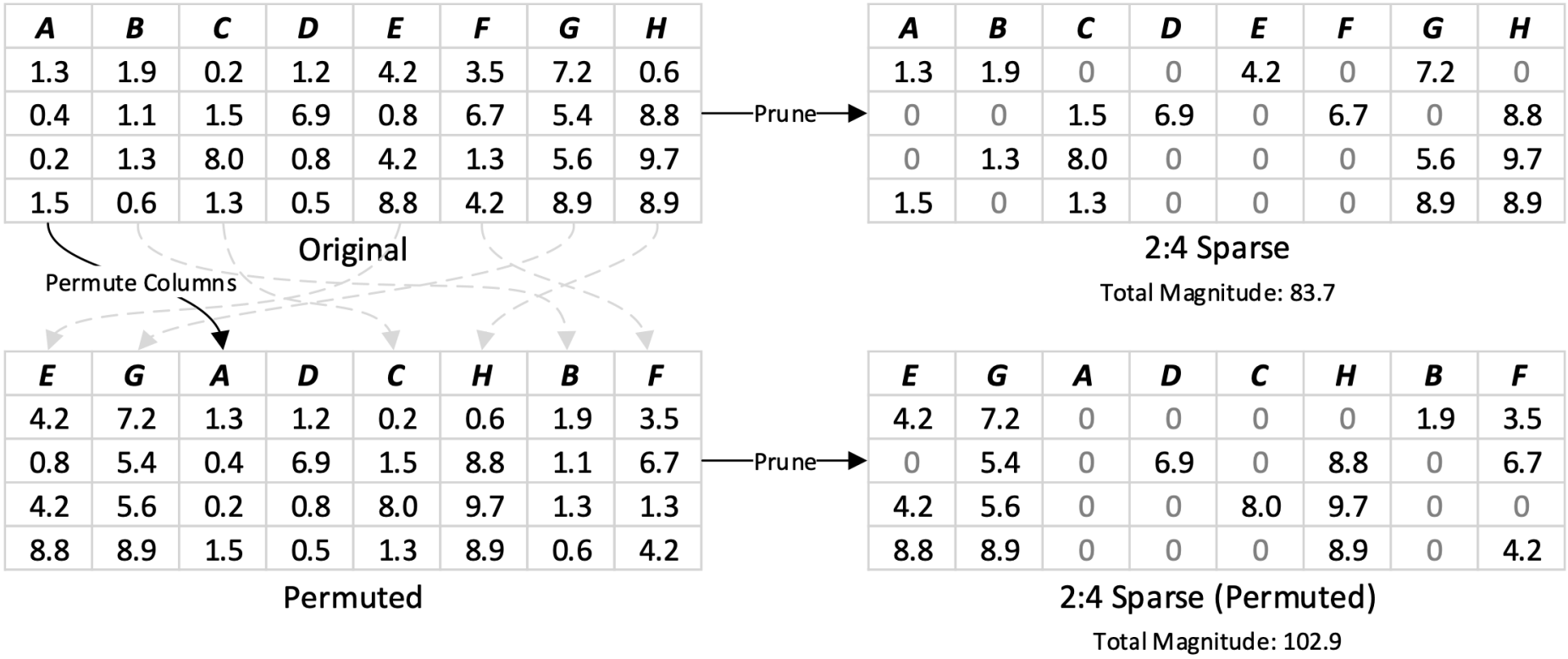
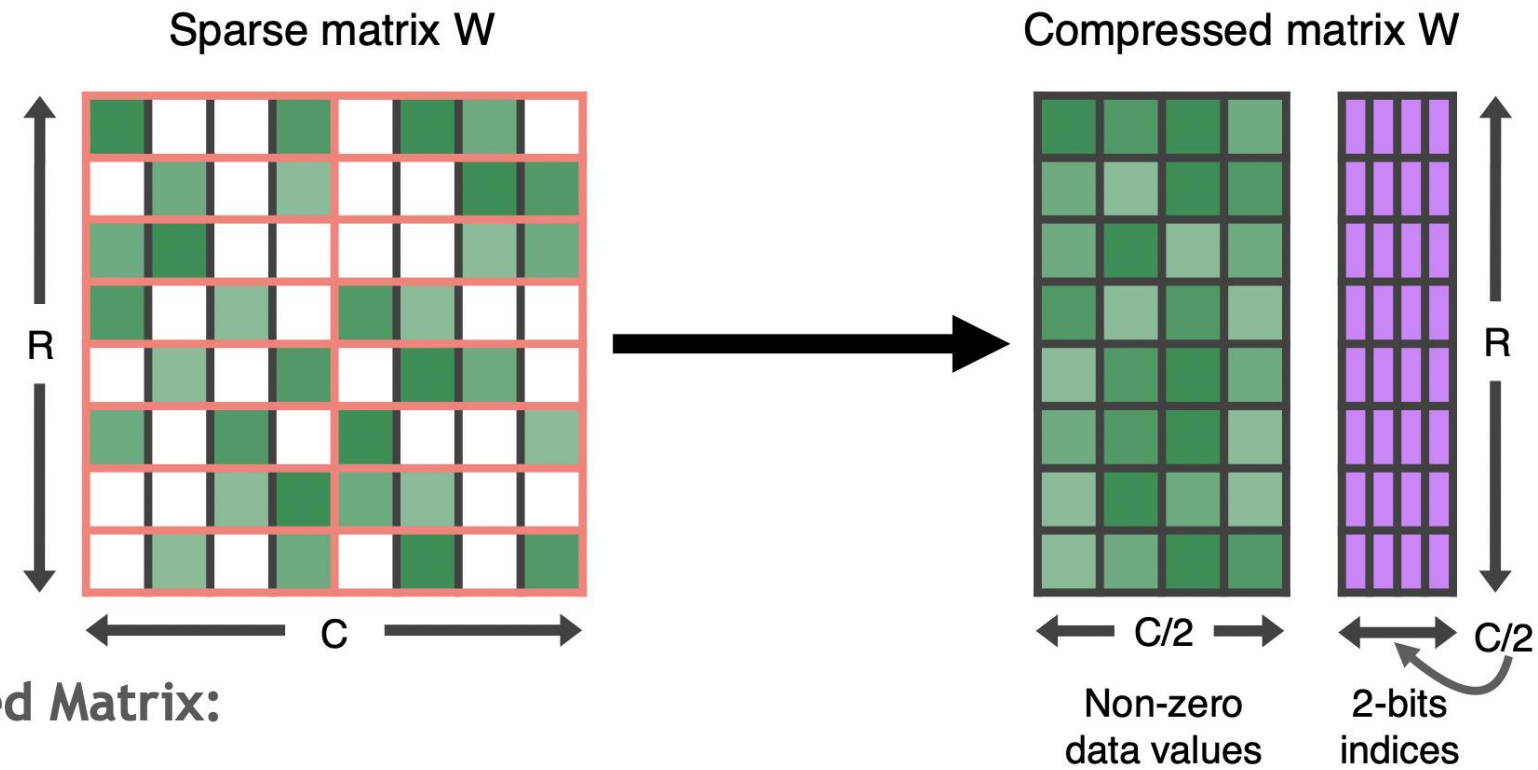


Fig. 6. Permuting columns of a weight matrix prior to pruning the matrix can reduce the effect of the 2:4 sparsity constraint on weight magnitude.

[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

2:4 COMPRESSED MATRIX FORMAT

At most 2 non-zeros in every contiguous group of 4 values



Compressed Matrix:

Data: ½ size

Metadata: 2b per non-zero element

16b data => 12.5% overhead

8b data => 25% overhead

2X Speedup Enabled by Sparse Tensor Core and 2:4 Sparsity

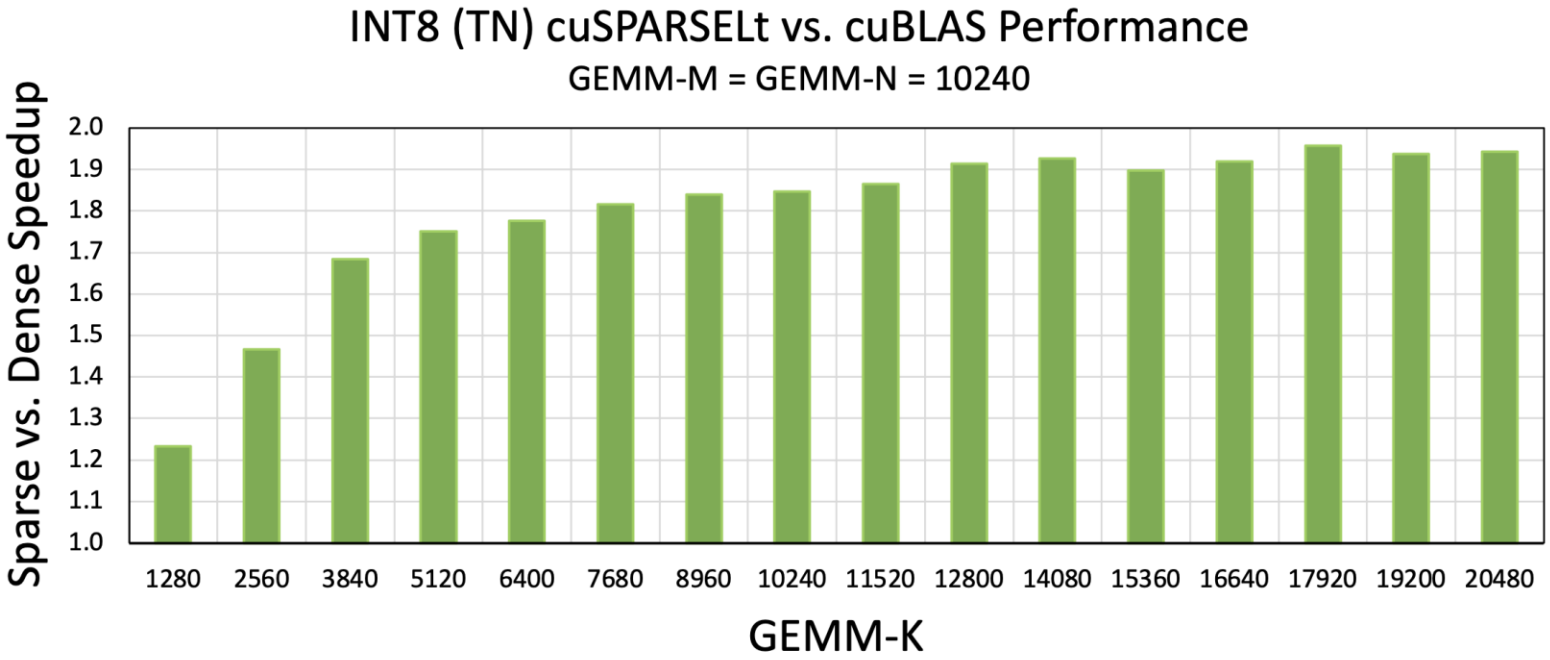


Fig. 3. Comparison of sparse and dense INT8 GEMMs on NVIDIA A100 Tensor Cores. Larger GEMMs achieve nearly a 2× speedup with Sparse Tensor Cores.

[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

IMAGE CLASSIFICATION

ImageNet

Network	Accuracy				
	Dense FP16	Sparse FP16		Sparse INT8	
ResNet-34	73.7	73.9	0.2	73.7	-
ResNet-50	76.6	76.8	0.2	76.8	0.2
ResNet-101	77.7	78.0	0.3	77.9	-
ResNeXt-50-32x4d	77.6	77.7	0.1	77.7	-
ResNeXt-101-32x16d	79.7	79.9	0.2	79.9	0.2
DenseNet-121	75.5	75.3	-0.2	75.3	-0.2
DenseNet-161	78.8	78.8	-	78.9	0.1
Wide ResNet-50	78.5	78.6	0.1	78.5	-
Wide ResNet-101	78.9	79.2	0.3	79.1	0.2
Inception v3	77.1	77.1	-	77.1	-
Xception	79.2	79.2	-	79.2	-
VGG-16	74.0	74.1	0.1	74.1	0.1
VGG-19	75.0	75.0	-	75.0	-

Permutation Helps on Efficient Networks

Network	Accuracy (FP16)		
	Dense	2:4 Sparse	
		Default	Permuted
MobileNet v2	71.55	69.56	71.56
SqueezeNet v1.0	58.09	54.08	58.38
SqueezeNet v1.1	58.21	56.96	58.23
MNASNet 1.0	73.24	71.99	73.27
ShuffleNet v2	68.32	66.97	68.42
EfficientNet B0	77.25	75.98	77.29
EfficientNet-WideSE B0	77.63	76.64	77.63

Pruning: Agenda

- Introduction
- Magnitude-based pruning
- Structured pruning
  - Groupwise brain damage
  - NVIDIA’s 2:4 rule pruning
- Lottery ticket pruning

Lottery Ticket Hypothesis

Can we train a sparse neural network from scratch?

Train Sparse Neural Network From Scratch

- Neural Network Pruning shows that
  - **a neural network can be reduced in size.**

- Question: **Can we directly train this sparse neural network from scratch?**

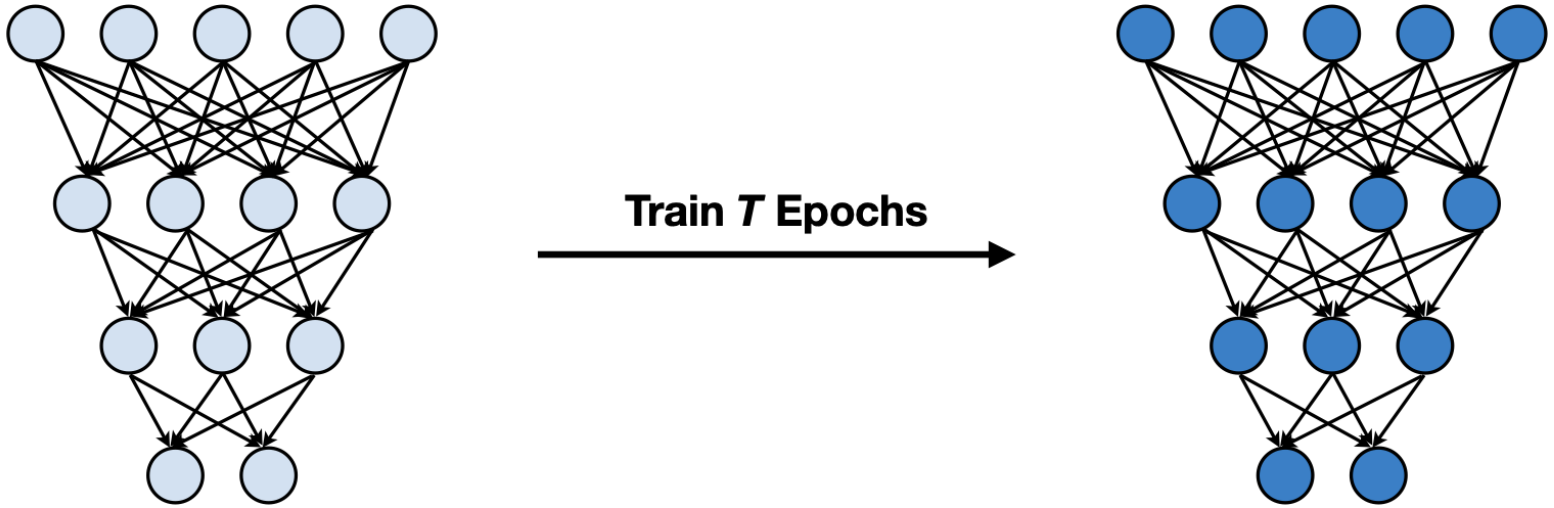
- Contemporary experience tells us that
  - **the architectures uncovered by pruning are harder to train from the start,**
  - **reaching lower accuracy than the original networks**

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

The Lottery Ticket Hypothesis

A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

—The Lottery Ticket Hypothesis

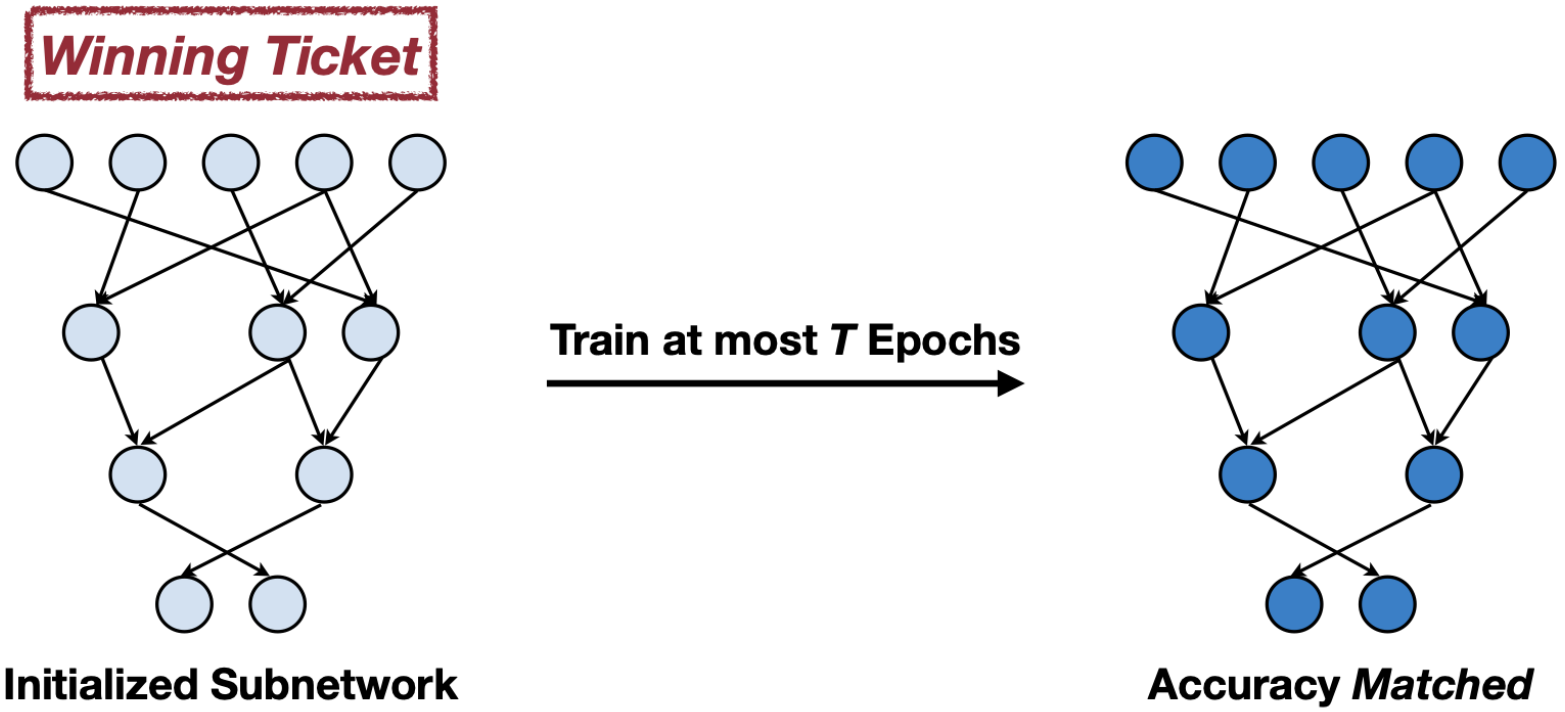


The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

The Lottery Ticket Hypothesis

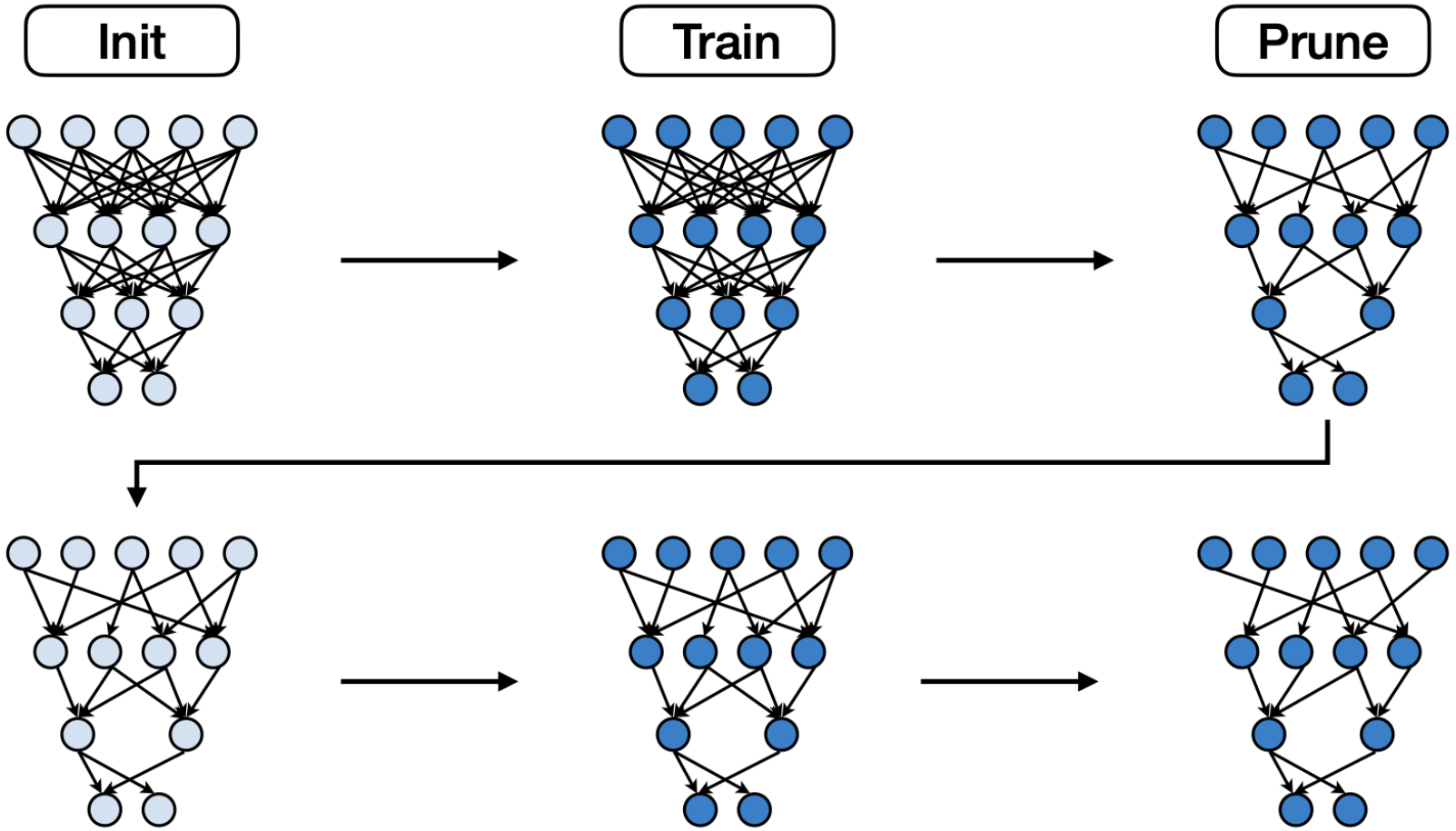
A randomly-initialized, dense neural network contains ***a subnetwork*** *that is initialized such that—when* ***trained in isolation*** *—it can* ***match the test accuracy*** *of the original network after training for* ***at most the same number of iterations.***

