Bandwidth-Efficient Deep Learning: The Tricks of the Trade

From network pruning to parameter reduction and acceleration

Dr. ir. Maurice Peemen

Electrical Engineering

My Background

- Masters Electrical Engineering at TU/e
- PhD work at TU/e
  - Thesis work with Prof. Dr. Henk Corporaal
  - Topic: Improving the Efficiency of Deep Convolutional Networks
- Staff Scientist at Thermo Fisher Scientific (formerly FEI Company)
  - Electron Microscopy Imaging Challenges
- Manager Data & AI Applications: Leading a global team of AI experts

The world leader in serving science

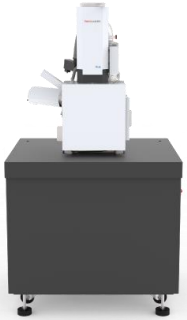
>125,000
colleagues

7,000
R&D scientists/engineers

$1.5B
invested in R&D

>$40B
in revenue

Electron Microscopes to Enable Science



SEM
Scanning Electron Microscope

Dualbeam
SEM + Focused Ion Beam

Wafer Dualbeam
Dualbeam for 300mm semiconductor wafers

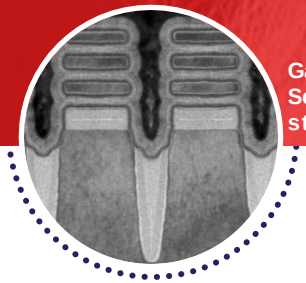
TEM
Transmission Electron Microscope

Working together

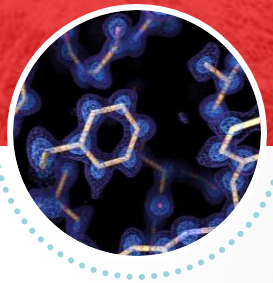
to enable our customers to make the world healthier, cleaner and safer.

Our instruments, software and workflow solutions

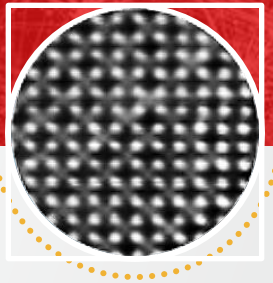
enable automation, improved ease-of-use, and expanded access to techniques that advance research and product development.



Gate-all-around Semiconductor device structure



Atomic Protein structure



Atomic structure of Li-Ion battery

Semiconductor

- Accelerate research and maximize production yields
- Enable development supporting 6G+, AI, edge & high-performance computing
- Top memory, logic, display, and foundry semiconductor manufacturers use our instruments & workflows

Life Sciences

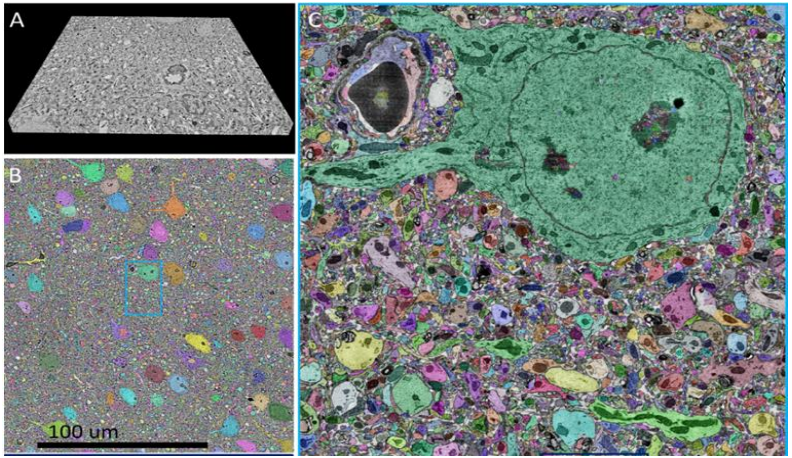
- Understand building blocks of life, including protein and cellular structure
- Use Cryo Electron Microscopy (Cryo-EM) to help scientists create better vaccines
- Top US and internationally funded institutes and pharma companies utilize cryo-EM solutions

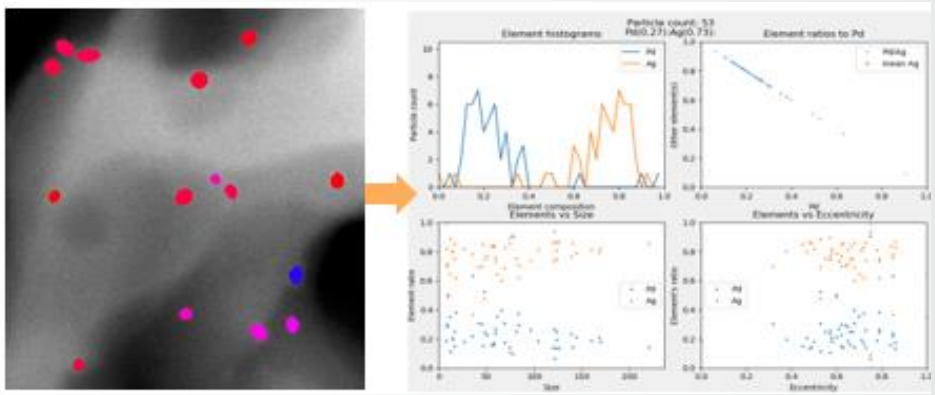
Materials Science

- Explore, characterize and develop new materials
- Understand failure mechanisms to develop better battery technology
- Top materials science research institutes and companies use our instruments in R&D

Leveraging AI in Electron Microscopy

Why are we exploring AI?





Solve hard customer problems

- Solve previously unaddressable problems
- Increase customer productivity
- Automate tedious, repetitive, error-prone tasks

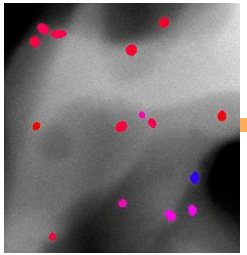
Solve problems faster

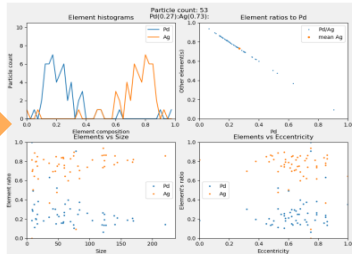
- Speed-up for existing but slow solutions
- Increase development productivity
- Build products faster

Majority of current AI applications for EM are image-based

AI in Thermo Fisher products enable customer value

Artificial Intelligence transforms images into physics, biology and process insights

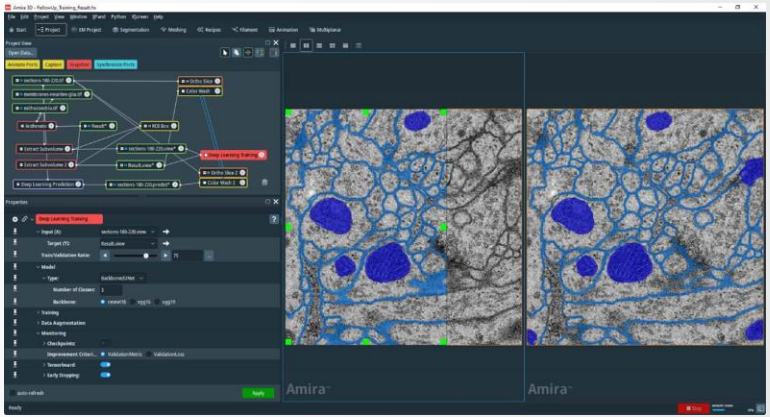


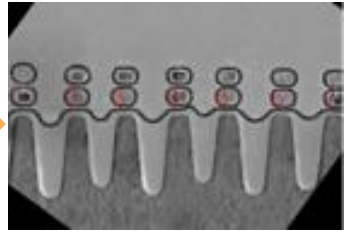


Nanoparticles

Get particle composition 200X faster on Talos TEM

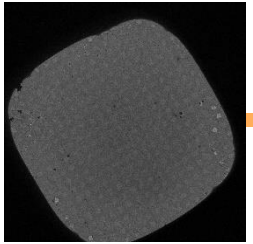
Deep Learning Training

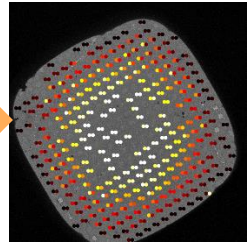




Semiconductor Metrology

Orient sample, drive to specific device on Metrios TEM



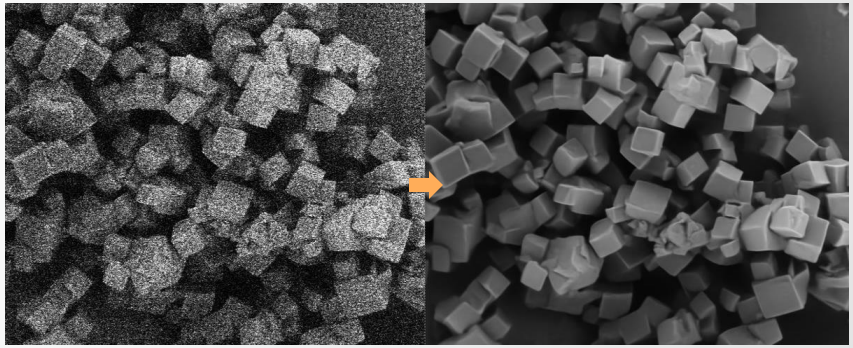


Cryo-EM

Identify the best places to image proteins on a Krios TEM

Live Denoising

Apreo SEM



We have several student assignments

Open assignments for Master internship or Master graduation projects

- From H2 2024 onwards

- Contact Dr. Alexios Balatsoukas-Stimming for assignment descriptions
  - Topics around AI application development
  - Training algorithms
  - Infrastructure and MLOps workflows

Bandwidth-Efficient Deep Learning: The Tricks of the Trade

From network pruning to parameter reduction and acceleration

Dr. ir. Maurice Peemen

Electrical Engineering

Artificial Intelligence is changing our lives

Self-driving cars

AlphaGo

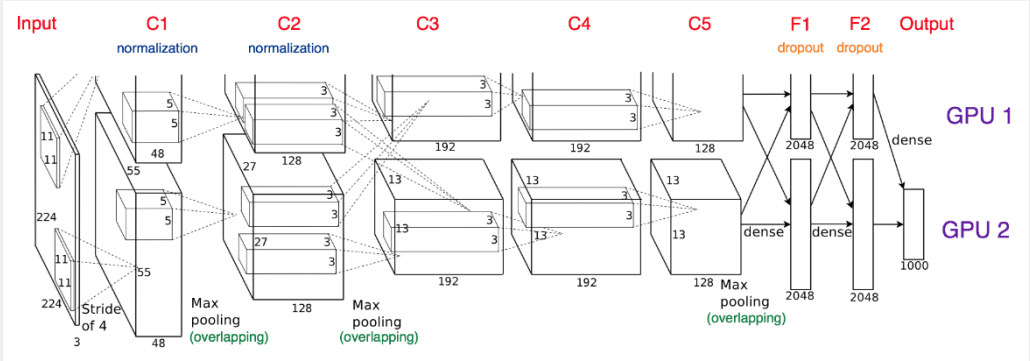
Live machine translation

Smart robots

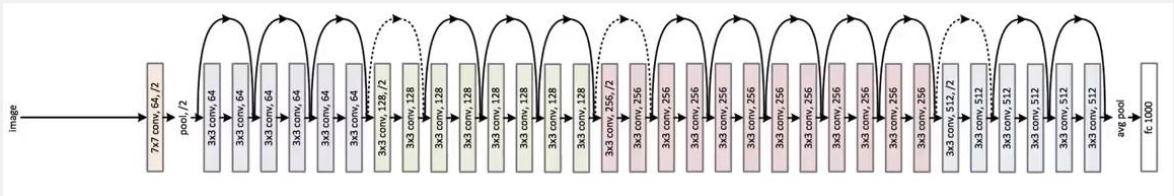
TU/e

Deep learning models have become huge

- Image Recognition
- AlexNet (2012) ILSVRC winner
  - 8 layers, 62Mparameters
  - 1.4 GFLOP inference
  - 16% error rate



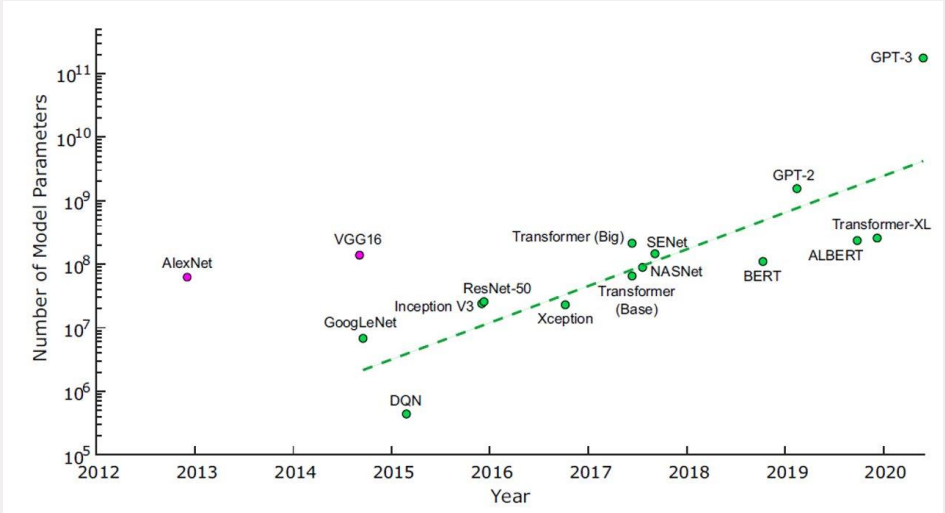
- ResNet (2015) ILSVRC winner
  - 152 layers, 60Mparameters
  - 22.6 GFLOP inference
  - 6.16% error rate



TU/e

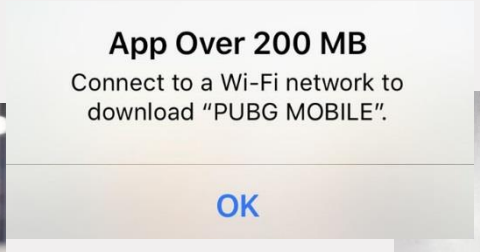
Trend towards huge models

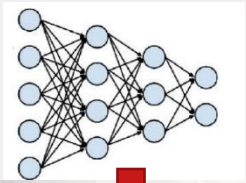
- Large language Models
  GPT-3 (175Billion parameters)
  - Chat GPT is derived from this model
- Protein Folding
  AlphaFold2 and AlphaFold-Multimer
  EvoFormer Transformer model
- ImageNet Record (90.9% Top1)
  CoAtNet-7 (2.44Billion parameters)