—The Lottery Ticket Hypothesis



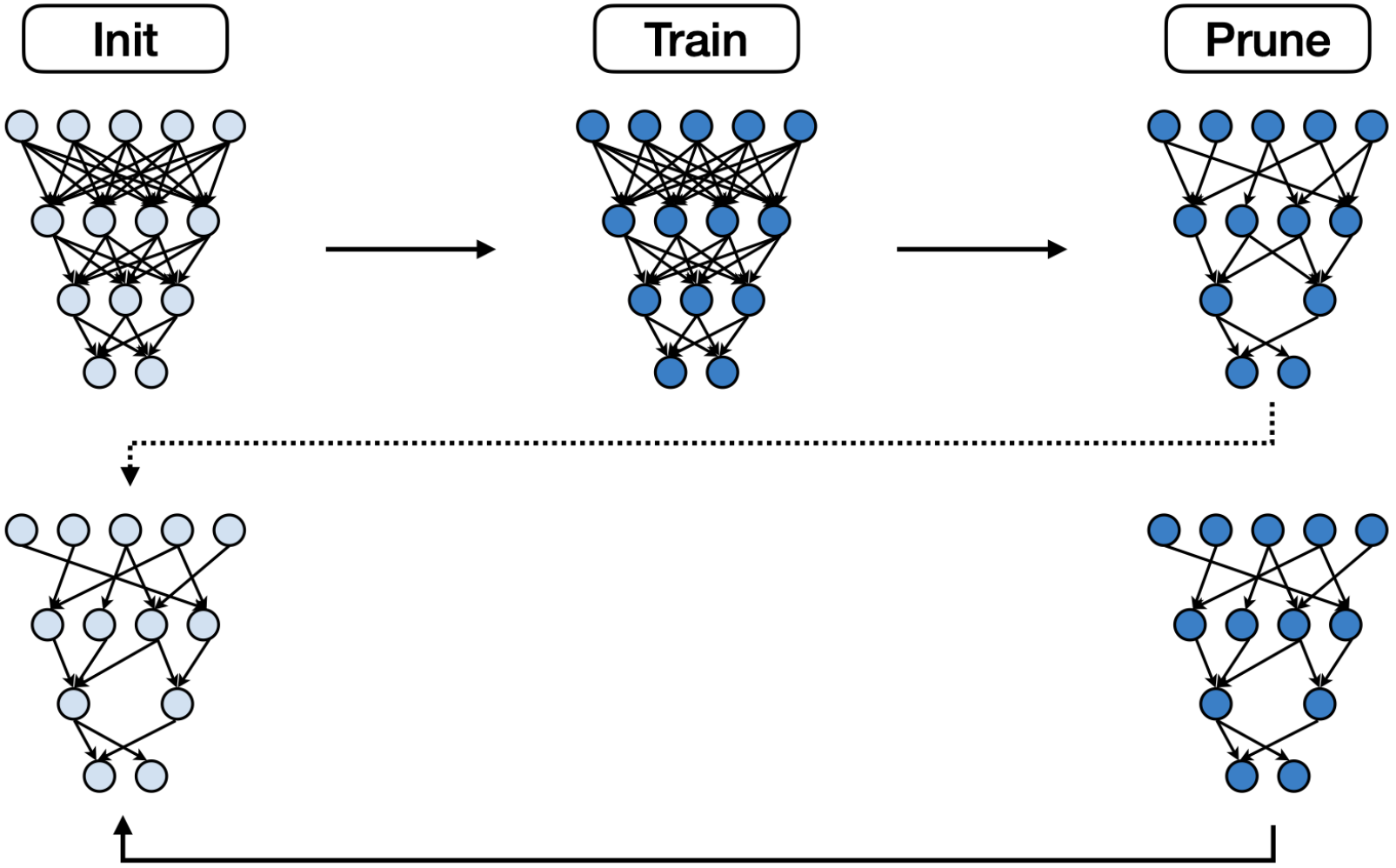
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



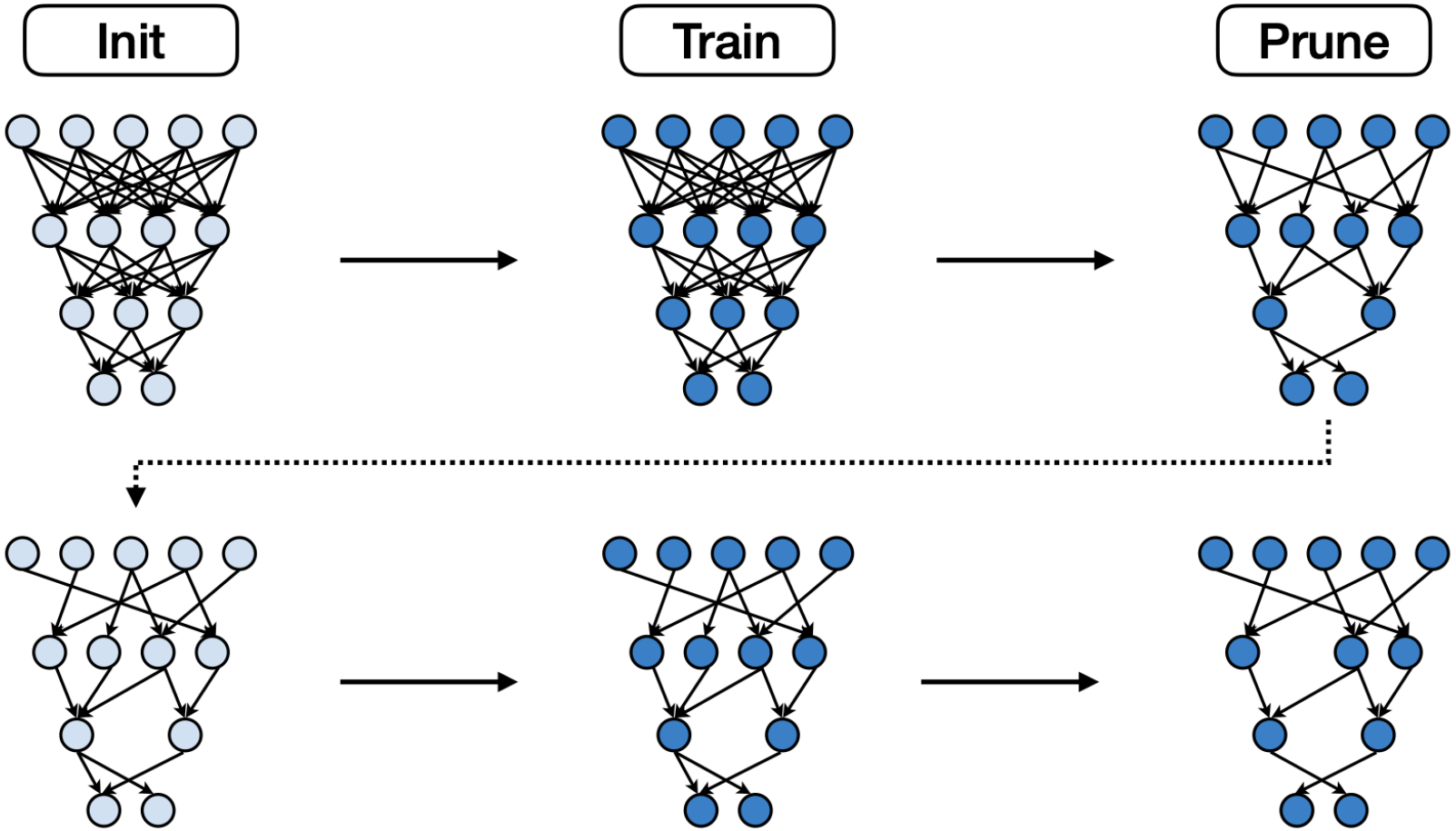
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



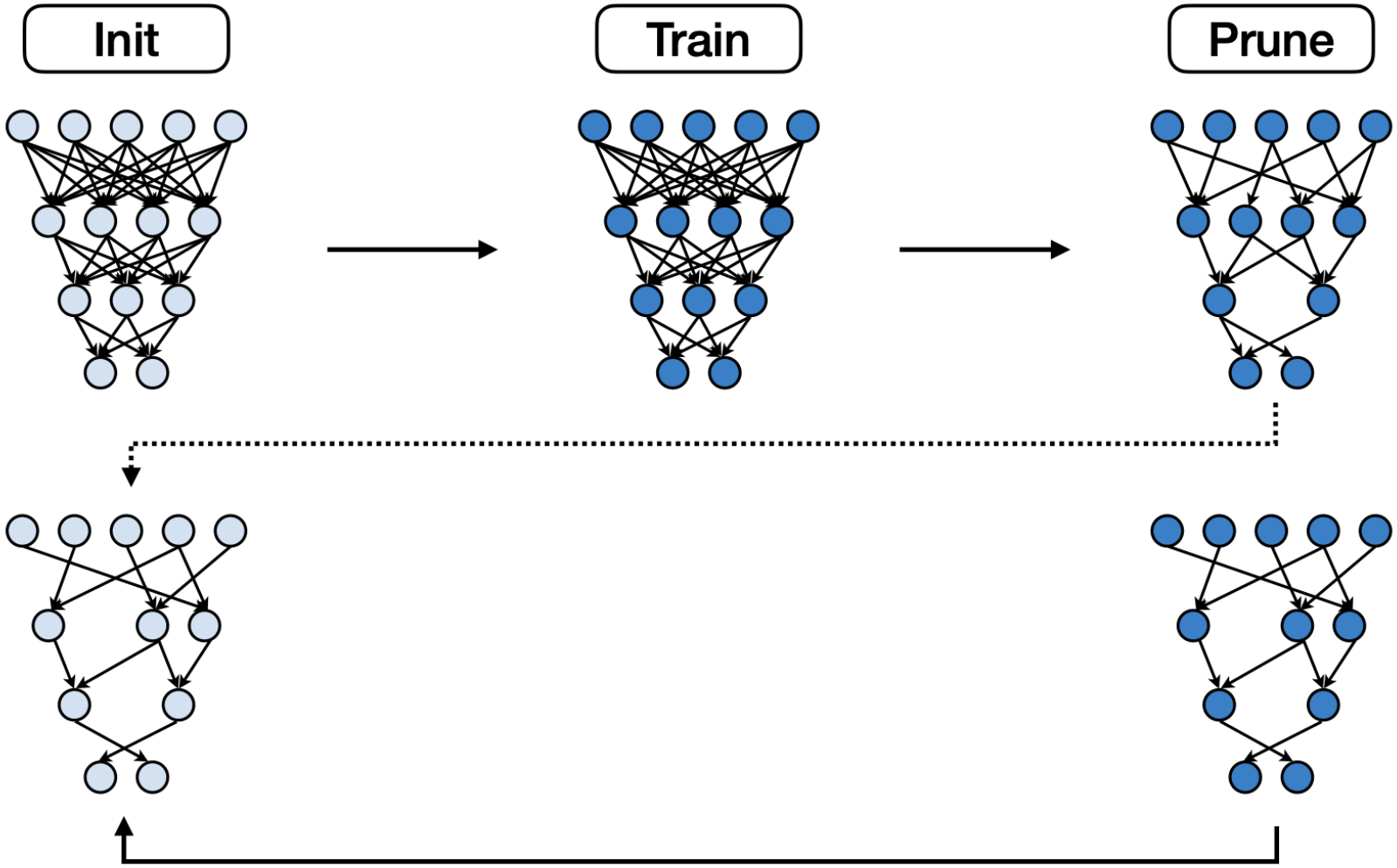
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



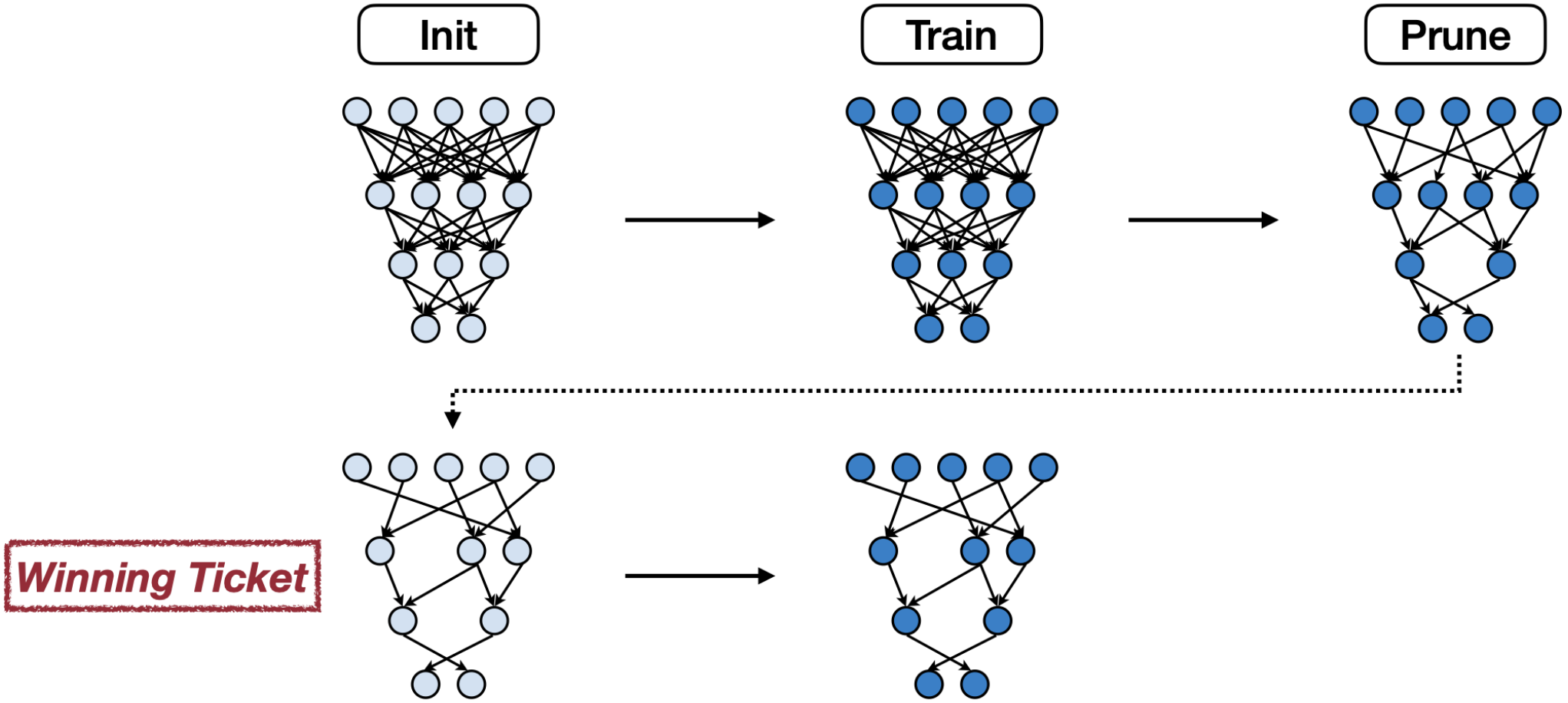
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning

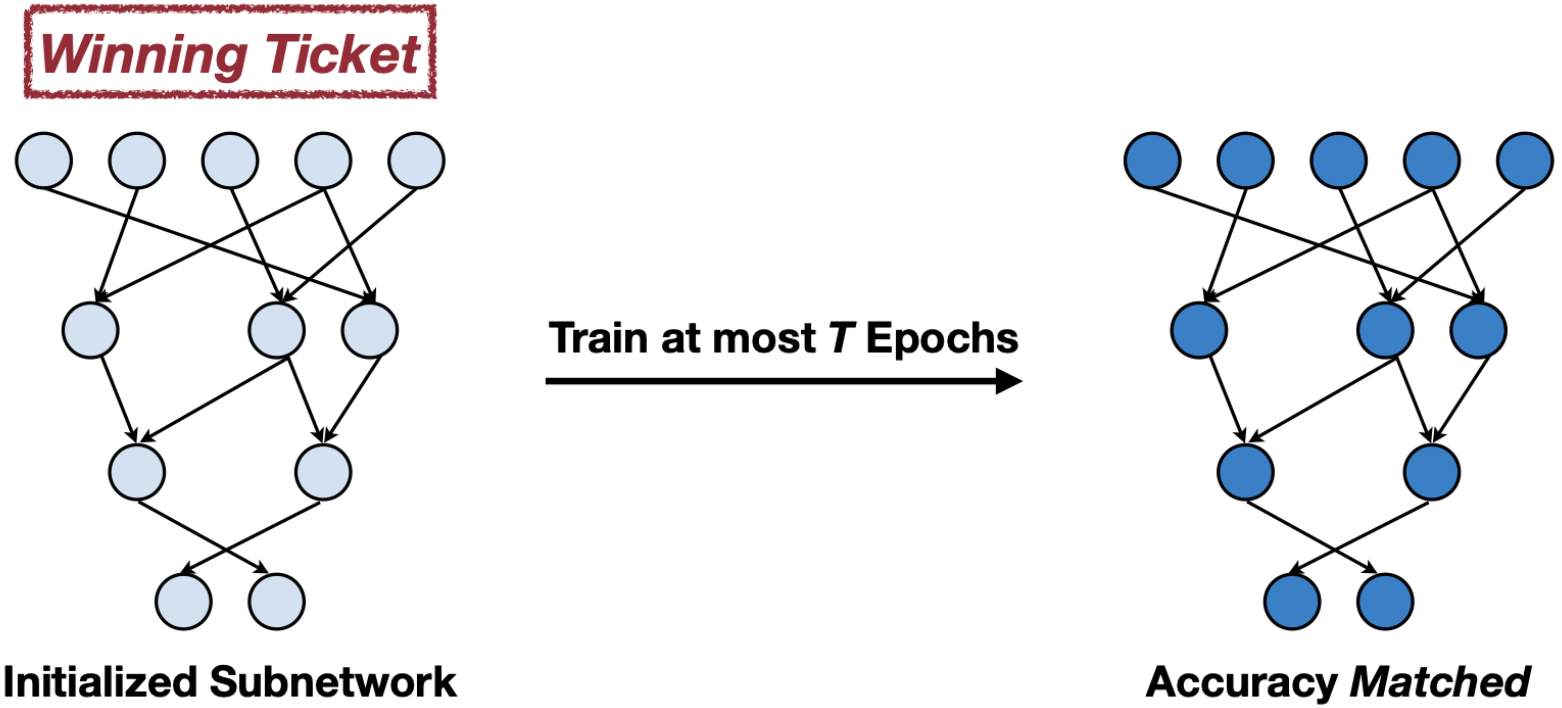


The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

The Lottery Ticket Hypothesis

A randomly-initialized, dense neural network contains ***a subnetwork*** *that is initialized such that—when* ***trained in isolation*** *—it can* ***match the test accuracy*** *of the original network after training for* ***at most the same number of iterations.***

—The Lottery Ticket Hypothesis



The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Pruning: Agenda

- Introduction
- Magnitude-based pruning
- Structured pruning
  - Groupwise brain damage
  - NVIDIA’s 2:4 rule pruning
- Lottery ticket pruning

Weekly Lecture / Lab Schedule

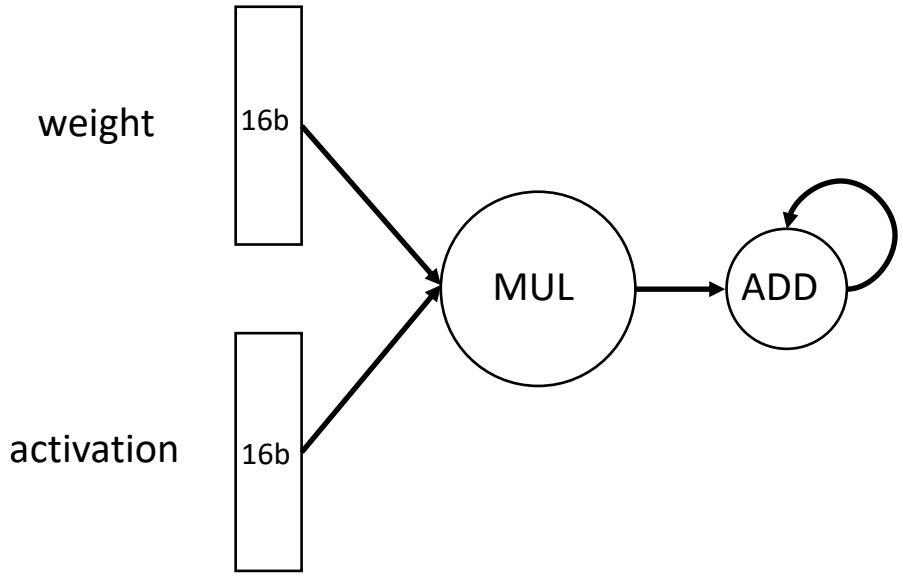
- W1 (March 6) Class introduction / (March 8) Orientation & team formation
- W2 13 Verilog 1 Combinational circuits / 15 Verilog 1 (tool installation, adder & multiplier combinational logic)
- W3 20 Verilog 2 Sequential circuits / 22 Verilog 2 (memory i/o, FSM sequential logic)
- W4 27 AI application introduction 1, Amaranth introduction 1 / 29 Amaranth (tool installation, MAC, adder tree)
- W5 4/3 AI application introduction 2, Amaranth introduction 2 (memory i/o, FSM sequential logic) / 5 Amaranth (PE)
- W6 10 AI application introduction 3, Neural network accelerator 1 / 12 Amaranth (stacked PEs)
- W7 17 Neural network accelerator 2 / 19 Amaranth (stacked PEs)
- W8 24 **Mid-term exam** / 26 Amaranth (stacked PEs)
- W9 5/1 Reading data from memory 1 (VA2PA, interconnect) / 3 Convolution lowering
- W10 8 Reading data from memory 2 (DRAM main memory), Compressing networks 1 (pruning) / 10 Tiling
- W11 15 Compressing networks 2 (pruning, low precision) / 17 PyTorch model – Amaranth simulator communication
- W12 22 Zero-skipping & low-precision hardware accelerator / 24 Quantization, project introduction
- W13 29 Invited talks (commercial solutions: Rebellions, Furiosa AI) / 31 Homework Q&A
- W14 6/5 **Final exam** / 7 Project Q&A
- W15 12 Claim & Project Q&A / 14 Project submission

Quantization: Agenda

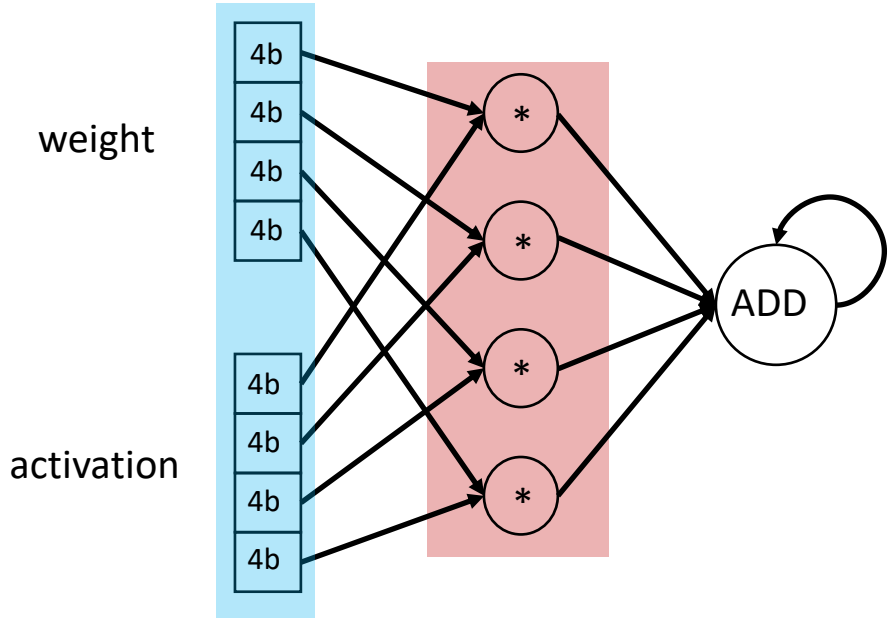
- Introduction
  - FP8 model training
  - Quantization-aware training (QAT)
- Quantization interval learning
  - PACT: Parameterized Clipping Activation Function
  - LSQ: Learned Step size Quantization
- Training stabilization and precision learning
  - NIPQ: Noise Injection Pseudo Quantization for Automated DNN Optimization
- Binary neural network
- Concluding remarks

Benefits of Low Precision Data Type

- Reduction in memory cost (# accesses and size)
- Reduction in computation cost (energy and gate count)



Conventional convolution with 16bit data

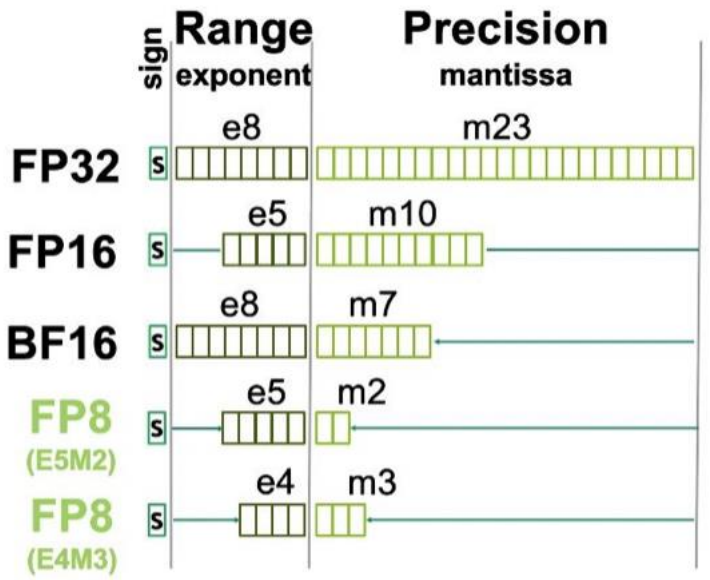


Convolution with narrow (4bit) data

[https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf]
[https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/]

Precision in Nvidia A100 and H100

- 8-bit floating point (FP8) is new in H100
- Int4 and binary precision is available



	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x	-	-
	IEEE FP64		19.5	1x	-	-

8b Floating Point (FP8) Are Supported by NVIDIA H100, Tesla Dojo, ARM, Qualcomm, …

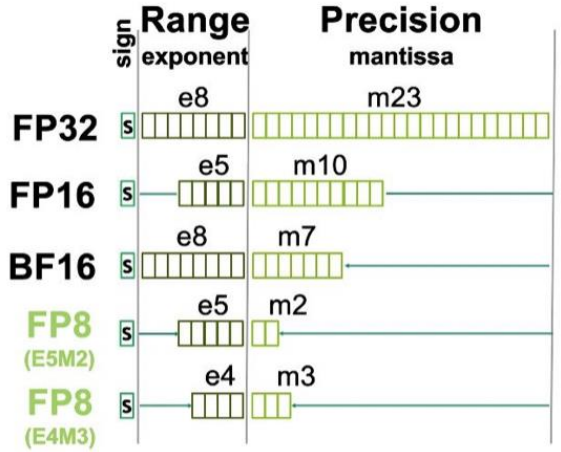
12 Sep 2022

FP8 Formats For Deep Learning

Paulius Micikevicius, Dusan Stosic, Patrick Judd, John Kamalu, Stuart Oberman, Mohammad Shoeybi, Michael Siu, Hao Wu
NVIDIA
{pauliusm, dstosic, pjudd, jkamalu, soberman, mshoeybi, msiu, skyw}@nvidia.com

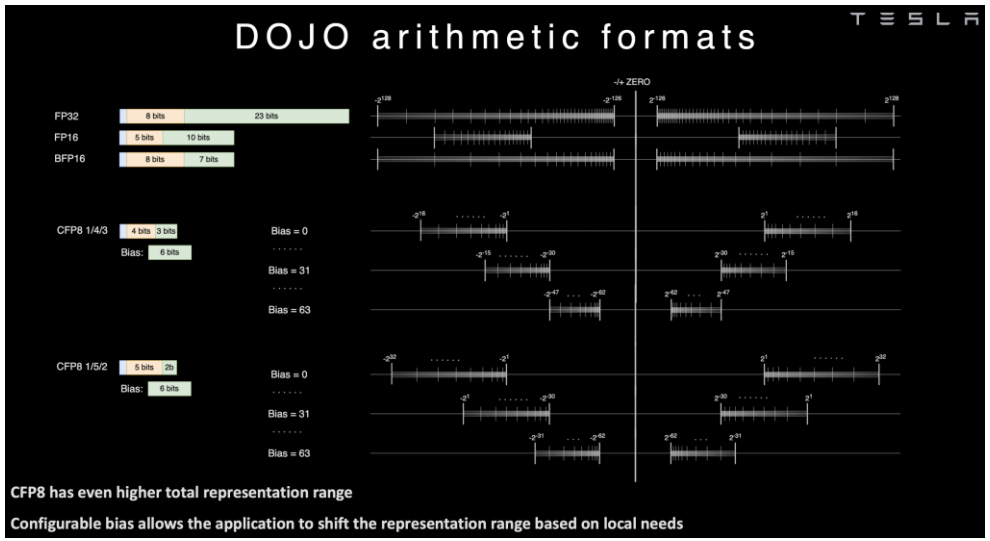
Neil Burgess, Sangwon Ha, Richard Grisenthwaite
Arm
{neil.burgess, sangwon.ha, richard.grisenthwaite}@arm.com

Marius Cornea, Alexander Heinecke, Naveen Mellempudi, Pradeep Dubey
Intel
{marius.cornea, alexander.heinecke, naveen.k.mellempudi, pradeep.dubey}@intel.com

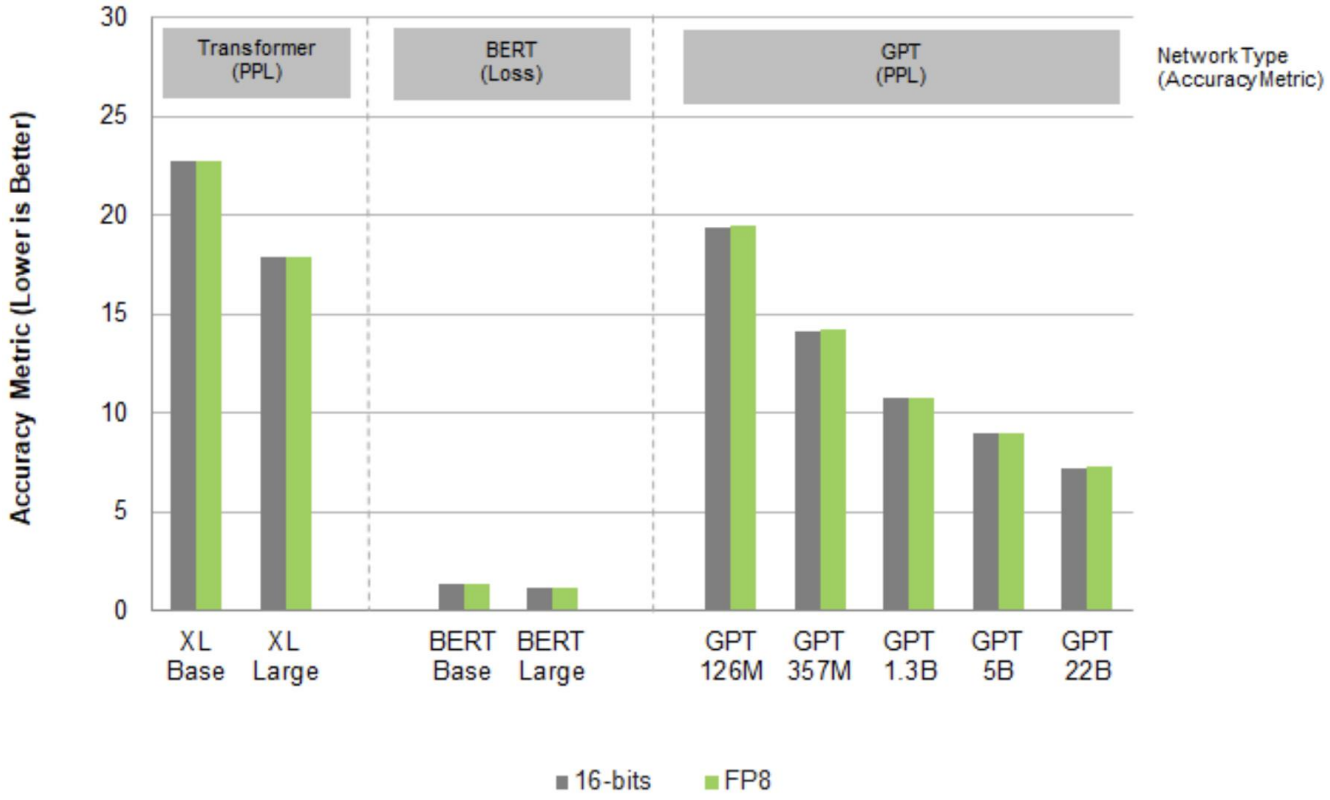


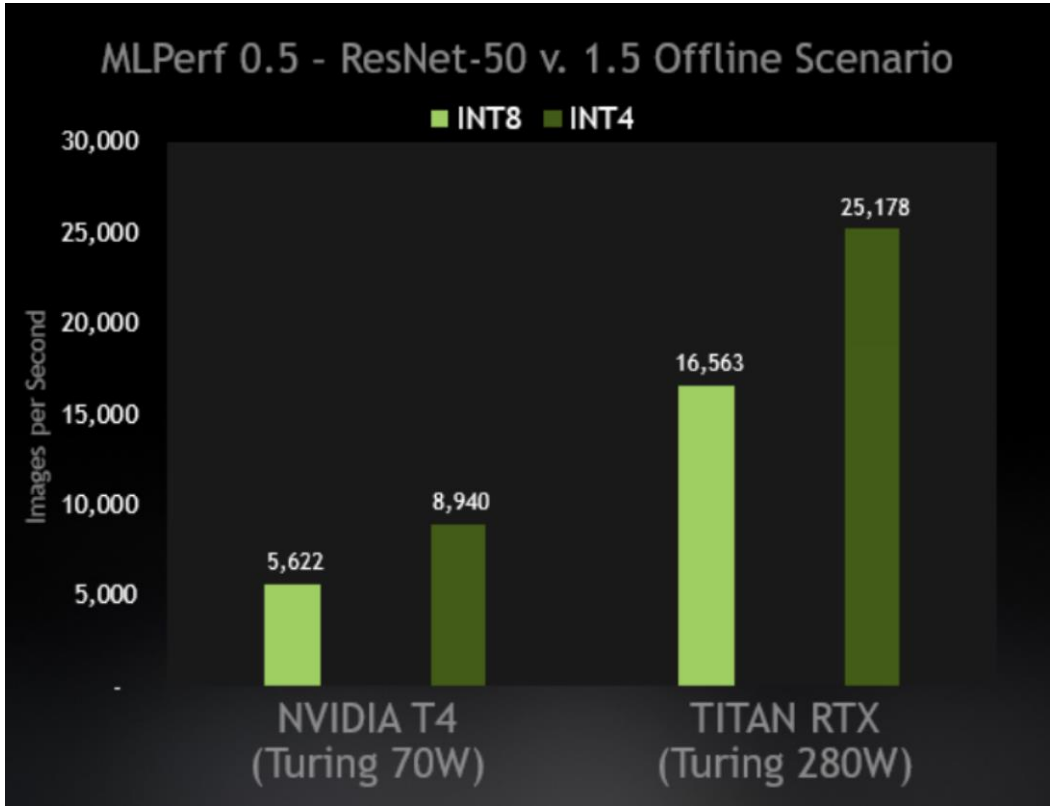
FP8 Quantization: The Power of the Exponent

Andrey Kuzmin*, Mart Van Baalen*, Yuwei Ren, Markus Nagel, Jorn Peters, Tijmen Blankevoort
Qualcomm AI Research†
{akuzmin,mart,ren,markusn,jpeters,tijmen}@qti.qualcomm.com



FP8 Offers 2X Speedup without Quality Loss
Int4 Can Offer 2X Speedup w.r.t. Int8



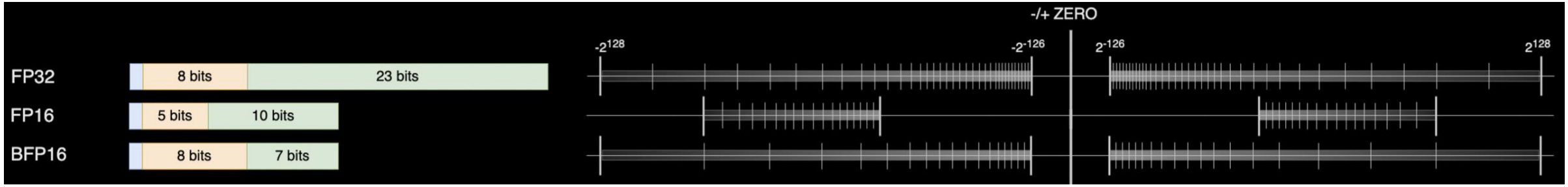


https://developer.nvidia.com/blog/nvidia-arm-and-intel-publish-fp8-specification-for-standardization-as-an-interchange-format-for-ai/

https://developer.nvidia.com/blog/int4-for-ai-inference/

[Dojo The Microarchitecture of Tesla's Exa-Scale Computer, Hot Chips 2022]

32 and 16 bit Floating Point Representations

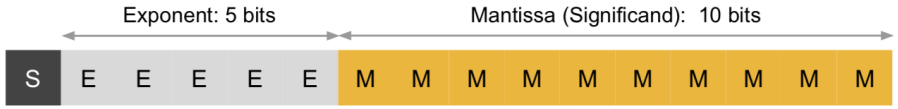


float32: Single-precision IEEE Floating Point Format

Range: ~$1e^{-38}$ to ~$3e^{38}$

float16: Half-precision IEEE Floating Point Format

Range: ~$5.96e^{-8}$ to 65504

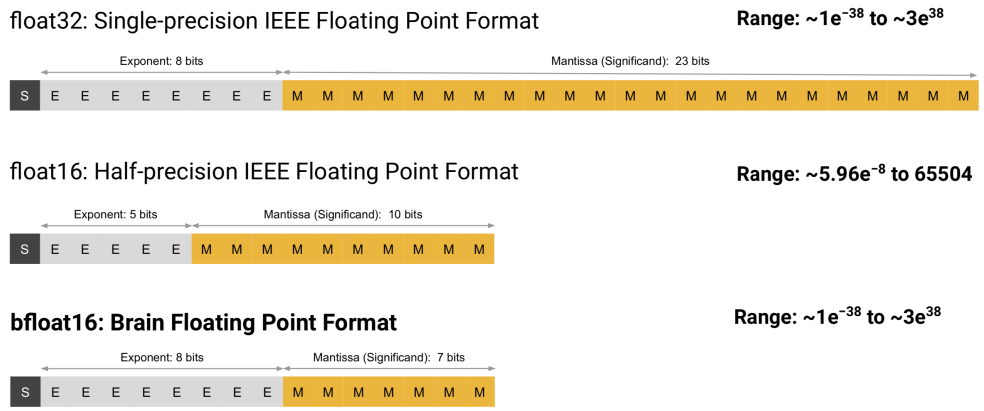


bfloat16: Brain Floating Point Format

Range: ~$1e^{-38}$ to ~$3e^{38}$

32 bit Floating Point Value Example

- [S,E,M] represents $(-1)^S \times (1.0 + 0.M) \times 2^{E\text{-bias}}$
  - bias = 127 for 32 bit floating point values
- Example, a decimal number, 365
  - S = 0
  - We need a representation of $1.M \times 2^{E\text{-bias}}$
  - $365 = 1.M\ (=1.42578125) \times 2^8$
  - E-bias = 8, bias = 127. Thus, E = 135 in decimal = 10000111 in binary
  - M = 0.42578125 in decimal = 0.01101101 in binary