The first challenge: Model Size

- Competition winning networks like AlexNet (2012) and ResNet152 (2015) are large
  - About 60M parameters resulting in a coefficient file of 240MB
- Hard to distribute large models through over-the-air updates





The second challenge: Speed

- Network design and training time have become a huge bottleneck

	Error rate	Training time
ResNet 18:	10.76%	2.5 days
ResNet 50:	7.02%	5 days
ResNet 101:	6.21%	1 week
ResNet 152:	6.16%	1.5 weeks

Training time benchmarked with fb.resnet.torch using four M40 GPUs

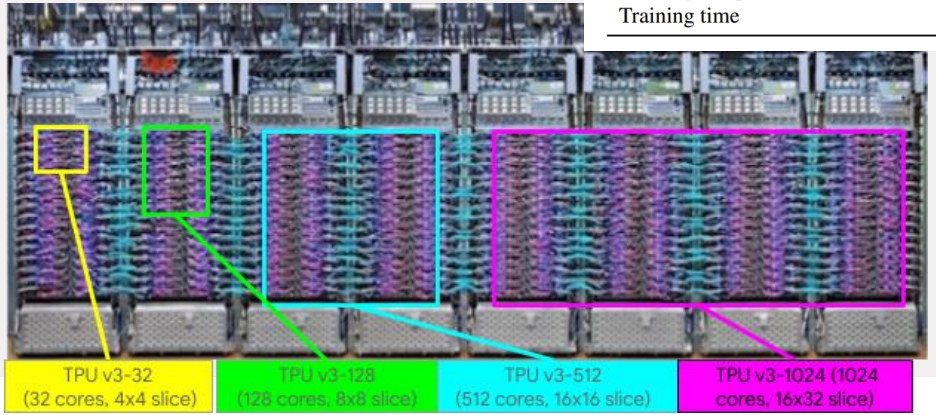
- Such a long training time limits the productivity of ML researchers

Training an AlphaFold Evoformer

Training for ~ 3 weeks

On 128 TPU Cores

	initial training	first fine-tuning		second fine-tuning				
Model	1	1.1	1.2	**1.1.1**	**1.1.2**	**1.2.1**	**1.2.2**	**1.2.3**
Parameters initialized from	Random	Model 1	···	Model 1.1	···	Model 1.2	···	···
Number of templates N_{templ}	4	4	0	4	···	0	···	···
Sequence crop size N_{res}	256	···	···	384	···	···	···	···
Number of sequences N_{seq}	128	512	···	···	···	···	···	···
Number of extra sequences $N_{\text{extra\_seq}}$	1024	···	···	5120	1024	5120	···	1024
Initial learning rate	10^{-3}	$5 \cdot 10^{-4}$	···	···	···	···	···	···
Learning rate linear warm-up samples	128000	0	···	···	···	···	···	···
Structural violation loss weight	0.0	1.0	···	···	···	···	···	···
"Experimentally resolved" loss weight	0.0	0.01	···	···	···	···	···	···
Training samples ($\cdot 10^6$)	9.2	1.1	1.7	0.3	0.6	1.4	1.1	2.4
Training time	6d 6h	1d 10h	2d 3h	20h	1d 13h	4d 1h	3d	5d 12h



The third challenge: Energy Efficiency

AlphaGo **1920 GPUs and 280 GPUs, $3000 electric bill per game**

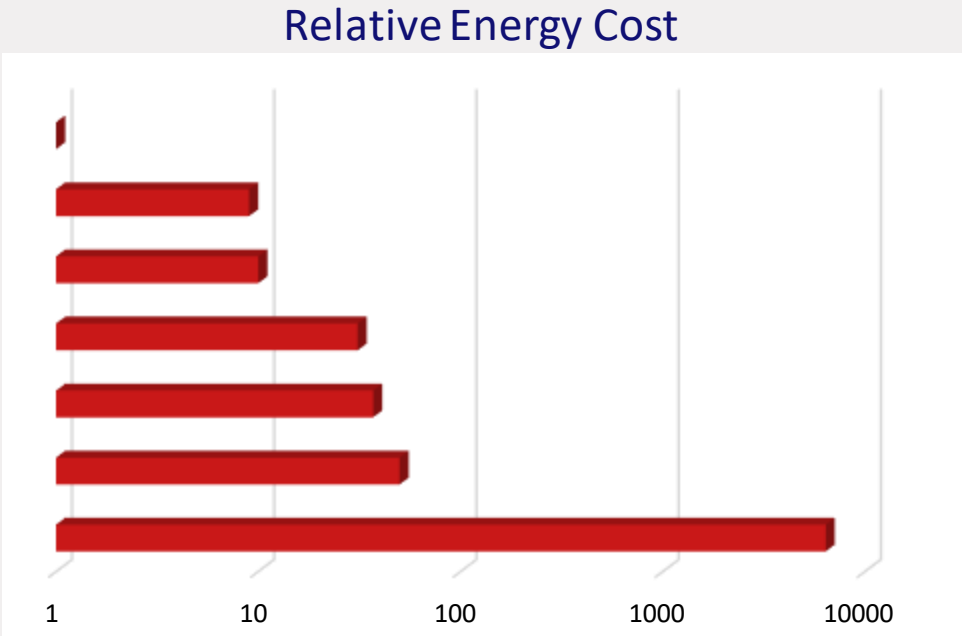
On smartphone: battery drains quickly
On data-center: Increased TCO

TU/e

Where is the Energy Consumed?

- Larger model → More memory references → More energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit fload ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640



TU/e

Where is the Energy Consumed?

- Deep learning models with many parameters are expensive!
- Reduce parameters to improve efficiency

1 External Memory Transfer

=

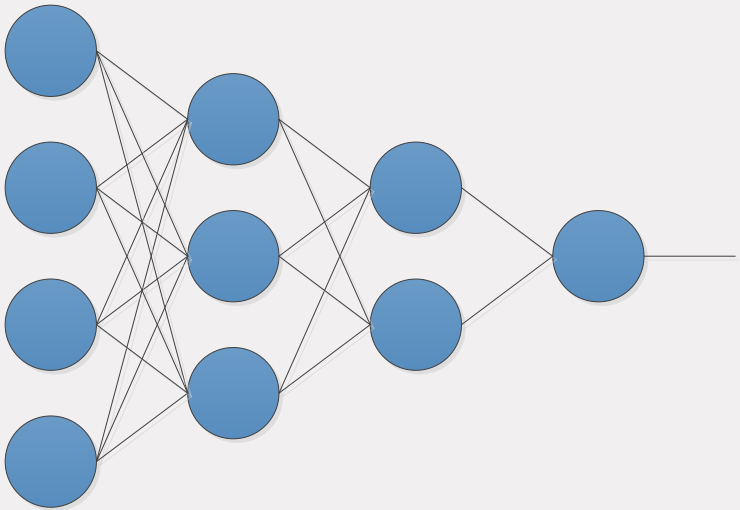
1000 Multiply or ADD operations

+ x + x + x + x + x + x + x + x + x + x +
x + x + x + x + x + x + x + x + x + x + x
+ x + x + x + x + x + x + x + x + x + x +
x + x + x + x + x + x + x + x + x + x + x
+ x + x + x + x + x + x + x + x + x + x +
x + x + x + x + x + x + x + x + x + ...