- [S 1bit, E 8bits, M 23bits] = 0 10000111 **01101101**00000000000000

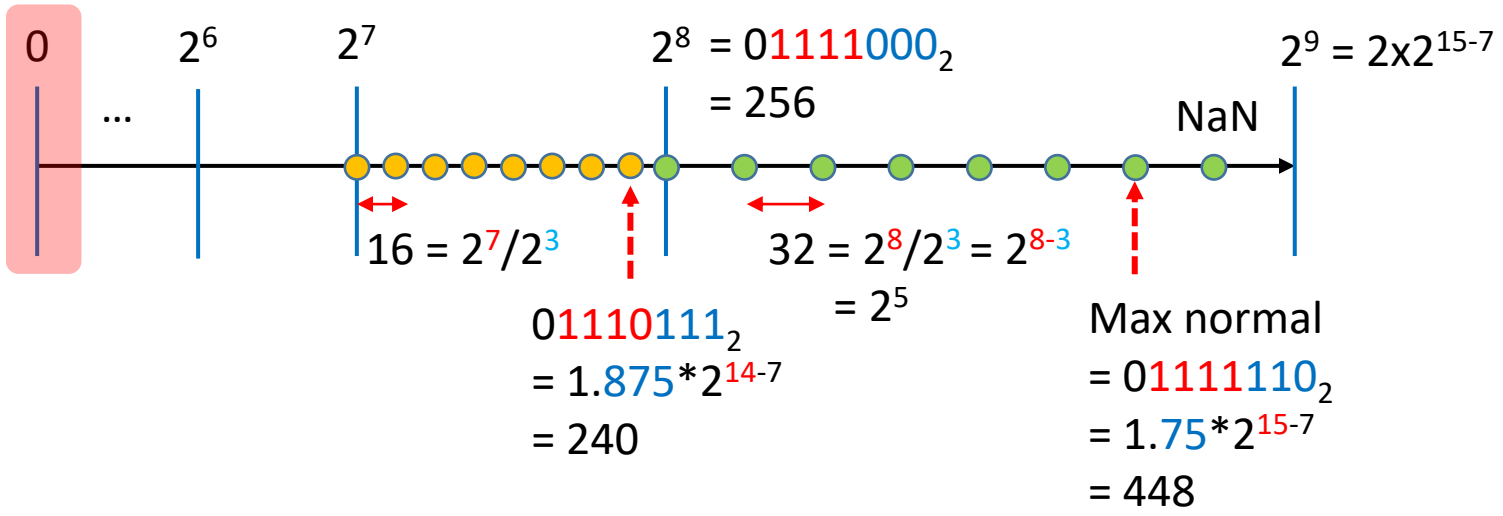
FP32

BF16

FP8 Representation

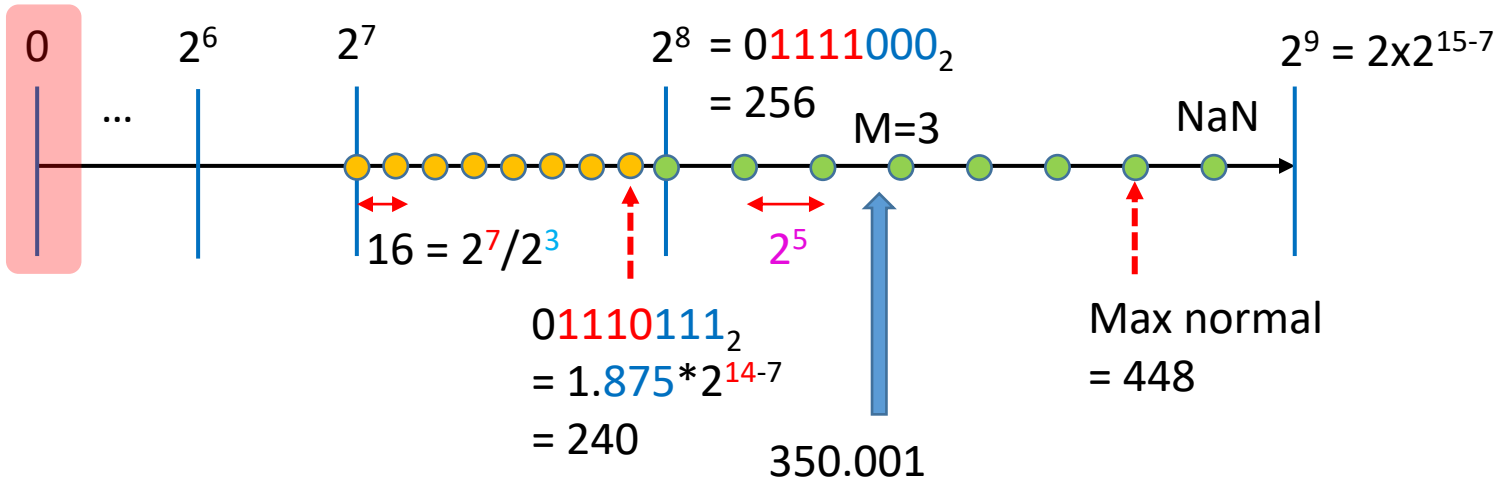
	E4M3	E5M2
Exponent bias	7	15
Infinities	N/A	$S.11111.00_2$
NaN	$S.1111.111_2$	$S.11111.\{01, 10, 11\}_2$
Zeros	$S.0000.000_2$	$S.00000.00_2$
Max normal	$S.1111.110_2 = 1.75 * 2^8 = 448$	$S.11110.11_2 = 1.75 * 2^{15} = 57,344$
Min normal	$S.0001.000_2 = 2^{-6}$	$S.00001.00_2 = 2^{-14}$
Max subnorm	$S.0000.111_2 = 0.875 * 2^{-6}$	$S.00000.11_2 = 0.75 * 2^{-15}$
Min subnorm	$S.0000.001_2 = 2^{-9}$	$S.00000.01_2 = 2^{-16}$

- E4M3 case
  - [S,E,M] represents $(-1)^S$ x (1.0 + 0.M) x $2^{E\text{-bias}}$



Representing a Real Value in FP8 i.e., FP8 Quantization

- E4M3 case
  - [S,E,M] represents $(-1)^S$ x (1.0 + 0.M) x $2^{E\text{-bias}}$

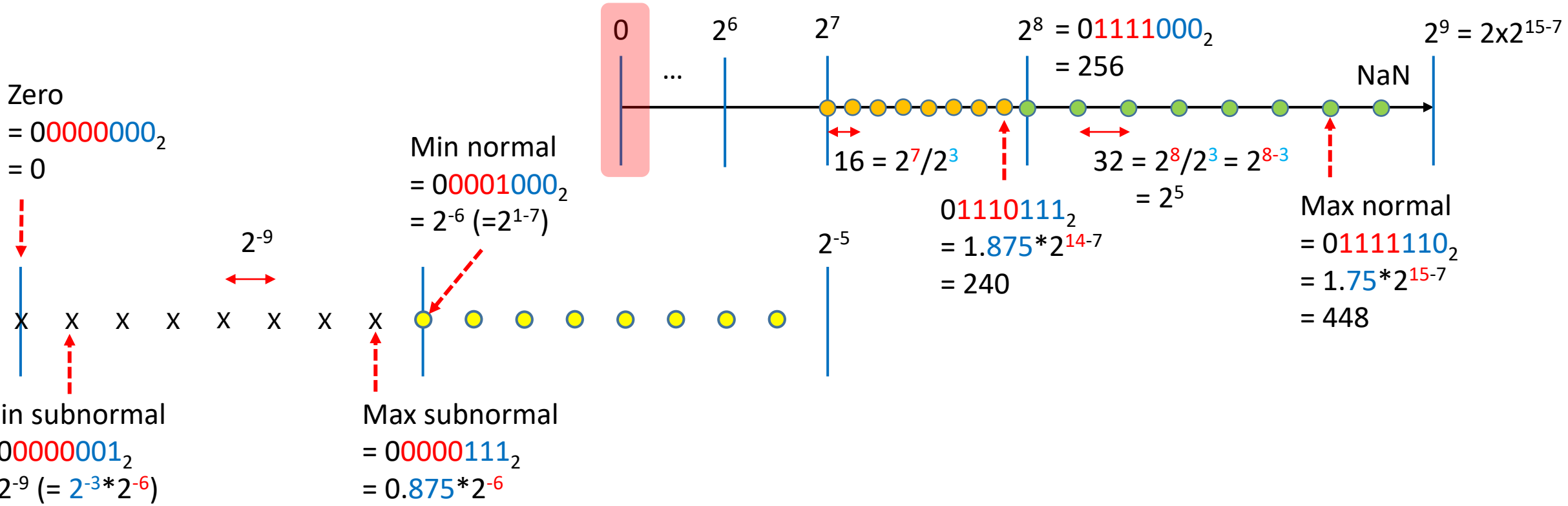


→ floor($\log_2$ 350.001) = 8 (=E-bias)
→ 350.001 – 256 = 93.999
→ 93.999/2^5 = 2.93 (rounding to 3)
→ M = 3
→ In FP8, 01111011_2 (= 352 in decimal)

FP8 Representation

	E4M3	E5M2
Exponent bias	7	15
Infinities	N/A	$S.11111.00_2$
NaN	$S.1111.111_2$	$S.11111.\{01, 10, 11\}_2$
Zeros	$S.0000.000_2$	$S.00000.00_2$
Max normal	$S.1111.110_2 = 1.75 * 2^8 = 448$	$S.11110.11_2 = 1.75 * 2^{15} = 57,344$
Min normal	$S.0001.000_2 = 2^{-6}$	$S.00001.00_2 = 2^{-14}$
Max subnorm	$S.0000.111_2 = 0.875 * 2^{-6}$	$S.00000.11_2 = 0.75 * 2^{-15}$
Min subnorm	$S.0000.001_2 = 2^{-9}$	$S.00000.01_2 = 2^{-16}$

- E4M3 case
  - [S,E,M] represents $(-1)^S$ x (1.0 + 0.M) x $2^{E\text{-bias}}$
    - If E = 0, then it's called ***subnormal***, we use $(-1)^S$ x (0.0 + 0.M) x $2^{1+E\text{-bias}}$



Training FP8 Model

- Train FP8 model (E5M2) with the same hyperparameters of FP16 training
- FP8 model is expected to be widely used due to its simplicity in training
  - It is enough to train FP8 model as usual

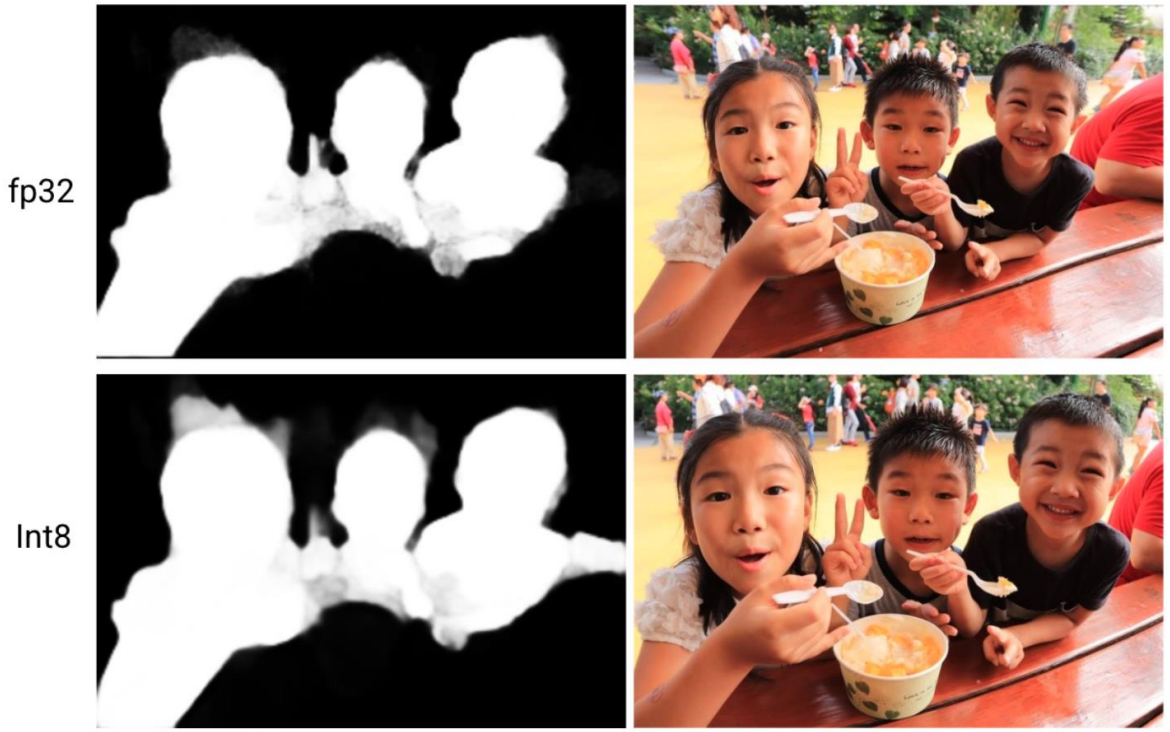
Model	Baseline	FP8
Transformer-XL Base	22.98	22.99
Transformer-XL Large	17.80	17.75
GPT 126M	19.14	19.24
GPT 1.3B	10.62	10.66
GPT 5B	8.94	8.98
GPT 22B	7.21	7.24
GPT 175B	6.65	6.68

Model	Baseline	FP8
VGG-16	71.27	71.11
VGG-16 BN	73.95	73.69
Inception v3	77.23	77.06
DenseNet 121	75.59	75.33
DenseNet 169	76.97	76.83
Resnet18	70.58	70.12
Resnet34	73.84	73.72
Resnet50 v1.5	76.71	76.76
Resnet101 v1.5	77.51	77.48
ResNeXt50	77.68	77.62
Xception	79.46	79.17
MobileNet v2	71.65	71.04
DeiT small	80.08	80.02

FP8 is Expected to Be Part of the Standard

- Currently, int8 is popular in mobile and server
- However, some models still adopt higher precision
  - E.g., FP16 is still used for semantic segmentation in Google and other companies
- FP8 is promising due to
  - No quality loss
  - Easy training

[Ten Lessons From Three Generations Shaped Google's TPUv4i]



Quantization: Agenda

- Introduction
  - FP8 model training
  - Quantization-aware training (QAT)
- Quantization interval learning
  - PACT: Parameterized Clipping Activation Function
  - LSQ: Learned Step size Quantization
- Training stabilization and precision learning
  - NIPQ: Noise Injection Pseudo Quantization for Automated DNN Optimization
- Binary neural network
- Concluding remarks

[Jacob, 2018]

Quantization-Aware Training (QAT) in a Nutshell

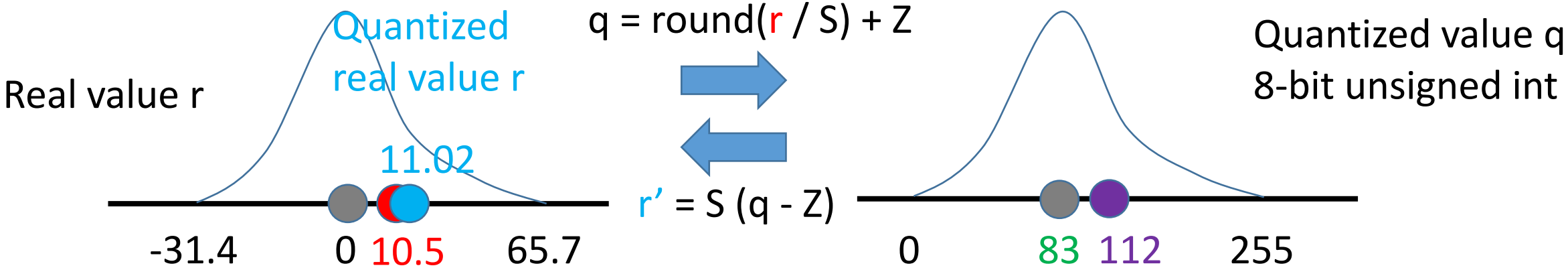
- Step 1 Train a network in full precision (FP)
- Step 2 Quantize FP weights and execute the model (quantizing activations)
  - In PyTorch, the quantized model still maintains, as floating point, the data types of weights and activations
  - We emulate low-precision operation on floating point values, e.g., they can have only integer values between -128 and 127 (after min/max based scaling) in int8 quantization
  - We call this method fake quantization since we do not execute the quantized model on the real hardware having the capability of low-precision computation
- Step 3 Back-propagate the error (using quantized weights/activations and full-precision gradient) and update the FP weights
  - If small updates are applied to the quantized weights, the updates may be lost after the subsequent quantization. Thus, FP weights are used to accumulated small weight updates. Then, in forward pass, the FP weights are quantized to give the quantized weights.
- Repeat Steps 2 and 3

Original Real Value vs. Faked Real Value (in Fake Quantization)

- Calculating scale (S=real range/int range) and zero (Z=int value for real zero) for int8

-31.4 → 0	-31.4 = S*(0 – Z)	SZ=31.4	Z=round(31.4/0.38)=83
65.7 → 255	65.7 = S*(255 –Z)	65.7+31.4=255S	S=97.1/255=0.38

- Original real value r = 10.5 is quantized to an integer 112 (=round(10.5/0.38) +83=29+83) which corresponds to a real value r' (= S (q – Z) = 0.38 * (112 – 83) = 11.02)
- In training, a faked, i.e., quantized real value r' (=11.02) is used instead of the original real value r (= 10.5)



Straight-Through Estimator (STE)

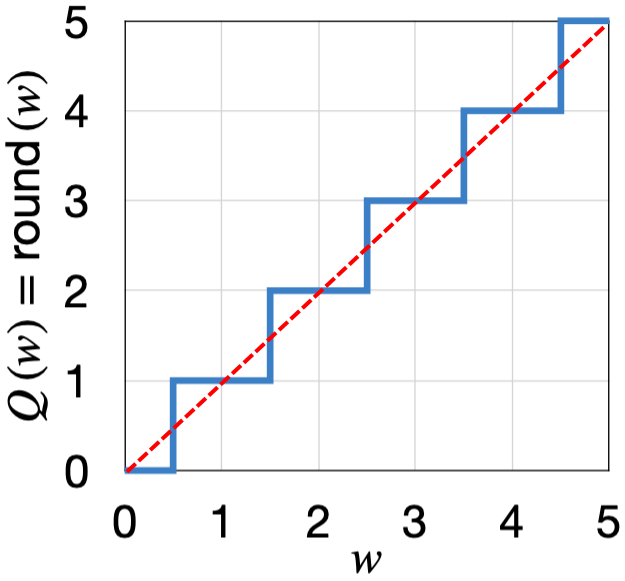
- Quantization is discrete-valued, and thus the derivative is 0 almost everywhere.

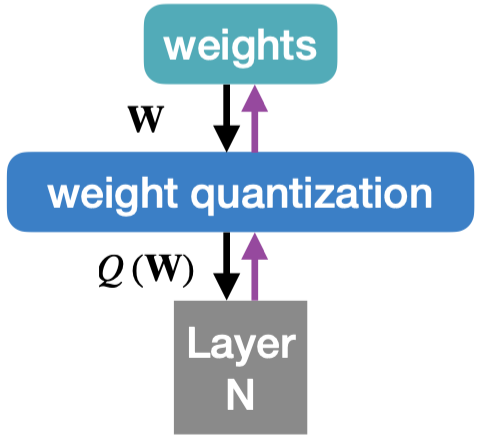
$$\frac{\partial Q(W)}{\partial W} = 0$$

  - The neural network will learn nothing since gradients become 0 and the weights won't get updated.

$$g_{\mathbf{W}} = \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial Q(\mathbf{W})} \cdot \underbrace{\frac{\partial Q(\mathbf{W})}{\partial \mathbf{W}}}_{=1} = 0$$

STE approximates quantization operation with an identity function





- Straight-Through Estimator (STE) simply passes the gradients through the quantization as if it had been the *identity* function.

$$g_{\mathbf{W}} = \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial Q(\mathbf{W})}$$

Neural Networks for Machine Learning [Hinton *et al.*, Coursera Video Lecture, 2012]
Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation [Bengio, arXiv 2013]

INT8 Linear Quantization-Aware Training

Neural Network	Floating-Point	Post-Training Quantization		Quantization-Aware Training	
		Asymmetric	Symmetric	Asymmetric	Symmetric
		Per-Tensor	Per-Channel	Per-Tensor	Per-Channel
MobileNetV1	70.9%	0.1%	59.1%	70.0%	70.7%
MobileNetV2	71.9%	0.1%	69.8%	70.9%	71.1%
NASNet-Mobile	74.9%	72.2%	72.1%	73.0%	73.0%

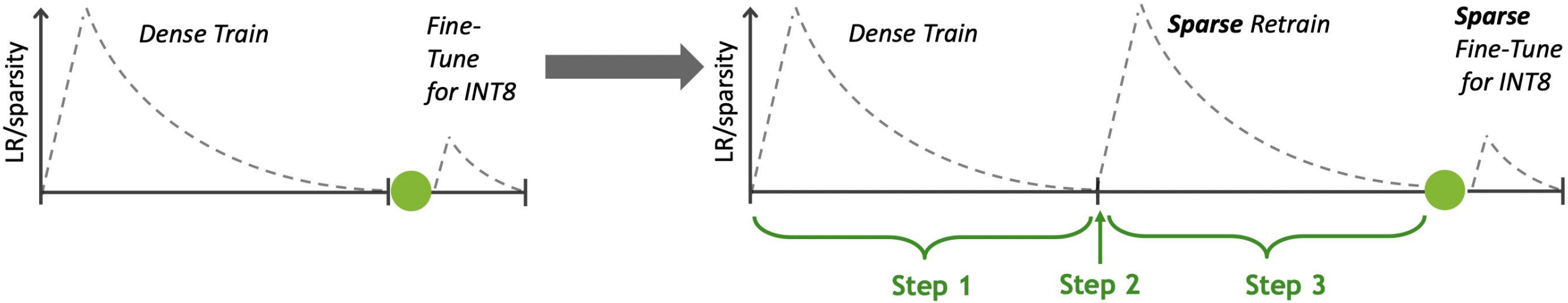
Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper [Raghuraman Krishnamoorthi, arXiv 2018]

[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

SPARSITY AND QUANTIZATION

Quantization Aware Training

Fine-tune for sparsity before fine-tuning for quantization



- **S22075: Integer Quantization for DNN Inference Acceleration**

[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

IMAGE CLASSIFICATION

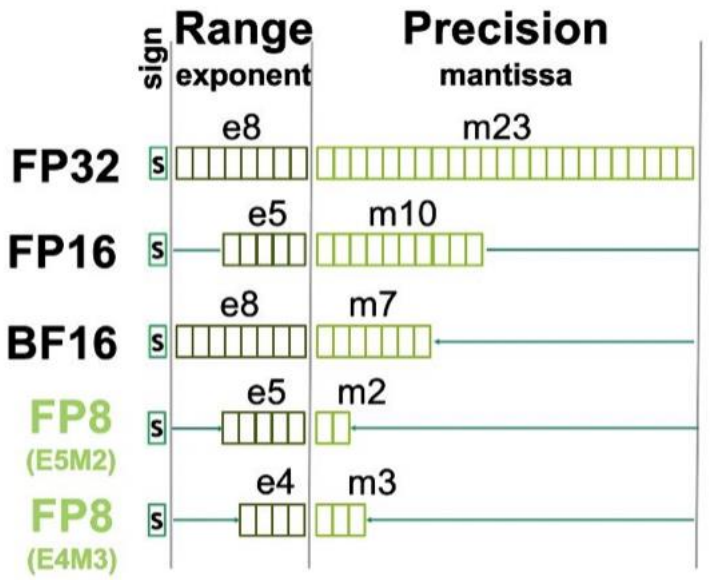
ImageNet

Network	Accuracy				
	Dense FP16	Sparse FP16		Sparse INT8	
ResNet-34	73.7	73.9	0.2	73.7	-
ResNet-50	76.6	76.8	0.2	76.8	0.2
ResNet-101	77.7	78.0	0.3	77.9	-
ResNeXt-50-32x4d	77.6	77.7	0.1	77.7	-
ResNeXt-101-32x16d	79.7	79.9	0.2	79.9	0.2
DenseNet-121	75.5	75.3	-0.2	75.3	-0.2
DenseNet-161	78.8	78.8	-	78.9	0.1
Wide ResNet-50	78.5	78.6	0.1	78.5	-
Wide ResNet-101	78.9	79.2	0.3	79.1	0.2
Inception v3	77.1	77.1	-	77.1	-
Xception	79.2	79.2	-	79.2	-
VGG-16	74.0	74.1	0.1	74.1	0.1
VGG-19	75.0	75.0	-	75.0	-

[https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf]
[https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/]

Precision in Nvidia A100 and H100

- 8-bit floating point (FP8) is new in H100
- Int4 and binary precision is available



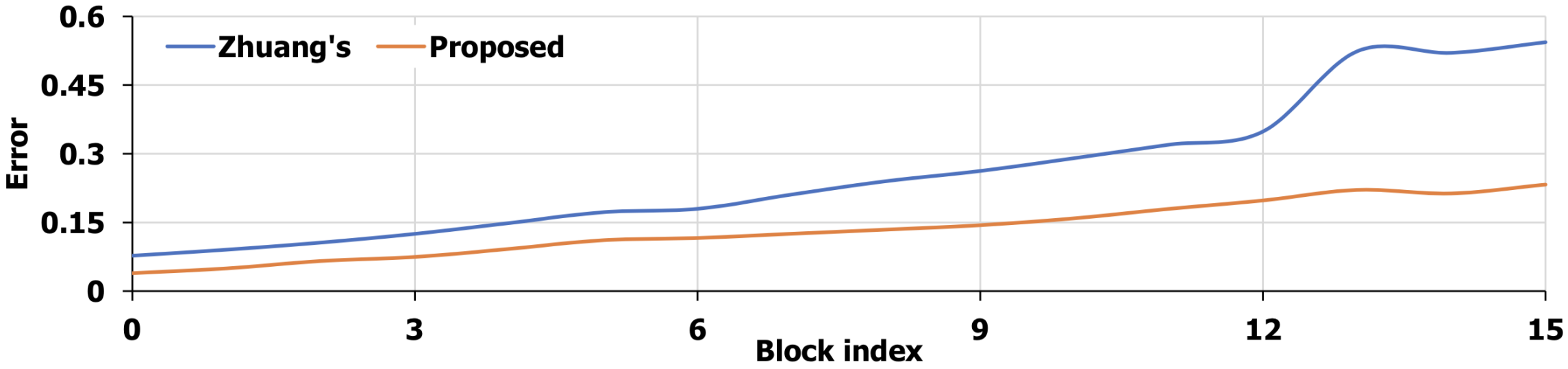
	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x	-	-
	IEEE FP64		19.5	1x	-	-

Quantization: Agenda

- Introduction
  - FP8 model training
  - Quantization-aware training (QAT)
- Quantization interval learning
  - PACT: Parameterized Clipping Activation Function
  - LSQ: Learned Step size Quantization
- Training stabilization and precision learning
  - NIPQ: Noise Injection Pseudo Quantization for Automated DNN Optimization
- Binary neural network
- Concluding remarks

1st Hurdle towards 4-bit and Lower Bit Quantization: Accumulated Quantization Error Problem

- Quantization operation in each layer generates quantization error
- The error is propagated and accumulated across layers
- Quantization error in 4-bit quantization of ResNet-50 blocks
  - Error = Cosine similarity between quantized and full-precision activations

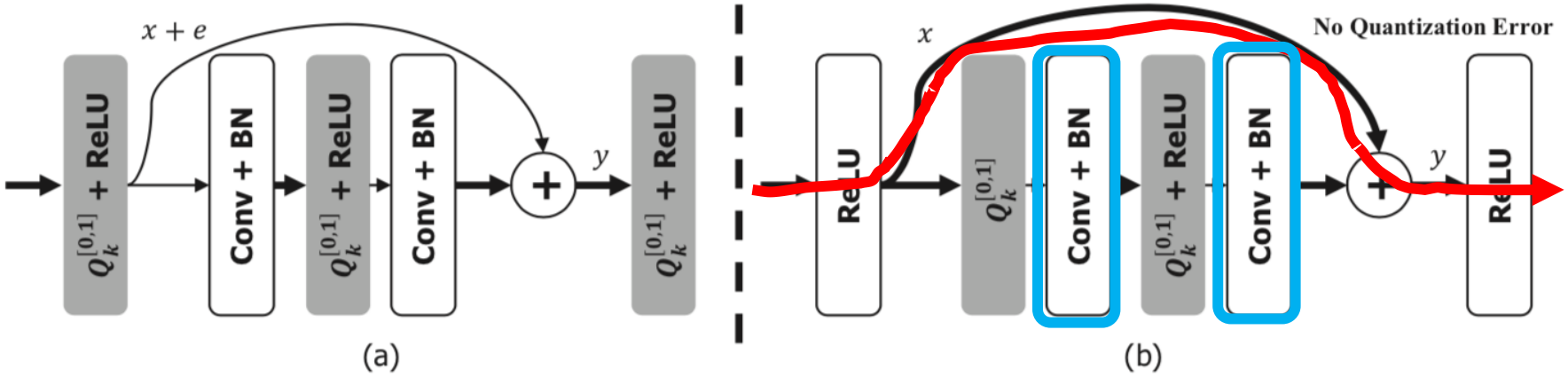


[Park, 2018]

Mixed Precision Network based on Precision Highway

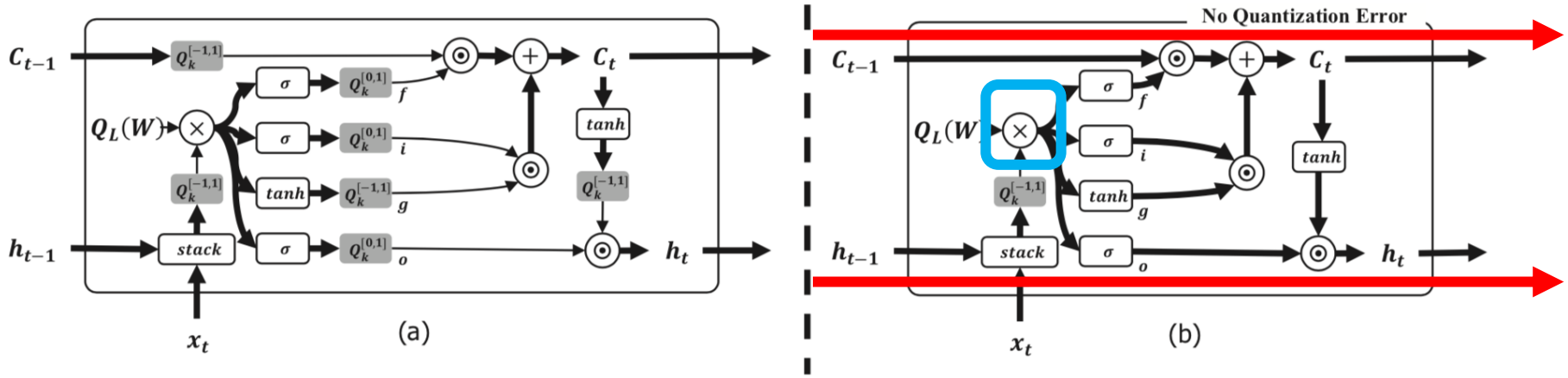
ResNet

High precision (8b) activation



LSTM

Low precision (3b) matrix computation



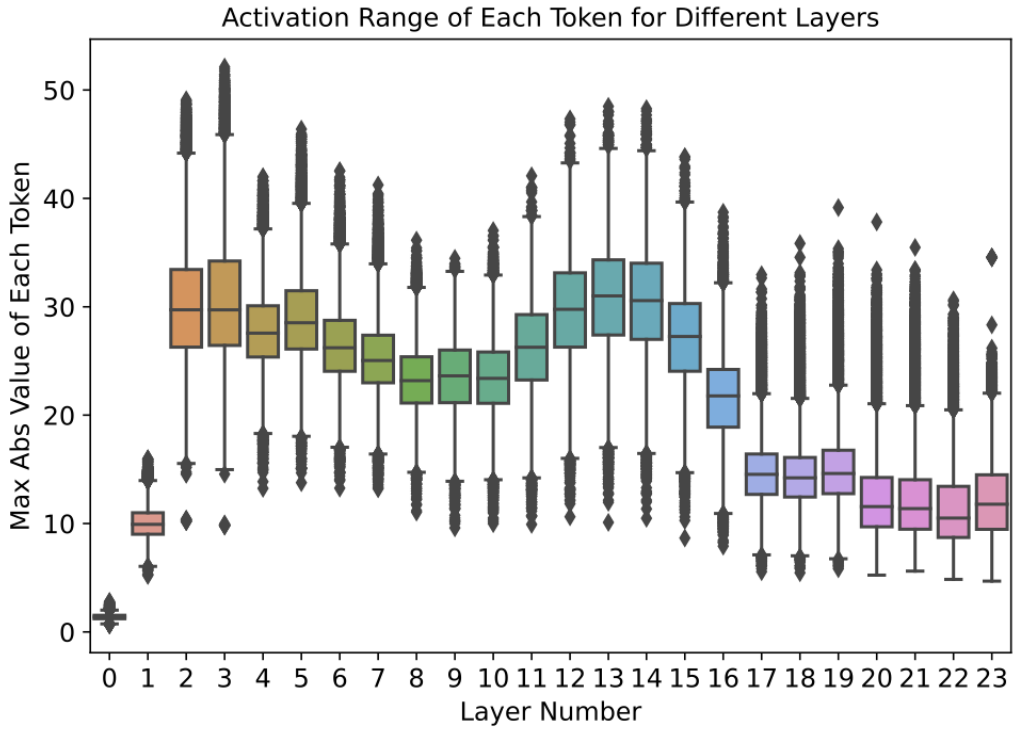
[Park, 2018]

ResNet-50 Results

- 3-bit matrix computation + 8-bit skip connections gives the same accuracy as full precision
- 2-bit computation (8-bit skip) gives only 2.9% top-1 accuracy loss

		Full	8-bit	6-bit
ResNet-18	**4-bit**	71.05 / 90.16	71.07 / 90.20	70.54 / 89.77
	3-bit	70.29 / 89.54	70.08 / 89.51	69.39 / 88.95
	2-bit	66.71 / 87.40	66.62 / 87.33	65.26 / 86.47
ResNet-50	**4-bit**	76.92 / 93.44	76.69 / 93.27	76.25 / 93.13
	3-bit	76.20 / 93.09	**76.08** / 93.03	75.33 / 92.63
	2-bit	73.55 / 91.40	73.15 / 91.34	72.79 / 91.20

Wide Distributions of Activation and Weight. Thus, Each Layer (or Channel) Needs Its Own Quantization Interval



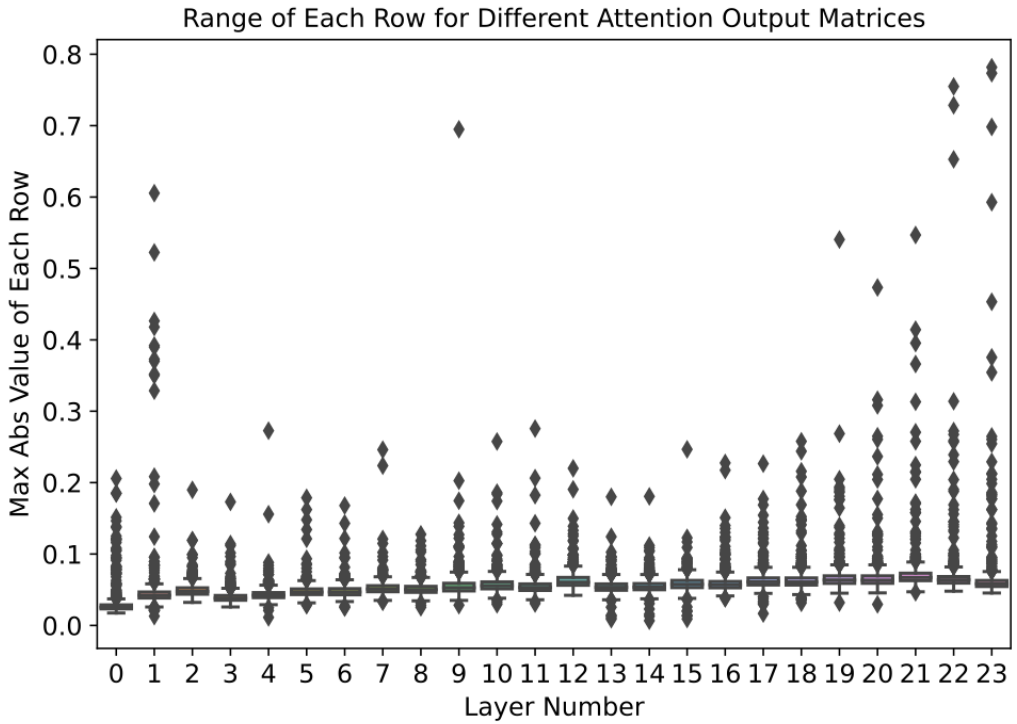
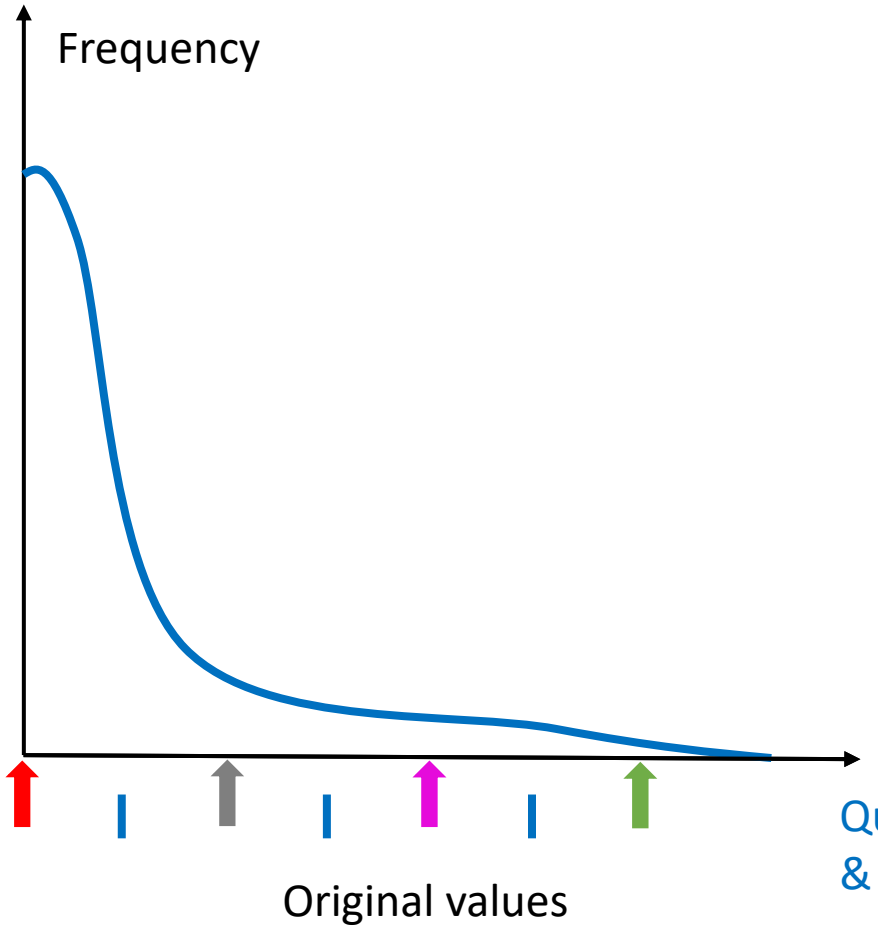
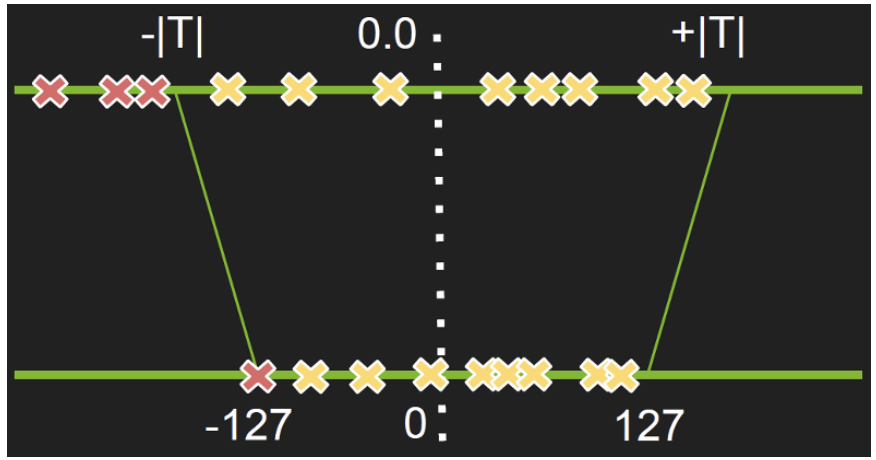


Figure 1: The activation range (left) and row-wise weight range of the attention output matrix (right) of different layers on the pretrained GPT-3_{350M}. See Figure C.1 for the results of $BERT_{base}$.

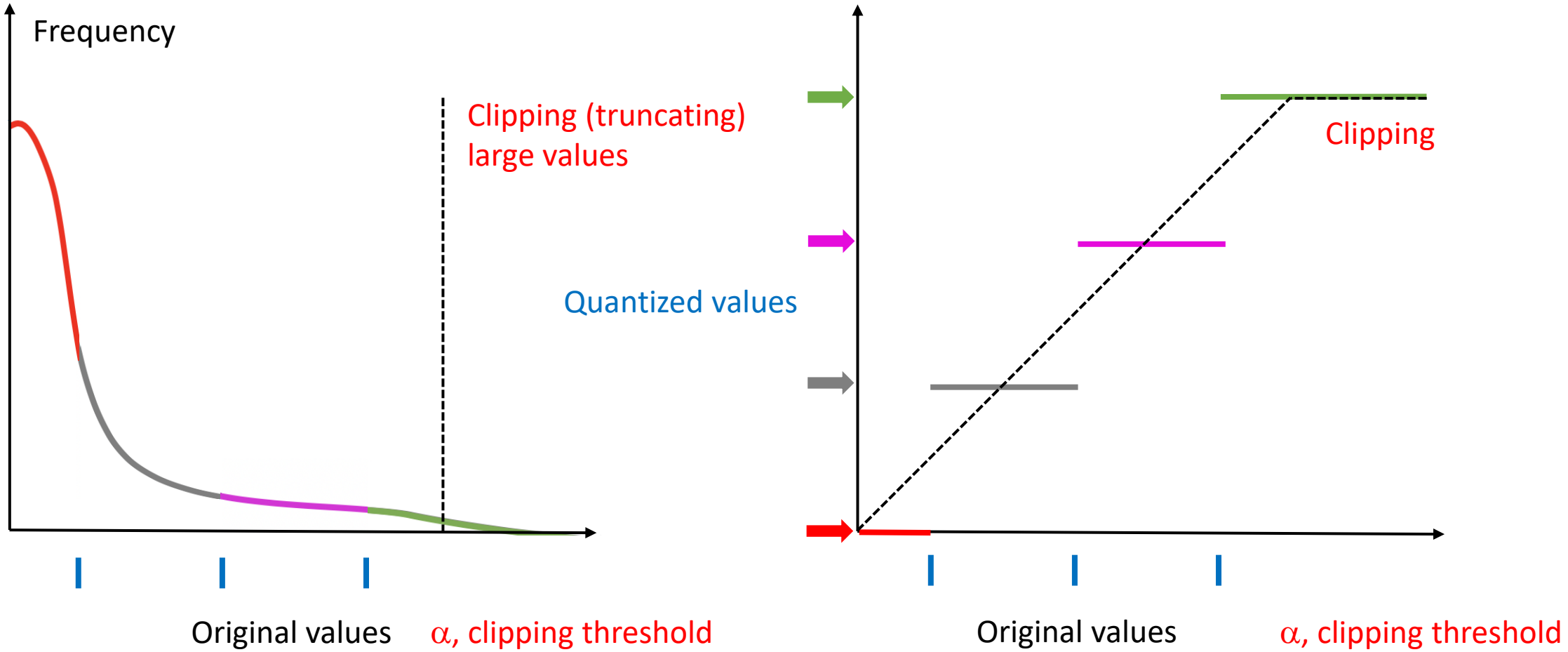
Clipping (Truncating) and Quantizing Activation



- Truncating large magnitude values enables us to reduce quantization error for the remaining data since quant error is proportional to the size of value range under quantization



Clipping (Truncating) and Quantizing Activation



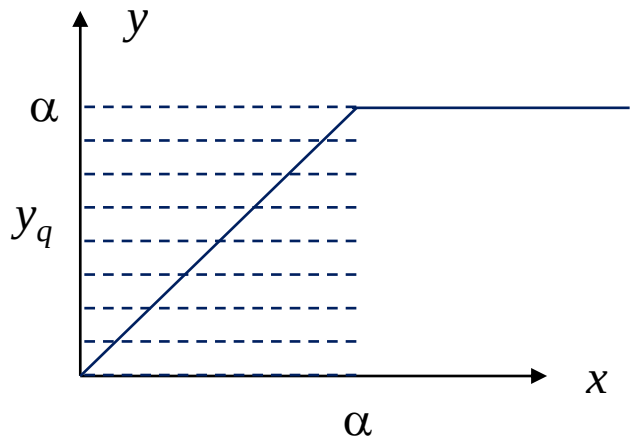
[Choi, 2018]

PACT: Parameterized Clipping Activation Function

- Learning clipping threshold α

Activation Function (bounded ReLU)

$$y = PACT(x) = 0.5(|x| - |x-\alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$



Quantized activation

$$y_q = round(y \cdot \frac{2^k - 1}{\alpha}) \cdot \frac{\alpha}{2^k - 1}$$

We can learn α by training, i.e., SGD!