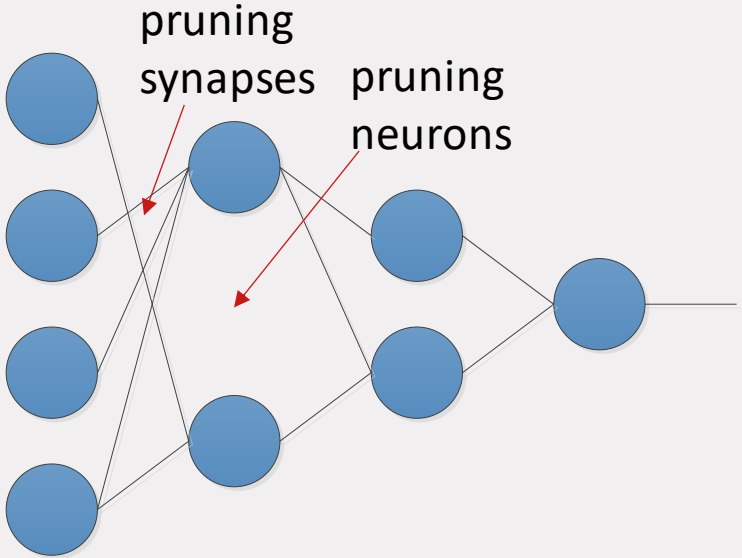
TU/e

How to compress a deep learning model?

before pruning



after pruning



TU/e

What can you win by Network Pruning?

Without loss of accuracy, you can reduce storage and compute substantially!

Neural Network	#Parameters			MACs
	Before Pruning	After Pruning	Reduction	Reduction
AlexNet	61 M	6.7 M	**9** ×	3 ×
VGG-16	138 M	10.3 M	**12** ×	5 ×
GoogleNet	7 M	2.0 M	**3.5** ×	5 ×
ResNet50	26 M	7.47 M	**3.4** ×	6.3 ×
SqueezeNet	1 M	0.38 M	**3.2** ×	3.5 ×

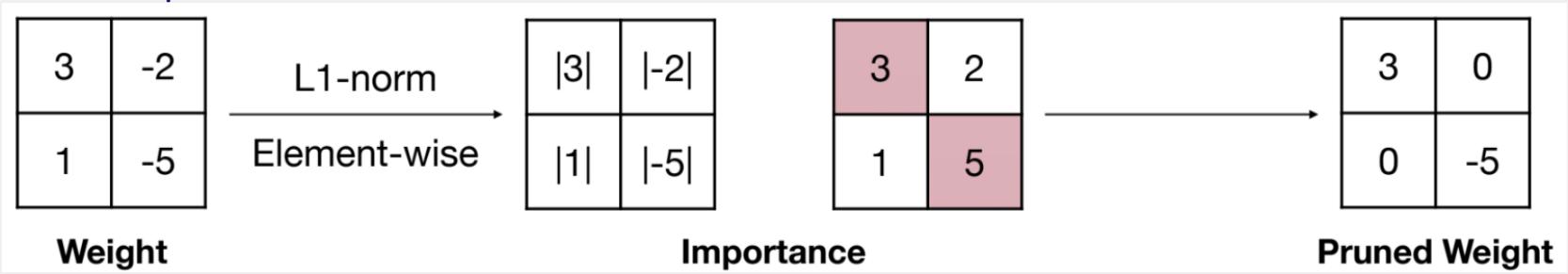
Why would the coefficient reduction differ from the MAC reduction?

Scenario 1 you only have a model: How to prune?

Magnitude-based Pruning: A heuristic pruning criterion

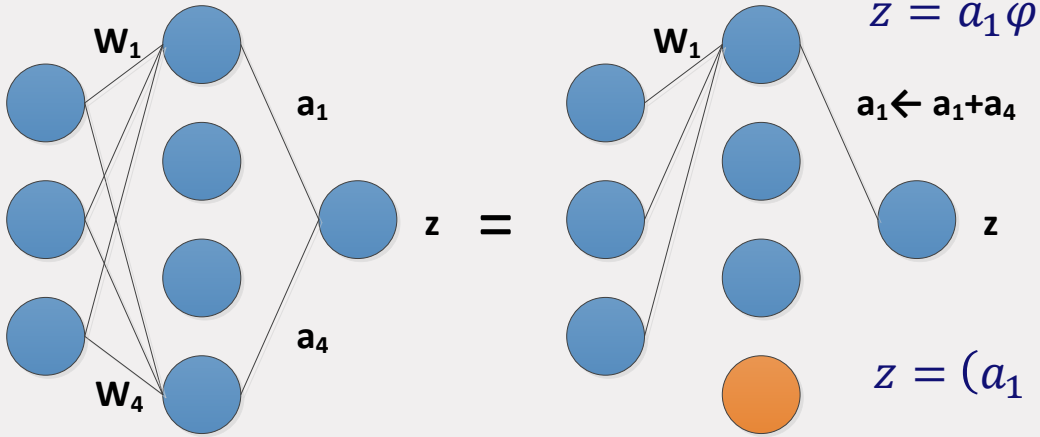
- Magnitude-based pruning assumes weights with **larger absolute values** to be more important than small weights
  - For element-wise pruning Importance $= |W|$

- Example



Scenario 1 You only have a model: How to prune?

- Naïve pruning: Remove weights based on magnitude, weights close to zero are removed
  - No well-founded theory, error increases rapidly
- Data-Free parameter pruning based upon weight similarity



$$z = a_1\varphi(W_1^T X) + a_2\varphi(W_2^T X) + \cdots + a_n\varphi(W_n^T X)$$

Suppose, $W_1 = W_4$

Then, $\varphi(W_1^T X) = \varphi(W_4^T X)$

$$z = (a_1 + a_4)\varphi(W_1^T X) + a_2\varphi(W_2^T X) + \cdots$$

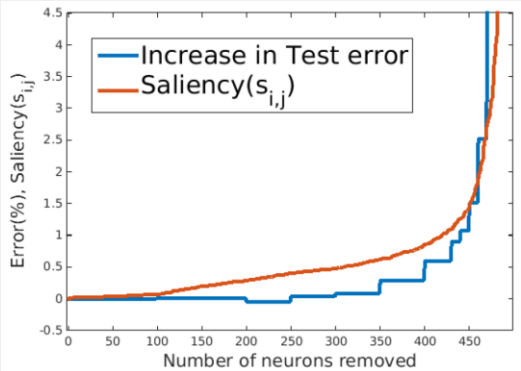
Srinivas et al. "Data-free parameter pruning for deep neural networks" BMVC 2015

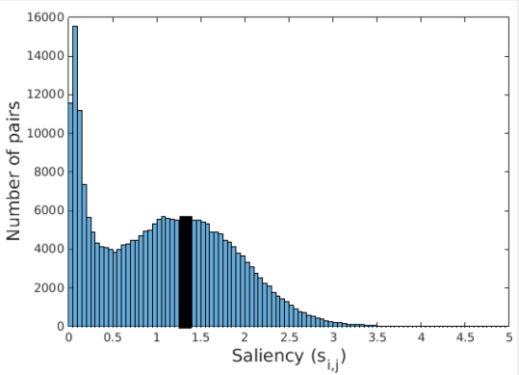
TU/e

Data-Free pruning uses only the model sensitivity

- In practice neurons are different, $\|W_1 - W_2\| = \|\varepsilon_{1,2}\| \geq 0$
  - Compute errors for Weight replacement and naïve removal, so called **saliency** matrix ***M***
    - Pick minimum entry in the list e.g. indices (i', j'), delete the j'^{th} neuron and update $a_{i'} \leftarrow a_{i'} + a_{j'}$
    - Update ***M*** by removing j'^{th} column and row, and update the i'^{th} column for updated $a_{i'}$