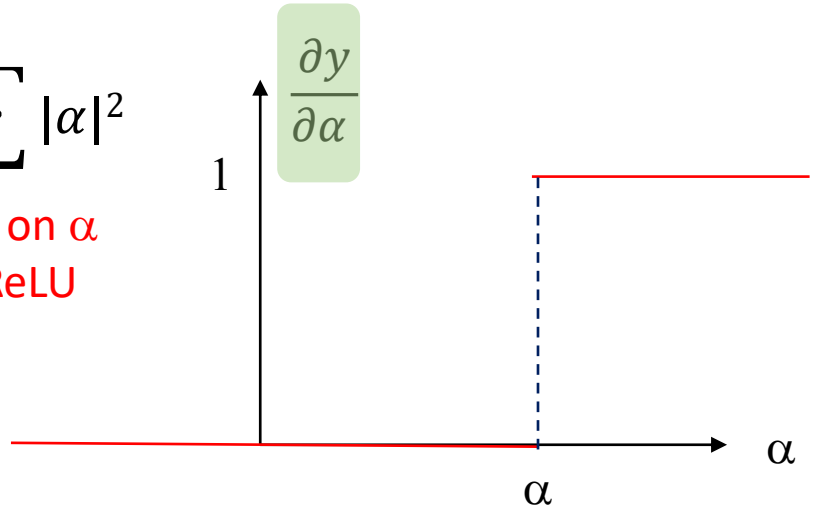
$$\frac{\partial L}{\partial \alpha} = \frac{\partial L}{\partial y_q}\frac{\partial y_q}{\partial \alpha} + \lambda|\alpha|$$

$$L = LCE + \frac{\lambda}{2}\sum |\alpha|^2$$

L2 regularization on α
The larger α → ReLU

$$\frac{\partial y_q}{\partial \alpha} = \frac{\partial y_q}{\partial y}\frac{\partial y}{\partial \alpha} = \begin{cases} 0, & x \in (-\infty, \alpha) \\ 1, & x \in [\alpha, +\infty) \end{cases}$$

=1, called straight through estimator (STE)

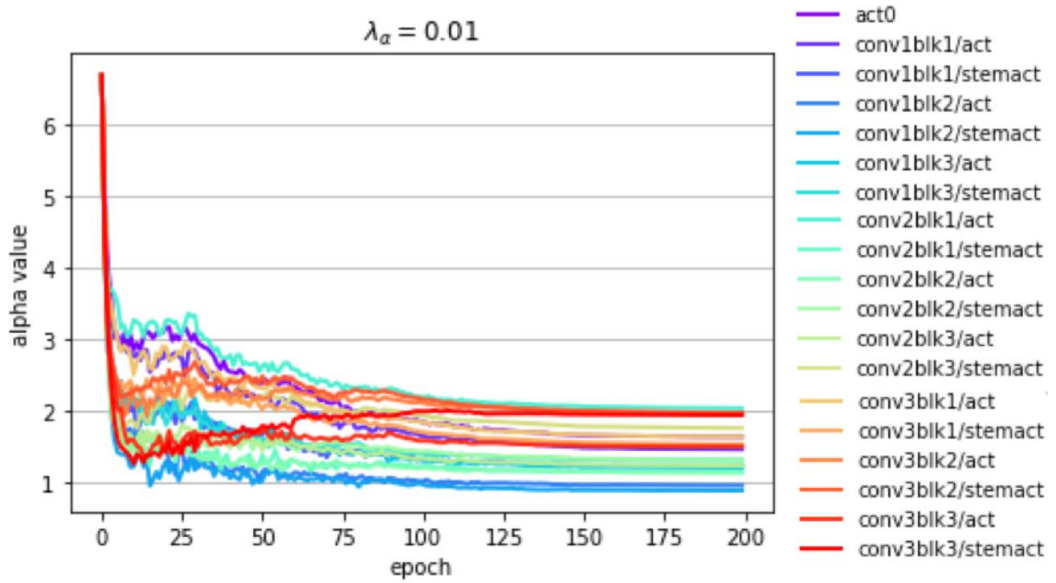


[Choi, 2018]

PACT Results: Learning Clipping Threshold

- Weight quantization based on statistics
- Promising in 4 bit activation

Network	FullPrec	DoReFa				PACT			
		2b	3b	4b	5b	2b	3b	4b	5b
CIFAR10	0.916	0.882	0.899	0.905	0.904	0.897	0.911	0.913	0.917
SVHN	0.978	0.976	0.976	0.975	0.975	0.977	0.978	0.978	0.979
AlexNet	0.551	0.536	0.550	0.549	0.549	0.550	0.556	0.557	0.557
ResNet18	0.702	0.626	0.675	0.681	0.684	0.644	0.681	0.692	0.698
ResNet50	0.769	0.671	0.699	0.714	0.714	0.722	0.753	0.765	0.767



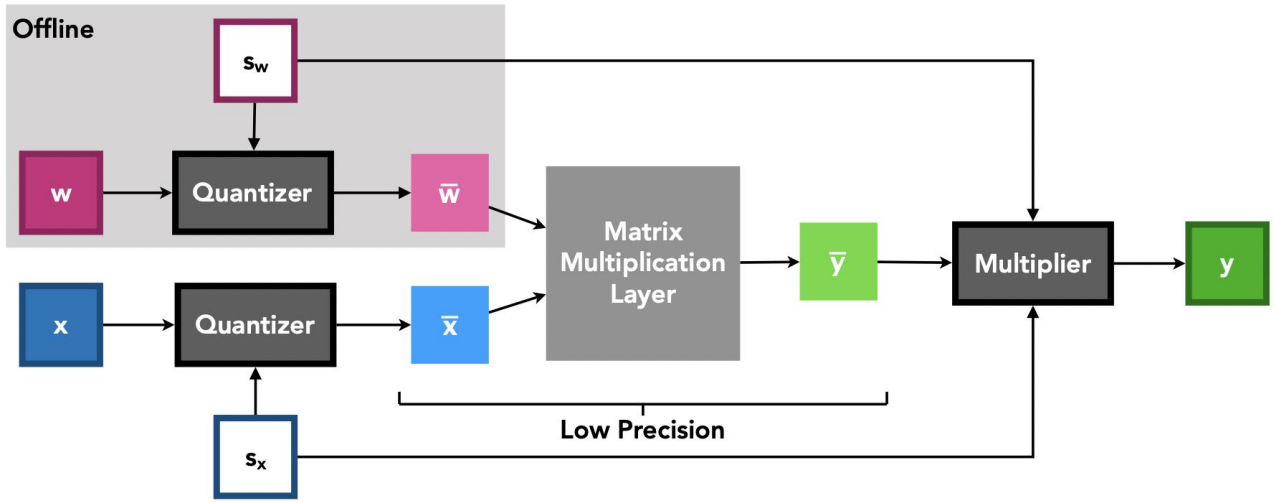
- Limitation
  - MobileNets?

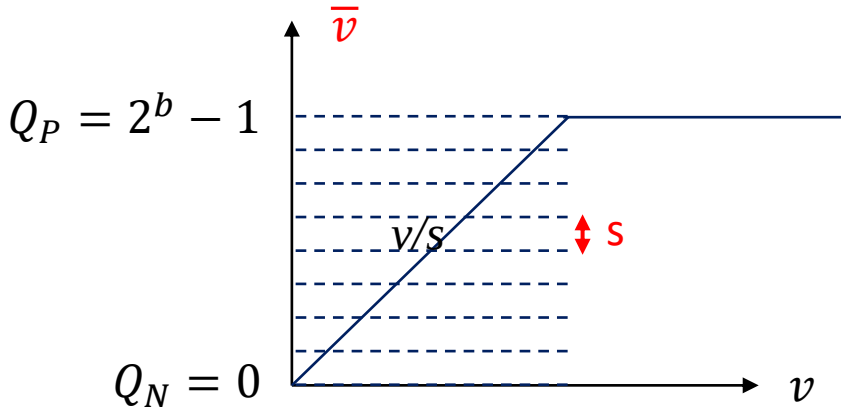
[Esser, 2020]

LSQ: Learned Step size Quantization

- Original value v, step size s, quantized integer $\bar{v}$, quantized real value $\hat{v}$

$$\bar{v} = \lfloor clip(v/s, -Q_N, Q_P) \rceil = \begin{cases} round(\frac{v}{s}) & \text{if } -Q_N < v/s < Q_P \\ -Q_N & \text{if } v/s \leq -Q_N \\ Q_P & \text{if } v/s \geq Q_P \end{cases}$$





[Esser, 2020]

LSQ: Learned Step size Quantization

- Original value v, step size s, quantized integer $\bar{v}$, quantized real value $\hat{v}$

$$\bar{v} = \lfloor clip(v/s, -Q_N, Q_P) \rceil = \begin{cases} r(\frac{v}{s}) & \text{if } -Q_N < v/s < Q_P \\ -Q_N & \text{if } v/s \le -Q_N \\ Q_P & \text{if } v/s \ge Q_P \end{cases}$$

$$\hat{v} = \bar{v} \times s$$

$$\frac{\partial L}{\partial s} = \frac{\partial L}{\partial \dots} \cdots \frac{\partial \hat{v}}{\partial s}$$

$$\frac{\partial \hat{v}}{\partial s} = \frac{\partial r(\frac{v}{s})}{\partial s} s + r(\frac{v}{s}) = \frac{\partial r()}{\partial s} \frac{\partial \frac{v}{s}}{\partial s} s + r(\frac{v}{s}) \quad \text{if } -Q_N < v/s < Q_P$$

=1 (STE)

$= -\frac{v}{s^2}$

$$\frac{\partial \hat{v}}{\partial s} = \begin{cases} -v/s + \lfloor v/s \rceil & \text{if } -Q_N < v/s < Q_P \\ -Q_N & \text{if } v/s \le -Q_N \\ Q_P & \text{if } v/s \ge Q_P \end{cases}$$

[Esser, 2020]

PACT and LSQ Have Different Error Backprop Characteristics

- LSQ has a wider window of error backprop than PACT
  - Thus, it may offer better accuracy in 4~2bits

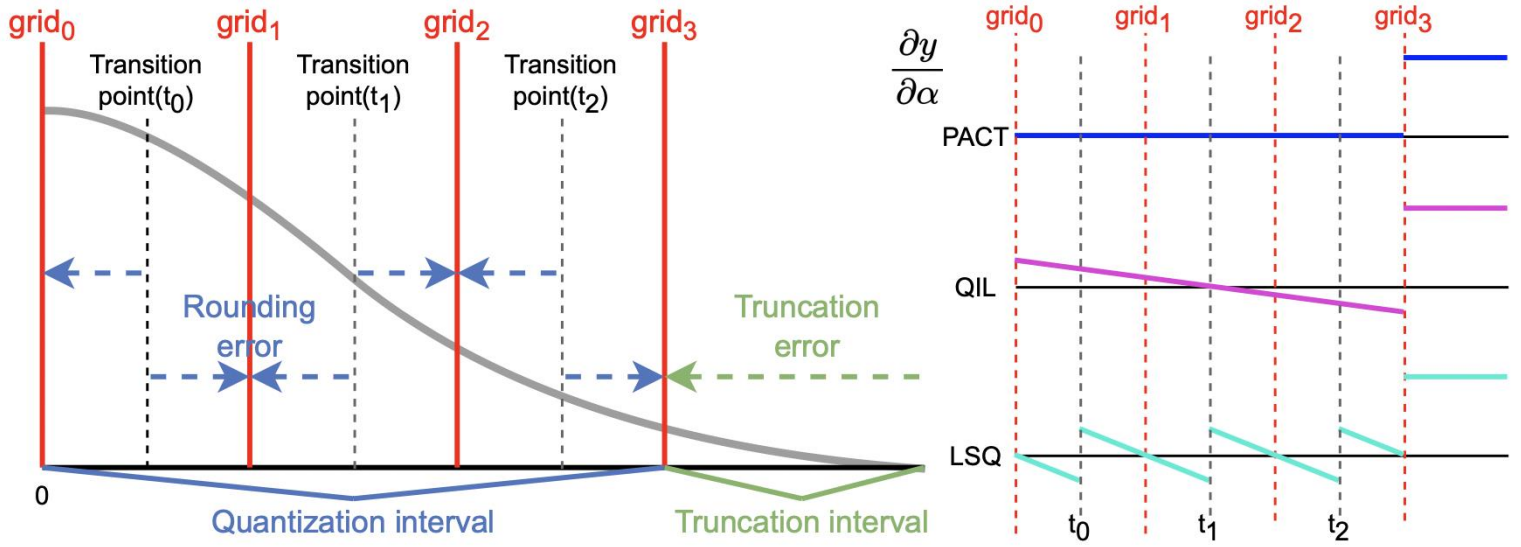
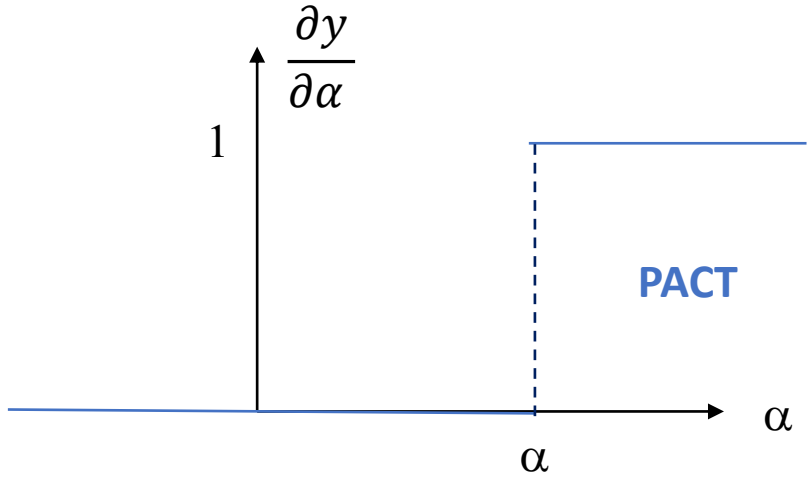


Fig. 1: Error components of quantization for activation distribution (left) and gradient of clipping threshold (right)



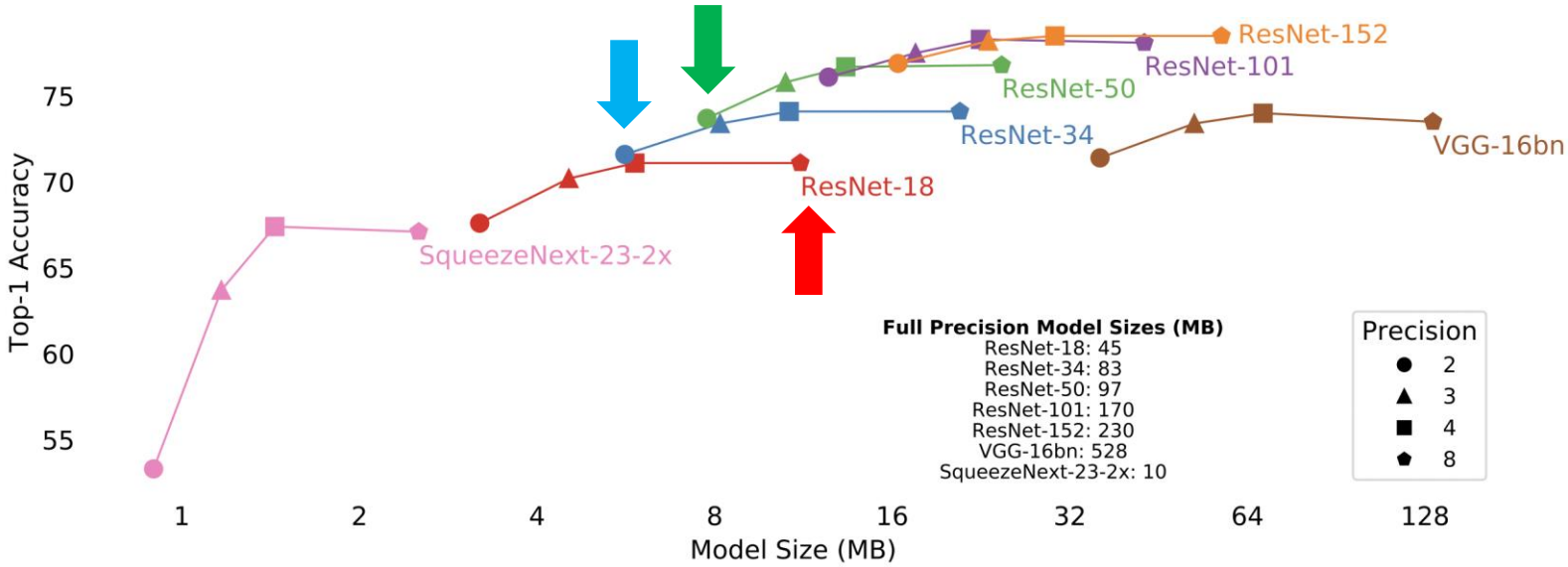
$$\frac{\partial \hat{v}}{\partial s} = \begin{cases} -v/s + \lfloor v/s \rceil & \text{if } -Q_N < v/s < Q_P \\ -Q_N & \text{if } v/s \leq -Q_N \\ Q_P & \text{if } v/s \geq Q_P \end{cases}$$

LSQ

[Esser, 2020]

LSQ: Experimental Results

- 2-bit ResNet-34 (50) network offers better accuracy than a smaller (# params) full precision network, e.g., ResNet-18
- 3-bit ResNet-50 offers the same top-1 accuracy as 32-bit model
- Limitation
  - MobileNets?



Network	Top-1 Accuracy @ Precision 2	3	4	8	*32*	Top-5 Accuracy @ Precision 2	3	4	8	*32*
ResNet-18	67.9	70.6	71.2	71.1	*70.5*	88.1	89.7	90.1	90.1	*89.6*
ResNet-34	72.4	74.3	74.8	74.1	*74.1*	90.8	91.8	92.1	91.7	*91.8*
ResNet-50	74.6	76.9	77.6	76.8	*76.9*	92.1	93.4	93.7	93.3	*93.4*

Quantization: Agenda

- Introduction
  - FP8 model training
  - Quantization-aware training (QAT)
- Quantization interval learning
  - PACT: Parameterized Clipping Activation Function
  - LSQ: Learned Step size Quantization
- Training stabilization and precision learning
  - NIPQ: Noise Injection Pseudo Quantization for Automated DNN Optimization
- Binary neural network
- Concluding remarks

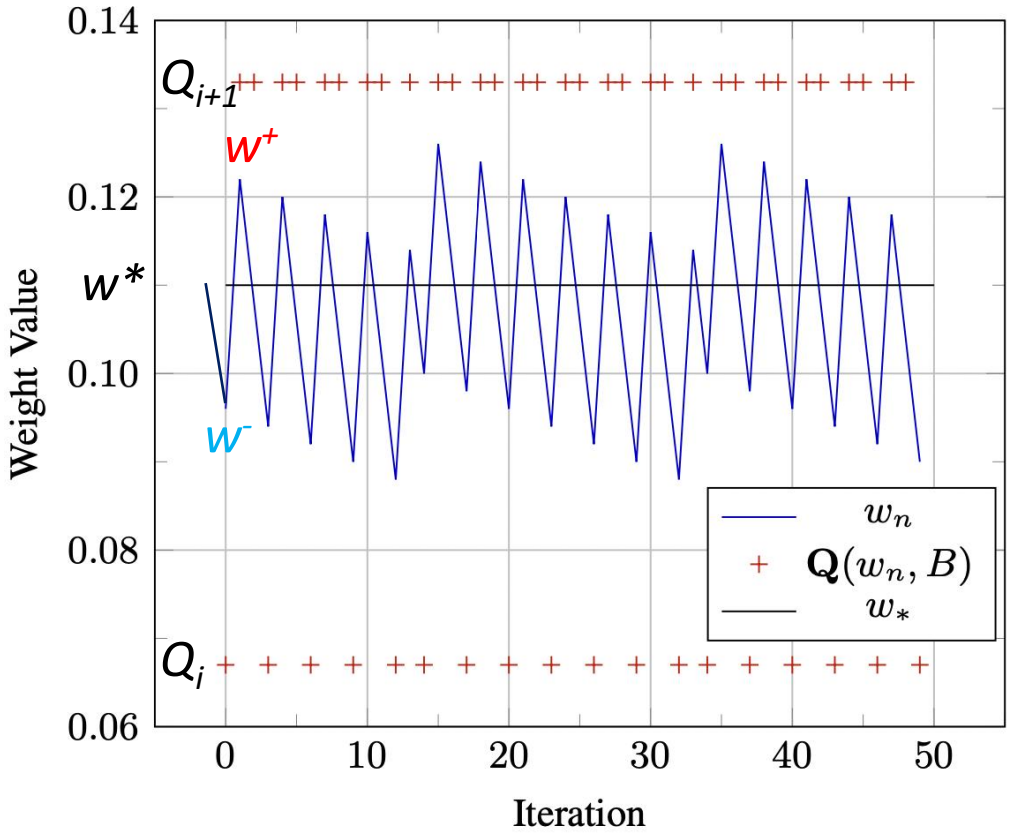
[Defossez, 2021]

Gradient Instability due to Oscillations of Quantized Values Makes Training Unstable

- Sup… optimal weight w* in full precision
- …
- … to w^-
- …
- … w^+

There are numerous (e.g., millions to billions of) parameters whose gradients can flip in subsequent training iterations, which can make training unstable.