- When to stop?
  - Saliency in line with test error
  - Find the mode in the gauss like curve (around Saliency 1.2)
  - Stop at that point

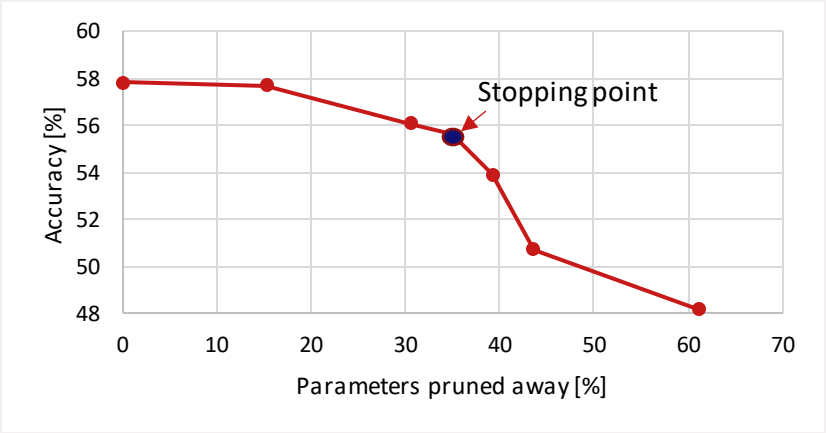




Srinivas et al. “Data-free parameter pruning for deep neural networks” BMVC 2015

Improve the efficiency of AlexNet

- Unpruned base accuracy 57.84%
- 61 Million weights in total 233MB
- Most parameters in fully connected layers
  - Data-Free pruning on layer 6 and 7



Layer	Operation	Parameters
Conv1+Relu	11x11@96	34,944
Conv2+Relu	5x5@256	614,656
Conv3+Relu	3x3@384	885,120
Conv4+Relu	3x3@384	1,327,488
Conv5+Relu	3x3@256	884,992
FC6+Relu	256x6x6@4096	37,748,736
FC7+Relu	4096@4096	16,777,216
FC8+Relu	4096@1000	4,096,000
Total	Conv: 3.7 million (6%)	FC: 58.6 million (94%)

- Data-Free pruning about 1.5x Compression
  - Still 39.6M parameters 151MB
  - Accuracy reduces with 2.24%

TU/e

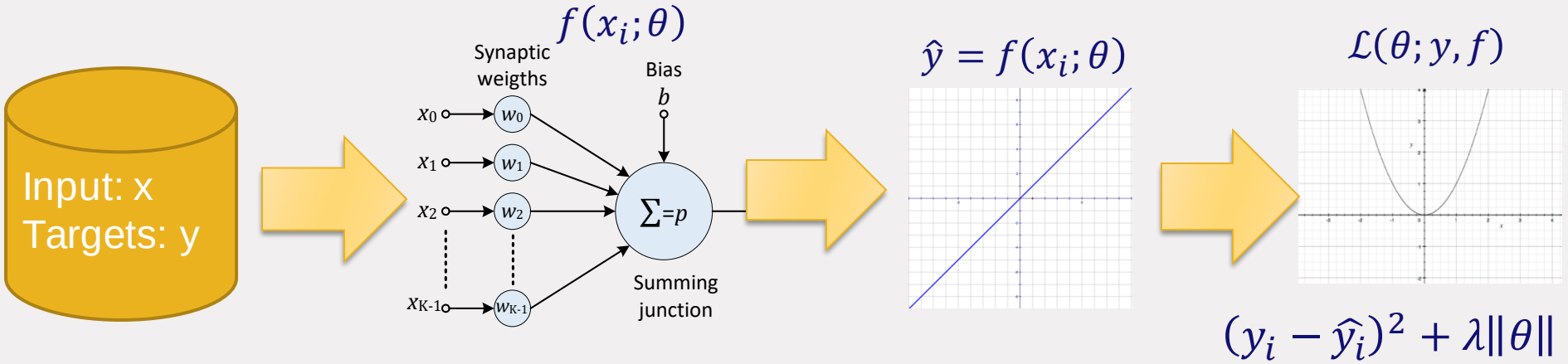
Scenario 2 You have data: How to prune aggressively?

- With access to training data, you can do a lot more

1. Train your network differently such that you have more zero weights
2. Retrain you network after pruning to fix the errors

Training a neural network: With a weight regularization

- The neural network is a function of inputs x_i and weights θ: $f(x_i;\theta)$
- Start with feed forward batch (i=1..64) through the network: $x_i \rightarrow \widehat{y_i}$
- Insert network results and desired target labels into a loss function: $\mathcal{L}(\theta; y, f)$
- Compute a score on how well the net performs, not only error also weight organization



TU/e

Weight regularization term

- L2 regularization: $\gamma \sum_{i=1}^{k} w_i^2$
  - Prevents large weights

- L1 regularization: $\gamma \sum_{i=1}^{k} |w_i|$
  - Results in sparse outputs

Tune the weights by gradient descent

- Compute the error gradients
- Update the coefficients to reduce error, also taking into account regularization
- Repeat

$$\theta^{t+1} = \theta^t - \eta_t \nabla_\theta \mathcal{L}$$

$$\frac{\partial f(x_i;\theta)}{\partial \theta}$$

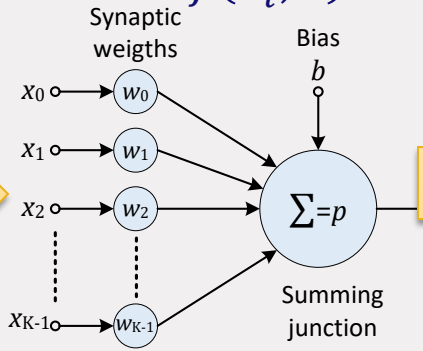
$$\frac{\partial \hat{y}}{\partial f}$$

$$\frac{\partial \mathcal{L}(\theta;\hat{y},f)}{\partial \hat{y}}$$

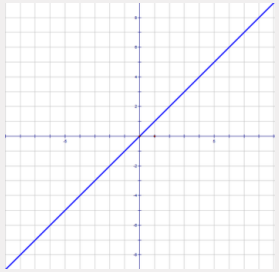
$$f(x_i;\theta)$$

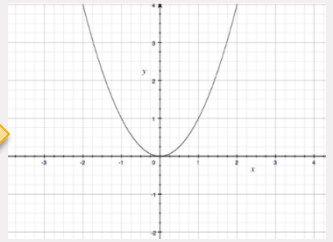
$$\mathcal{L}(\theta;y,f)$$

Input: x
Targets: y



$$\hat{y} = f(x_i;\theta)$$



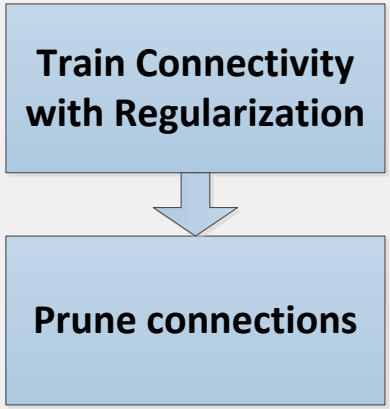


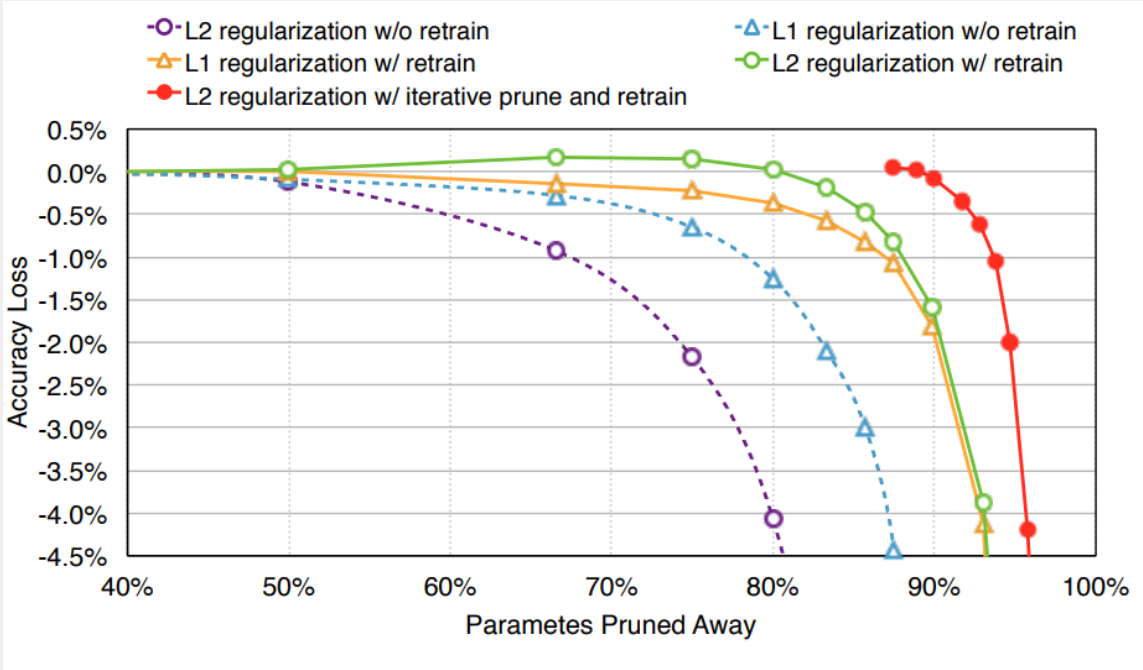
$$(y_i - \hat{y}_i)^2 + \lambda\|\theta\|$$

TU/e

Pruning zeros out after training with a regularizer

- Prune small coefficients
- L2 and L1 perform better than data free





- L1 performs better than L2 regularization: More zero connections

Why would L1 promote more sparsity than L2?

L1 vs L2 what happens with large weights?

L1 (W= -10) = 10

L2 (W= -10) = 100

L2 reduces the number of large weights

L1 vs L2 what happens with small weights?

L1 (W= 0.1) = 0.1

L2 (W= 0.1) = 0.01

L1 reduces the number of small weights

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

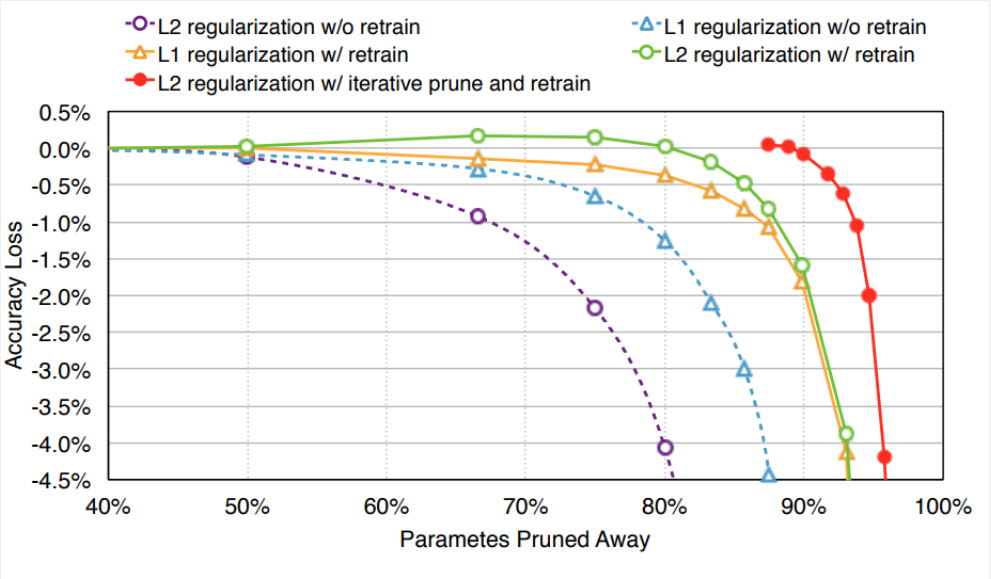
Loss function

Regularization Term

Improve the efficiency of AlexNet

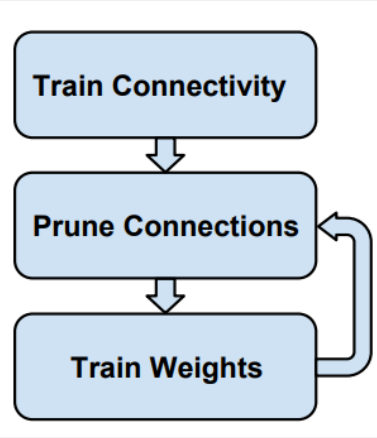
- With L1 regularization
  - Prune to 25% of the original size
  - 4x less parameters
  - From 61M to 15.25M parameters
  - From 233MB to 58MB
  - Accuracy -0.7%

How to reduce even more?



Iterative Pruning and Retraining

- Train a neural network until reasonable solution or download a pretrained net
  1. Prune the weights base on magnitudes that are less than a threshold
  2. Train the network until a reasonable solution is obtained
  3. Iterate to step 1

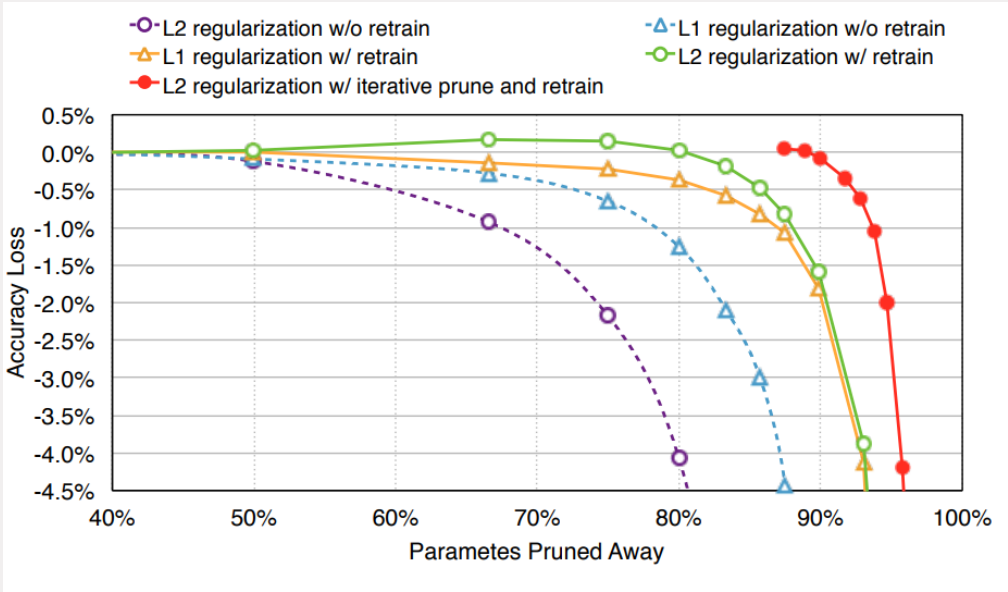


S. Han et al. “Learning both Weights and Connections for Efficient Neural Networks” (NIPS15)

Iterative Pruning and Retraining

- With one retraining to recover from pruning damage
  - L2 regularization performs better
  - 85% pruning of parameters, 6.7x reduction
  - From 61M to 9.1M Parameters
  - Or 233MB to 34.7MB