In addition, such parameters will converge towards the quantization threshold due to decreasing learning rate



[Park, 2022]

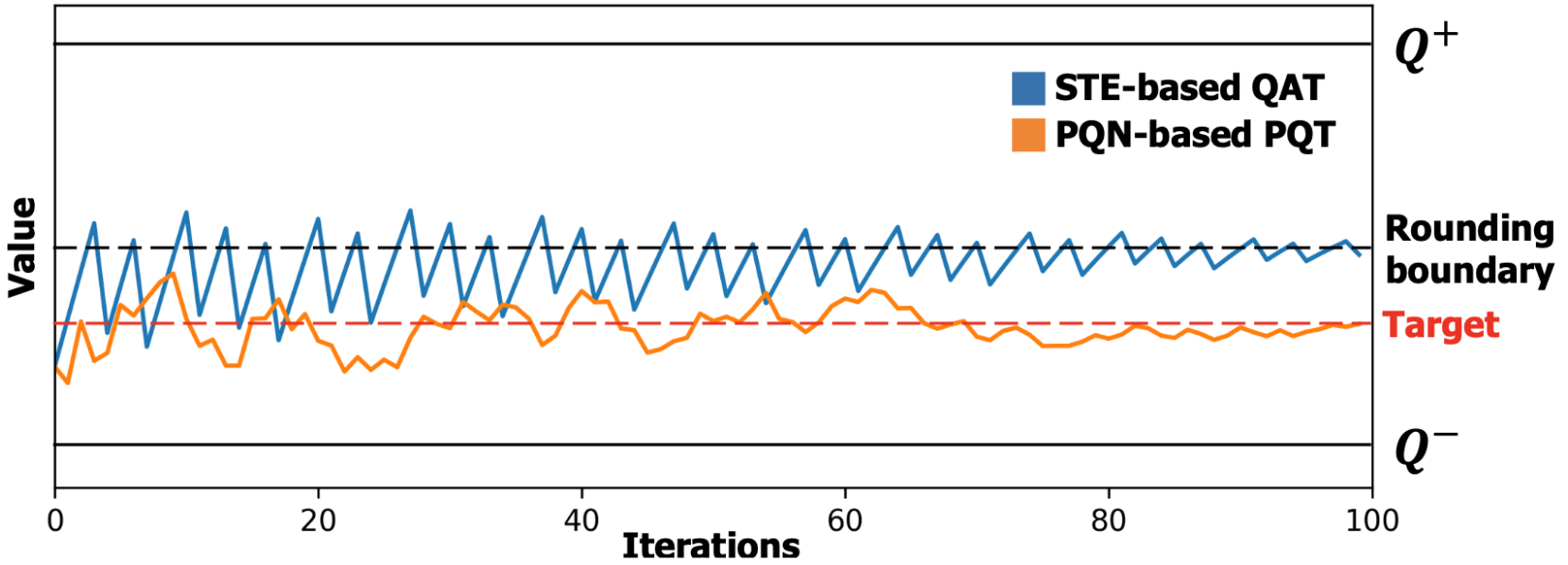
Simple Example

$$\mathcal{L}^{Q}_{exp}(x, \alpha, b) = \sum_{i=0}^{N-1} \Big(t_i - Q(x_i|\alpha, b) \Big)^2$$

$$x \in \mathbb{R}^N$$

$$Q(x|\alpha, b) = \begin{cases} 0, & x \leq 0 \\ \lfloor x/\Delta \rceil \cdot \Delta, & 0 < x < \alpha \\ \alpha, & x \geq \alpha, \end{cases}$$

In Quantization-aware Training (QAT)
the weight can converge
near the quantization boundary



[Defossez, 2021]

DiffQ: Differentiable Model Compression via Pseudo Quantization Noise

- Rounding incurs quantization error $[-\Delta/2, \Delta/2]$ $\Delta = \frac{Max - Min}{2^B - 1}$

$$\forall w \in [0,1], B \in \mathbb{N}_*, \mathbf{Q}(w,B) = \frac{\text{round}\left(w \cdot (2^B - 1)\right)}{2^B - 1}$$

- Mimicking quantization error injection makes # bits trainable
  - In forward pass, pseudo quantization noise is injected to activation or weight

$$\tilde{\mathbf{Q}}(x,B) = x + \frac{\Delta}{2} \cdot \mathcal{U}[-1,1]$$

  - # bits, B is trainable. Thus, # of bits for each weight or activation can be learned

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial \dots} \dots \frac{\partial \tilde{Q}}{\partial B} \qquad \frac{\partial \tilde{Q}}{\partial B} = \frac{\partial \tilde{Q}}{\partial \Delta} \frac{\partial \Delta}{\partial B} \qquad \Delta = \frac{Max - Min}{2^B - 1}$$

[Defossez, 2021]

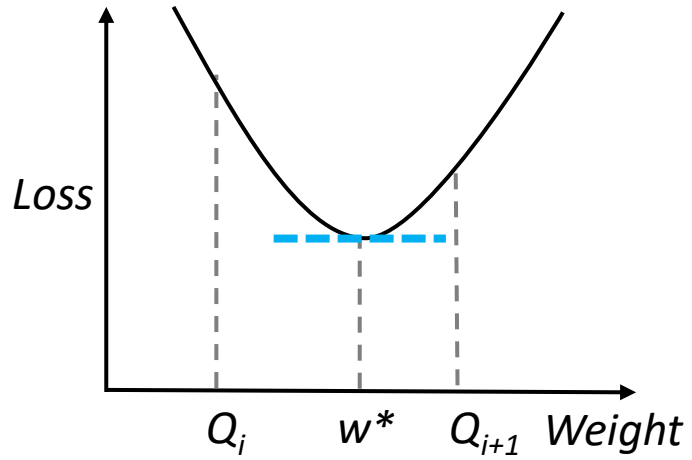
DiffQ: No Oscillations and No STE. Thus, Training Becomes Stable

- We utilize floating point values (of weights and activations) while injecting pseudo noise which emulates quantization error
- Thus, since quantized values are not used, there are no oscillations due to quantization, which stabilizes training
- $\tilde{Q}$ is a floating point value consisting of the original value x and injected pseudo noise

$$\tilde{\mathbf{Q}}(x, B) = x + \frac{\Delta}{2} \cdot \mathcal{U}[-1, 1]$$

- Only floating point values are used = no STE is used

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial \dots} \cdots \frac{\partial \tilde{Q}}{\partial B} \qquad \frac{\partial \tilde{Q}}{\partial B} = \frac{\partial \tilde{Q}}{\partial \Delta} \frac{\partial \Delta}{\partial B} \qquad \Delta = \frac{Max - Min}{2^B - 1}$$



[Defossez, 2021]

DiffQ: Results

- Training loss (e.g., for classification) = CE loss + model size
  - Typically, per-layer B, i.e., a single $B_w(i)$ is assigned to all the weights of layer i
  - Total model size, M(b) = $\sum_{i=1}^{\#\ layers} B_w(i) * N_w(i)$

CE loss of quantized model

Model size # weight bits

$$\min_{w,b} L(f_{\tilde{\mathbf{Q}}(w,b)}) + \lambda(m)\mathbf{M}(b)$$

Smaller λ gives higher accuracy and larger model

Larger λ gives smaller model and lower accuracy

Table 4: Image classification results for the ImageNet benchmark. Results are presented for DIFFQ and QAT using 4 and 8 bits using the DeiT model (Touvron et al. 2020). We report Top-1 Accuracy (Acc.) together with Model Size (M.S.).

	TOP-1 ACC. (%) ↑	M.S. (MB) ↓
UNCOMPRESSED	81.8	371.4
QAT 4BITS	79.2	41.7
QAT 8BITS	81.6	82.9
DIFFQ (λ=1e−2)	**82.0**	45.7
DIFFQ (λ=0.1)	81.5	**33.02**

[Park, 2022]

NIPQ: Noise Injection Pseudo Quantization for Automated DNN Optimization

- Goal: automatic per-layer quantization for mixed precision model
- Idea: **DiffQ** + **PACT**
  - Determine per-layer weight/activation precision # bits, B (in DiffQ) and the clipping threshold (or range) α (in PACT)
  - Only floating values are used, which enables stable training

Learning # bits, B in DiffQ

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial \dots} \cdots \frac{\partial \tilde{Q}}{\partial B} \qquad \tilde{\mathbf{Q}}(x, B) = x + \frac{\Delta}{2} \cdot \mathcal{U}[-1, 1]$$

Trainable!

$$\frac{\partial \tilde{Q}}{\partial B} = \frac{\partial \tilde{Q}}{\partial \Delta}\frac{\partial \Delta}{\partial B} \qquad \frac{\partial \tilde{Q}}{\partial \Delta} = \frac{1}{2}\mathcal{U}[-1,1] \qquad \frac{\partial \Delta}{\partial B} = \frac{\partial}{\partial B}\frac{Max - Min}{2^B - 1}$$

Learning range α in PACT

We can learn α by training, i.e., SGD!

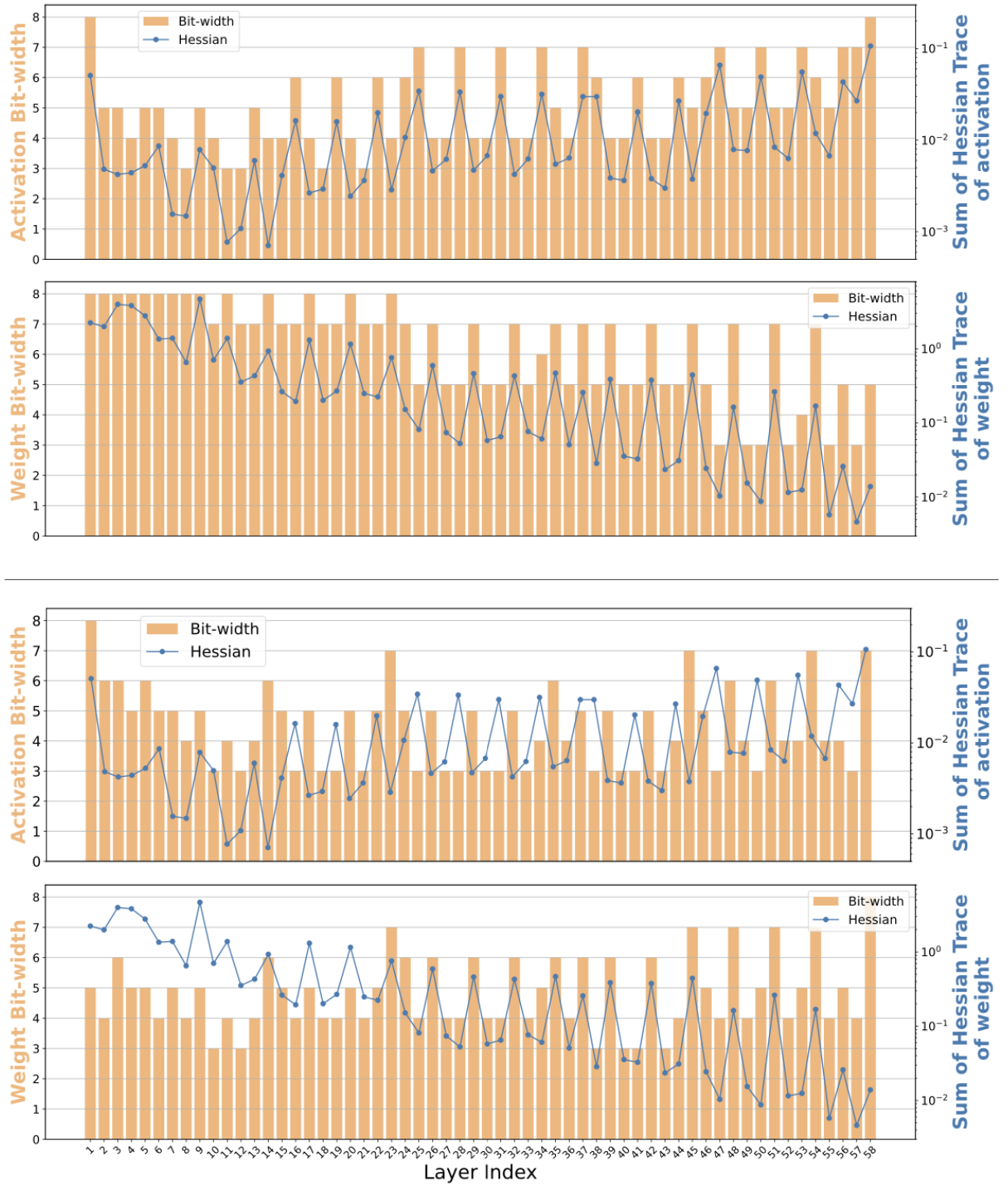
$$\frac{\partial L}{\partial \alpha} = \frac{\partial L}{\partial y_q}\frac{\partial y_q}{\partial \alpha} + \lambda|\alpha|$$

$$y_q = \begin{cases} 0 & if\ x < 0, \\ \alpha & if\ x > \alpha, \\ \tilde{\mathrm{Q}} & otherwise. \end{cases} \qquad \frac{\partial y_q}{\partial \alpha} = \begin{cases} 0, & x \in (-\infty, \alpha) \\ 1, & x \in [\alpha, +\infty) \end{cases}$$

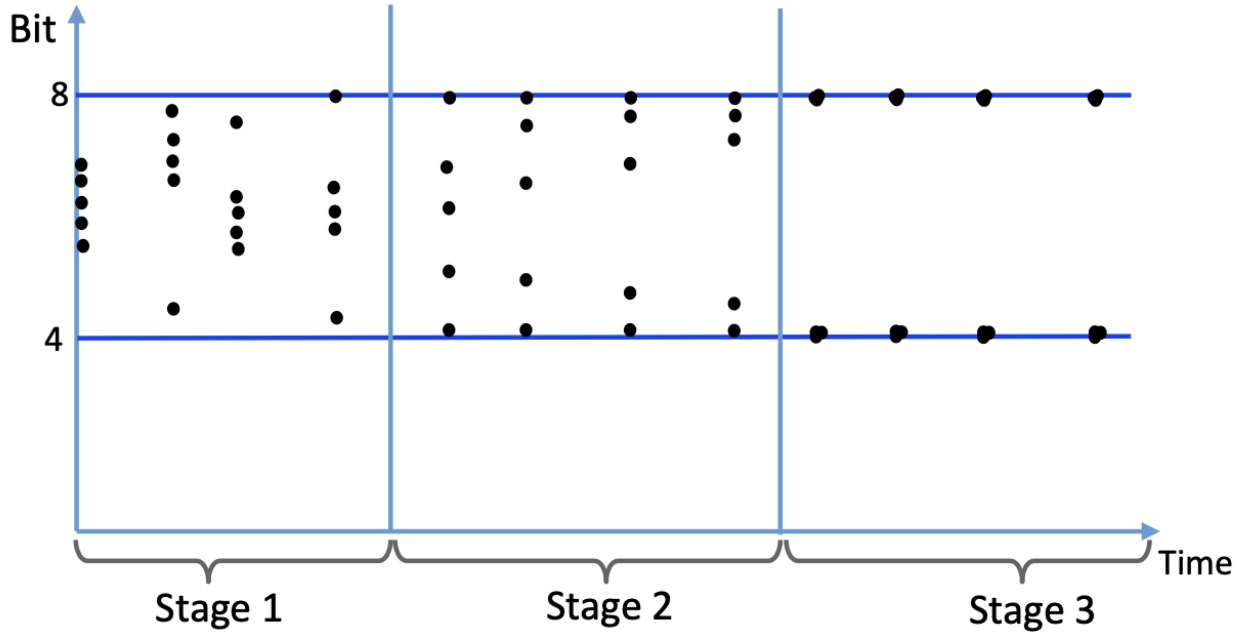
[Park, 2022]

Results: Automatic Mixed Precision Model

- MobileNet-v2, CIFAR-100 dataset
- For average 4 bit weight and activation (up)
  - Bit-width and Hessian align with each other
- For average 1.5 BOPS (down)
  - Weak alignment of bit-width and Hessian to meet the constraint of computation cost

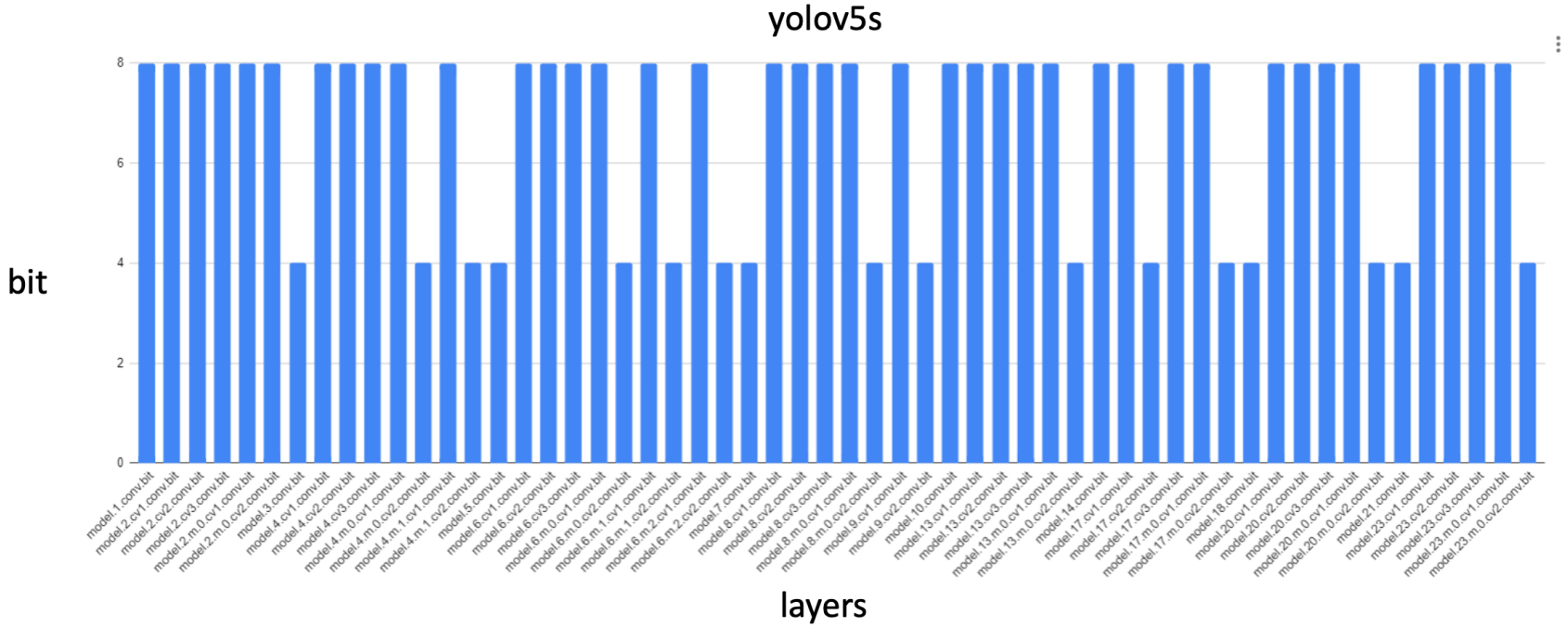


- NIPQ style layerwise bitwidth training: 4-8 Mixed-precision
  - Overall process consists of 3 stages
    - **The black dots in the figure indicate the update of layer-wise bit-width**
    1. Optimize avg-bits toward specific target-bit
       - The layer-wise bit is selected from {4, 5, 6, 7, 8}
    2. Pushing each bits to the possible candidates (i.e. {4, 8})
    3. Fix all bits to {4, 8} and train with real quantization, not pseudo-quantization noise



8/4bit Mixed Precision Models

- EfficientNet on ImageNet top 1 accuracy
  - FP32 model = 75.876%, average 6.29 bit model = 75.718%
- Yolov5s on PASCAL VOC mAP
  - FP32 model = 0.856, W8/A8 model = 0.837, average 5.8 bit model = 0.833

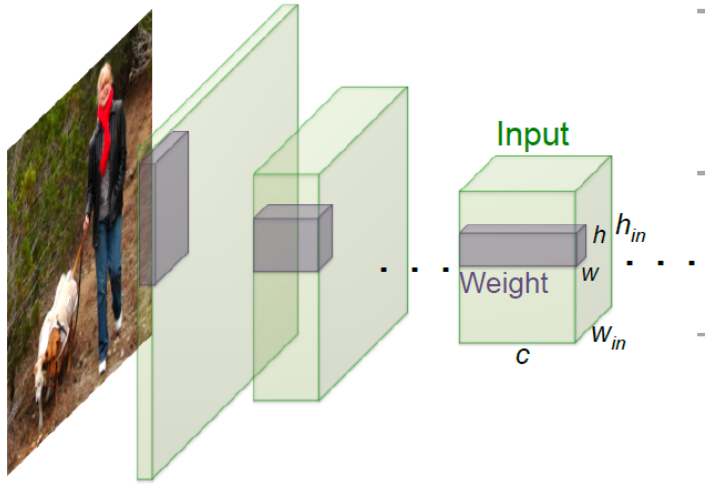


Quantization: Agenda

- Introduction
  - FP8 model training
  - Quantization-aware training (QAT)
- Quantization interval learning
  - PACT: Parameterized Clipping Activation Function
  - LSQ: Learned Step size Quantization
- Training stabilization and precision learning
  - NIPQ: Noise Injection Pseudo Quantization for Automated DNN Optimization
- Binary neural network
- Concluding remarks