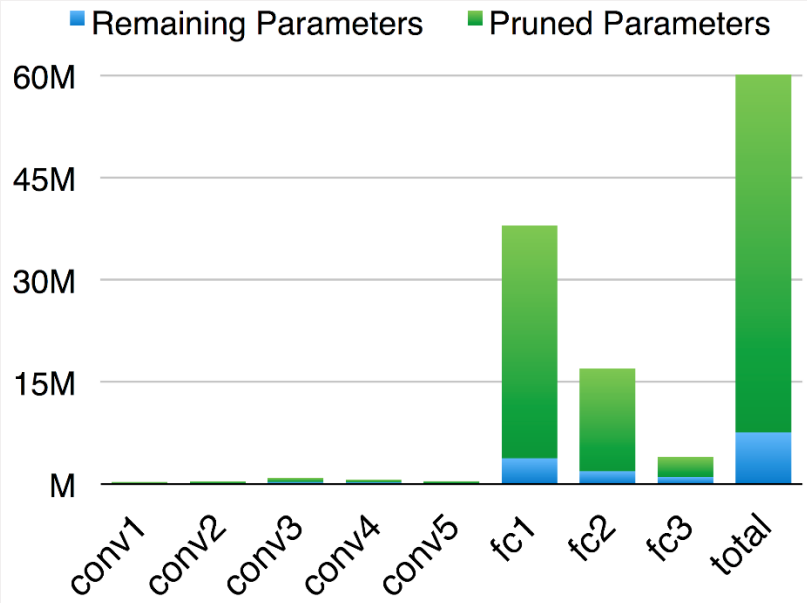
- Iterative pruning even better
  - Without loss of quality prune 90%
  - 10x reduction, from 61M to 6.1M
  - Or 233MB to 23.3MB



TU/e

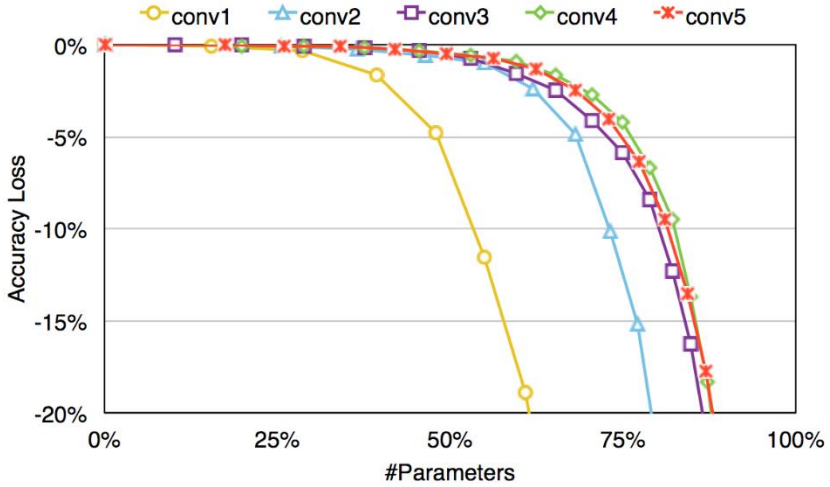
Analysis: What happened to Alexnet?

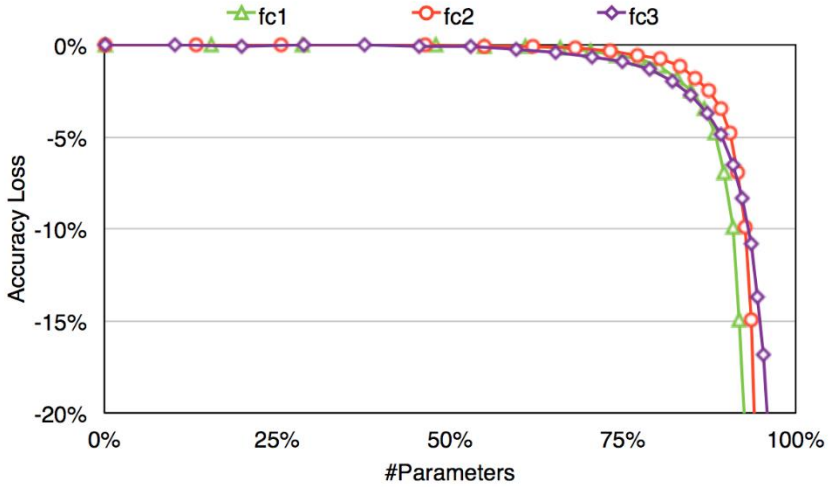
Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10
Total	61M	1.5B	54%	11%	30%



Where does pruning help the most?

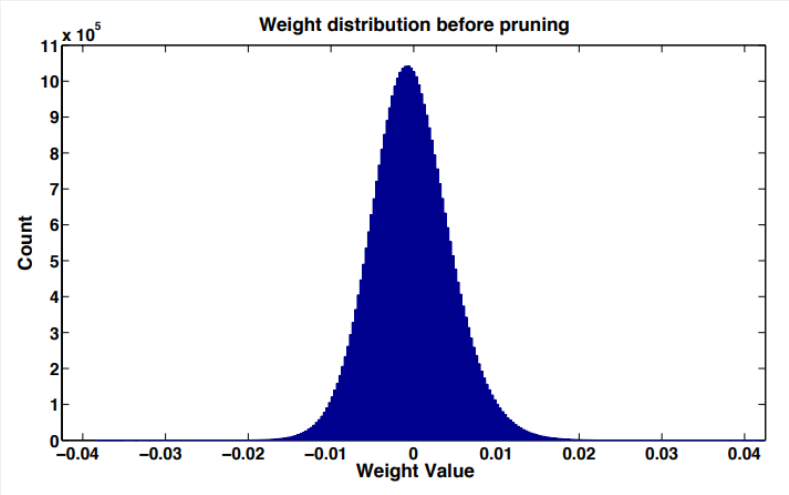
Fully connected layers

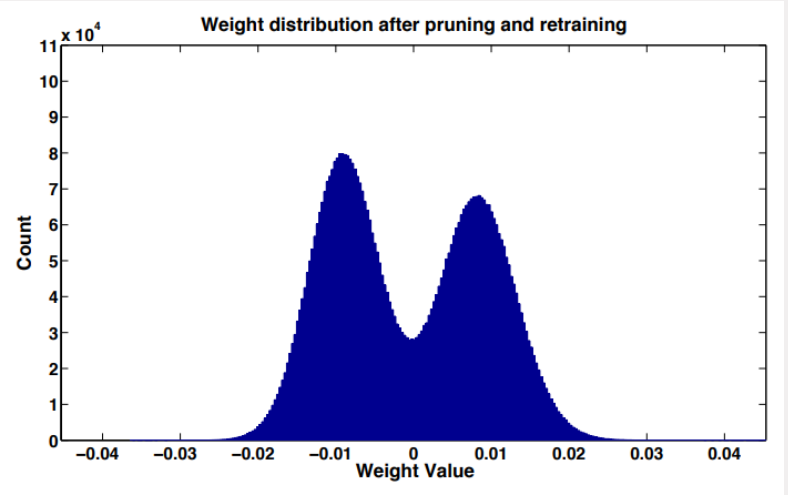




What happens to the weight distribution?