Binary Network

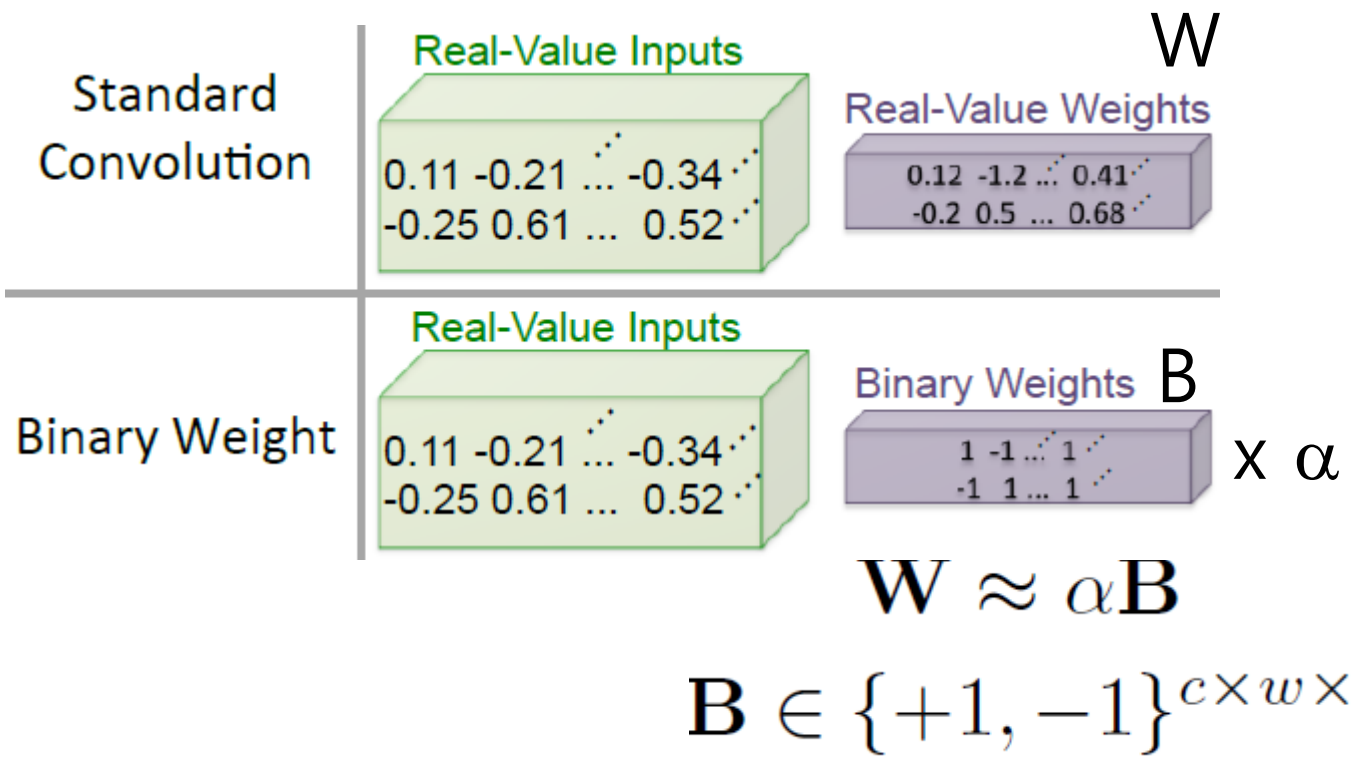
- Two types of network
  - Binary-Weight network: only weights are binary, activations are 32-bit floating point values. It does not hurt accuracy while reducing weight size significantly
  - XNOR-Net: both weight and activation are binary



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Binary-Weight-Network

we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$.



$$\mathbf{W} \approx \alpha \mathbf{B}$$

$$\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$$

$$\mathbf{I} * \mathbf{W} \approx (\mathbf{I} \oplus \mathbf{B})\,\alpha$$

$\oplus$ indicates a convolution without any multiplication.

For *k*-th filter (output feature map) on *l*-th layer

$$\alpha = \mathcal{A}_{lk} \quad \mathcal{W}_{lk} \approx \mathcal{A}_{lk}\mathcal{B}_{lk}$$

How to Obtain **B** and α ?

we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$.

Solve this

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$

$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

$$\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$$

$$J(\mathbf{B}, \alpha) = \alpha^2 \mathbf{B}^\mathsf{T}\mathbf{B} - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{W}^\mathsf{T}\mathbf{W}$$

$\mathbf{B}^\mathsf{T}\mathbf{B} = n$ is a constant

$\mathbf{W}^\mathsf{T}\mathbf{W}$ is also a constant c

$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{c}$$

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}}\{\mathbf{W}^\mathsf{T}\mathbf{B}\} \quad s.t. \ \mathbf{B} \in \{+1, -1\}^n$$

How to Obtain **B** and α ?

we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$.

$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha \mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{c}$$

Solve this

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$

$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}}\{\mathbf{W}^\mathsf{T}\mathbf{B}\} \quad s.t. \ \mathbf{B} \in \{+1, -1\}^n$$

$$\mathbf{B}^* = \operatorname{sign}(\mathbf{W})$$

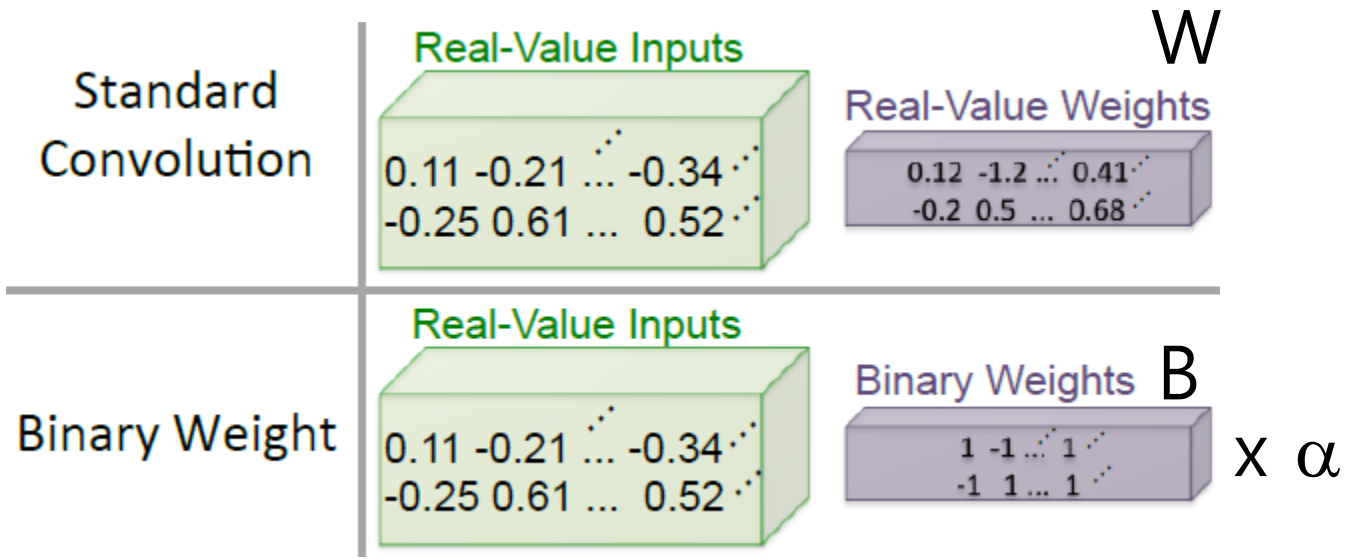
$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha \mathbf{W}^\mathsf{T}\mathbf{B}^* + \mathbf{c}$$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T}\mathbf{B}^*}{n}$$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T} \operatorname{sign}(\mathbf{W})}{n} = \frac{\sum |\mathbf{W}_i|}{n} = \frac{1}{n}\|\mathbf{W}\|_{\ell 1}$$

Binary-Weight-Network

we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$.



$$\mathbf{W} \approx \alpha \mathbf{B}$$

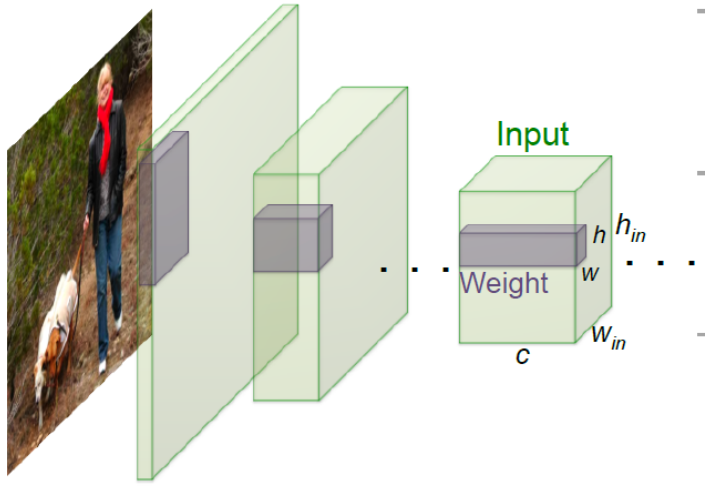
$$\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$$

$$\mathbf{B}^* = \text{sign}(\mathbf{W})$$

$$\alpha^* = \frac{\mathbf{W}^\intercal \text{sign}(\mathbf{W})}{n} = \frac{\sum |\mathbf{W}_i|}{n} = \frac{1}{n} \|\mathbf{W}\|_{\ell 1}$$

Binary Network

- Two types of network
  - Binary-Weight network: only weights are binary, activations are 32-bit floating point values. It does not hurt accuracy while reducing weight size significantly
  - XNOR-Net: both weight and activation are binary

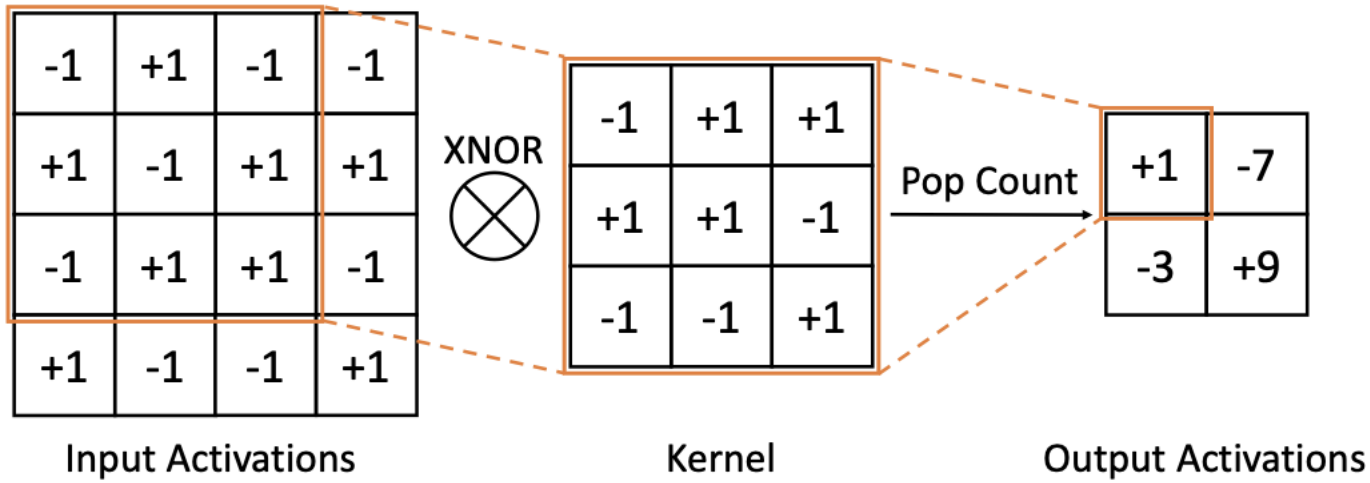


	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

XNOR-Network (Binary Weight and Binary Activation Network)

- Why XNOR?
  - { 1, -1, 1, 1} dot_prod { -1, -1, 1, -1} = 1*(-1)+(-1)*(-1)+1*1+1*(-1) =-1+1+1-1=0
  - XNOR operation

A	B	A XNOR B (1/0 vs 1/-1)
0 (-1)	0 (-1)	1 (=(-1)*(-1)=1)
0 (-1)	1 (1)	0 (=(-1)*1=-1)
1 (1)	0 (-1)	0 (=1*(-1)=-1)
1 (1)	1 (1)	1 (=1*1=1)

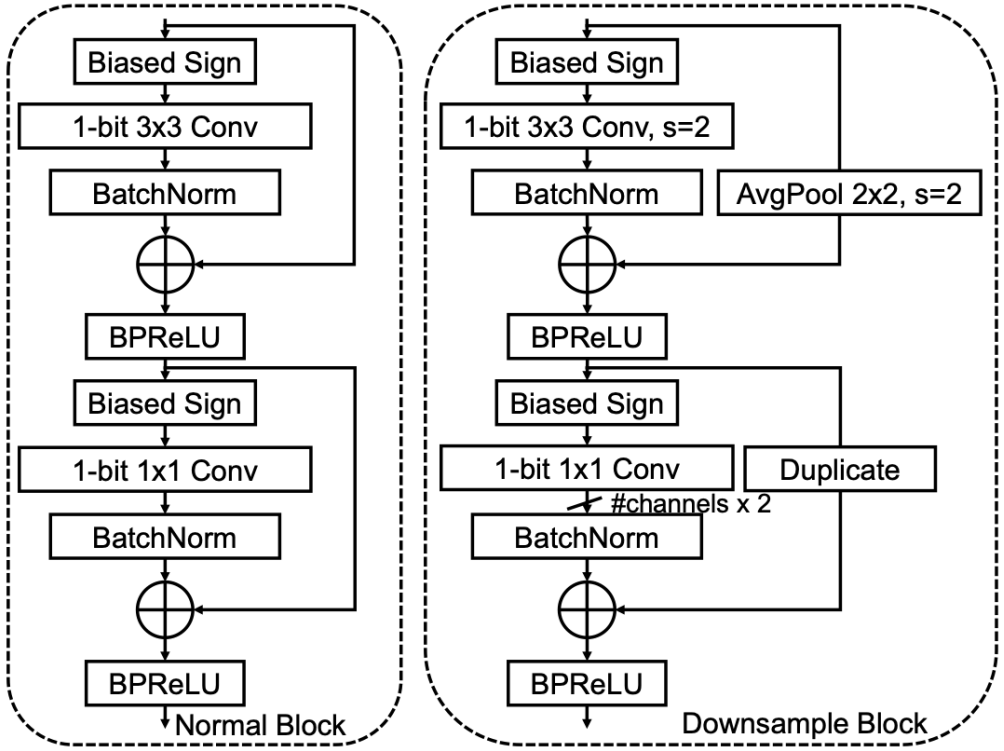


- Binary value multiplications + accumulation + binary output → XNOR operations + bit counts + take the sign of addition result (# 1's - # (-1)'s)

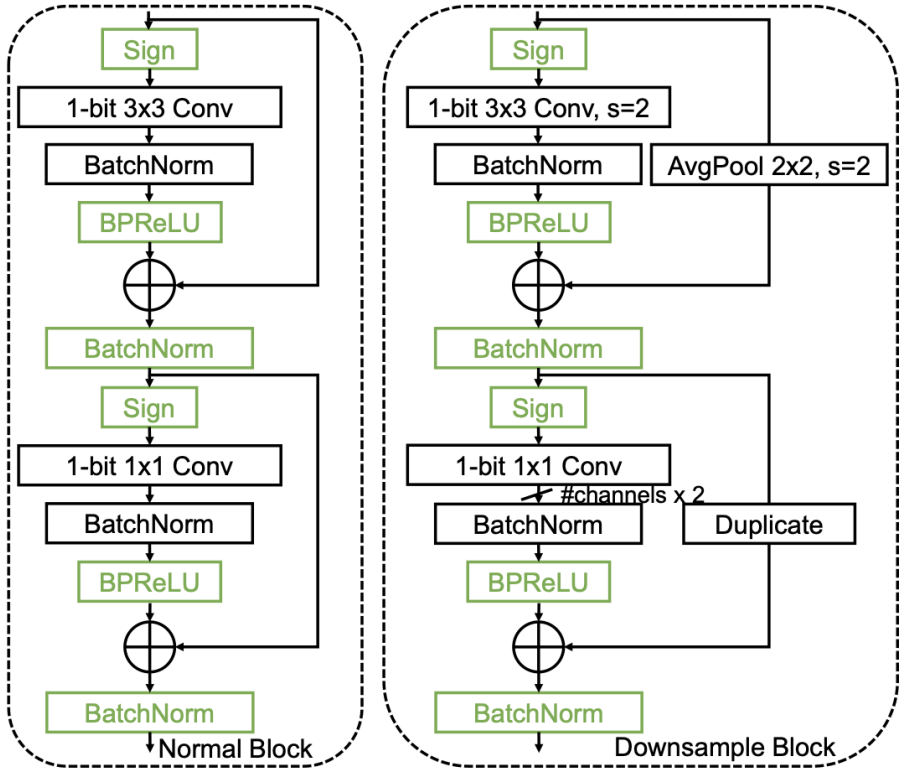
[Zhang20]

There is NO General Binarization Method yet!

- Special network structures are being proposed for 1 bit and 2 bits



ReActNet



FracBNN

[Zhang20]

FracBNN's 1bit Weight+2bit Activation Gives Good ImageNet Accuracy

Network	Precision (W/A)	Model Size (MB)	IMAC ($\times 10^8$)	BMAC ($\times 10^9$)	FPMAC ($\times 10^8$)	Top-1 (%)	Top-5 (%)
Bi-RealNet-34 [31]	1/1	3.18	0	3.53	1.39	62.2	83.9
MeliusNet-29 [2]	1/1	5.10	0	5.47	1.29	65.8	86.2
MeliusNet-42 [2]	1/1	10.1	0	9.69	1.74	69.2	88.3
Real-to-Binary Net [33]	1/1	1.92	0	1.68	1.56	65.4	86.2
ReActNet-A [30]	1/1	4.56	0	4.82	0.12	69.4	-
BNN Ensemble [59]	1/1	11.52	0	10.10	8.34	61.0	-
FP-BNN [28]	1/1	11.4	1.10	1.03	0	42.9	66.8
JPEGCompress [34]	1/8	0.72	6.21	0	0	70.8	90.1
Synetgy [51]	4/4	2.16	3.30	0	0.09	68.3	88.1
ResNet-18 [17]	8/8	11.69	0	0	18.2	69.8	89.1
MobileNet [20]	8/8	4.20	0	0	5.69	70.6	-
MobileNet V2 [41]	8/8	3.47	0	0	3.00	71.8	91.0
ShuffleNet 1.5× [54]	8/8	3.40	0	0	2.92	71.5	90.2
FracBNN	1/1.4	4.56	0.01	7.30	0	**71.8**	90.1

Concluding Remarks

- FP8 is likely to be the standard in a few years
  - It overcomes the int8 limitation and, above all, is easy to use
- In near future, a mix of individual FP8 and block-level Int4 is expected
  - Layerwise/channelwise Block-FP8 = int3/4/5 representation
    - E.g., a layerwise block-FP8 (E4M3) = 4bit tensor with a shared layerwise scale in int4
- For widespread usage of low precision computation
  - Ease of use: pseudo quantization noise methods, e.g., NIPQ is promising since they need little manual tuning
  - Low cost training: Currently PTQ (post-training quantization) is applied to large language models (LLMs) like GPT-3 due to training cost of QAT. However, for higher qualities and efficiency, we will ultimately need QAT for LLMs

Weekly Lecture / Lab Schedule

- W1 (March 6) Class introduction / (March 8) Orientation & team formation
- W2 13 Verilog 1 Combinational circuits / 15 Verilog 1 (tool installation, adder & multiplier combinational logic)
- W3 20 Verilog 2 Sequential circuits / 22 Verilog 2 (memory i/o, FSM sequential logic)
- W4 27 AI application introduction 1, Amaranth introduction 1 / 29 Amaranth (tool installation, MAC, adder tree)
- W5 4/3 AI application introduction 2, Amaranth introduction 2 (memory i/o, FSM sequential logic) / 5 Amaranth (PE)
- W6 10 AI application introduction 3, Neural network accelerator 1 / 12 Amaranth (PE controller)
- W7 17 Neural network accelerator 2 / 19 Q&A
- W8 24 **Mid-term exam** / 26 Amaranth (outer product accelerator)
- W9 5/1 Reading data from memory 1 (VA2PA, interconnect) / 3 PyTorch (simple CNN on MNIST)
- W10 8 Reading data from memory 2 (DRAM main memory) / 10 Quantization aware training (QAT)
- W11 15 Compressing networks (pruning, low precision) / 17 Convolution lowering & tiling
- W12 22 Zero-skipping & low-precision hardware accelerator / 24 PyTorch – Amaranth communication
- W13 29 Invited talk (1h online Google Edge TPU, 1h offline Furiosa Date to be determined soon) / 31 Zero skipping
- W14 6/5 **Final exam** / 7 Project Q&A
- W15 12 (online) Claim & Project Q&A / 14 (online) Project Q&A, submission