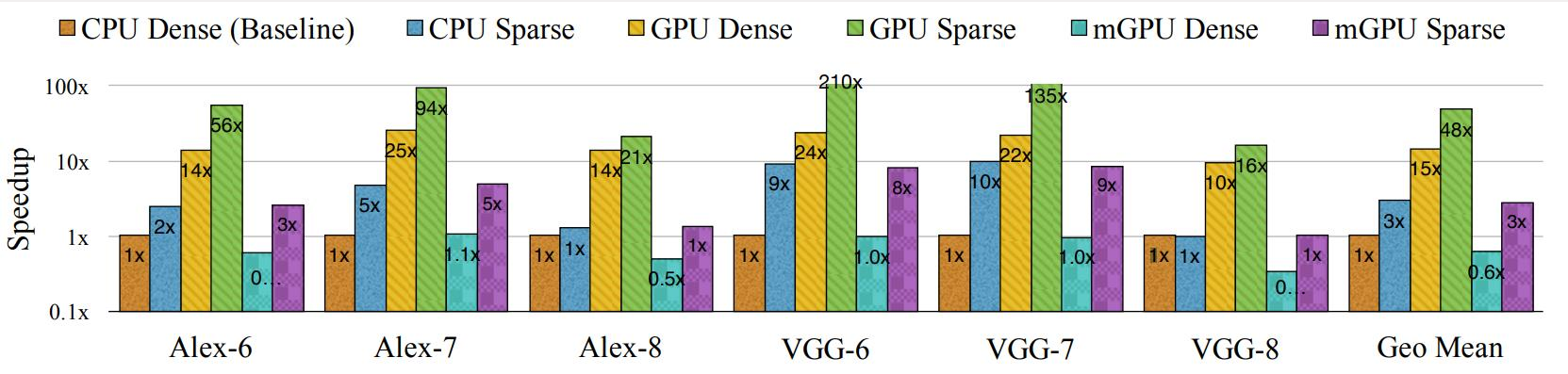
- Before: Most weights are close to zero; almost all between [-0.015, 0.015]
- After pruning: Bimodal distribution and more spread across x-axis, between [-0.025, 0.025]





Using sparse Matrix Computations

- Use Intel Core i7 5930K, MKL CBLAS GEMV (full) vs MKS SPBLAS CSRMV (sparse)
- Use NVIDIA GTX Titan X, cuBLAS GEMV (full) vs cuSPARSE CSRMV (sparse)
- Use NVIDIA Tegra K1 as embedded GPU

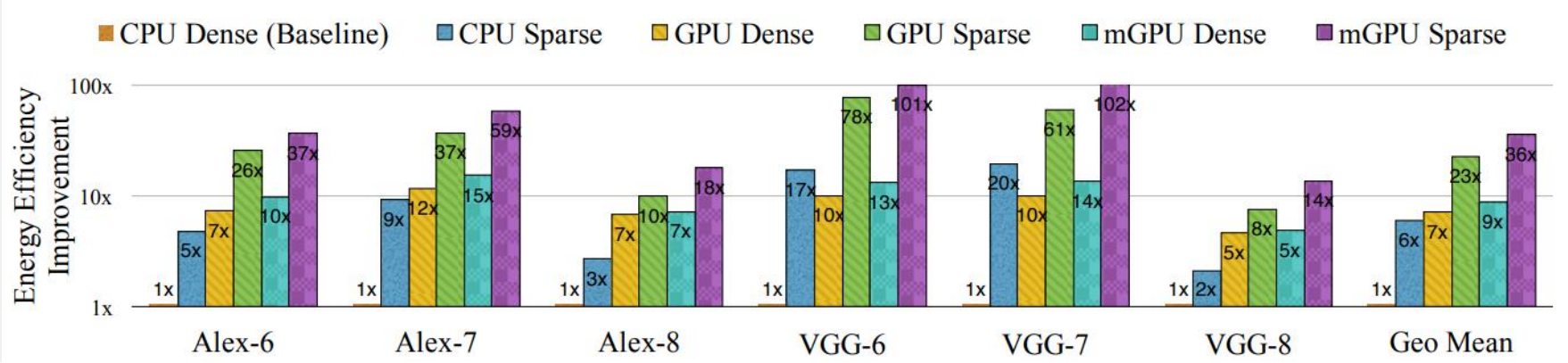


- 3x Speedup for CPU 3.2x for GPU and 5x for mobile GPU

S. Han "Efficient Methods and Hardware for Deep Learning" (PhD Thesis 2017)

Energy Efficiency

- 6x improvement CPU 3.2x improvement GPU 4x embedded GPU
- Difficult to exploit the large parameter reduction due to irregularity
  - Sparse matrices have also storage overhead; 16% for storing indices



TU/e

A Simple Model for the Benefits of Sparsity

Resources (storage, communication, compute)

Work (in bytes or operations)

Peak available (bytes, bytes/s, ops/s)

Efficiency of resource usage (0 < E ≤ 1)

$$R(esources) = \frac{W(ork)}{P(eak) * E(fficiency)}$$

Compare against dense implementation: **R**ealized **R**esource **R**eduction from sparsity

Compressed sparse row (CSR)

Estorage, sparse = 0.5 for CSR

$$R3 = \frac{R_{dense}}{R_{sparse}} = \frac{W_{dense}}{P_{dense}E_{dense}} * \frac{P_{sparse}E_{sparse}}{W_{sparse}}$$

Computation with Accelerators

Specialized matrix multiplication hardware is a disadvantage for sparsity

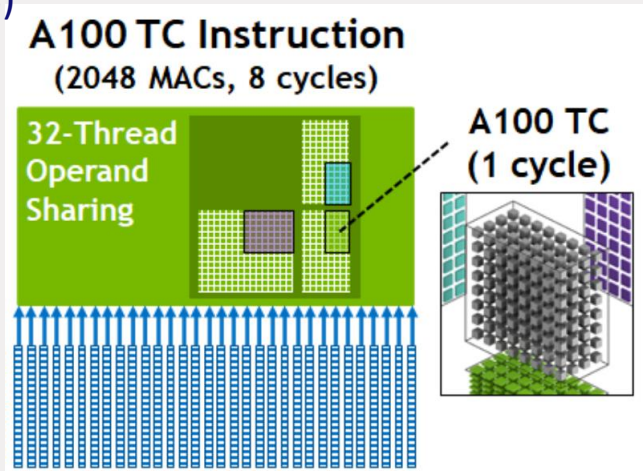
Latest Nvidia A100 GPU with tensor cores

Sparse Matrix Multiply (SpMM) peak is 1/16 of dense matrix multiplication (GEMM) peak

- To make it worse SpMM achieves 1/3 of peak, (**E**sparse ≤ 1/3)
- GEMM achieves 2/3 of peak (**E**dense)

Two main messages!

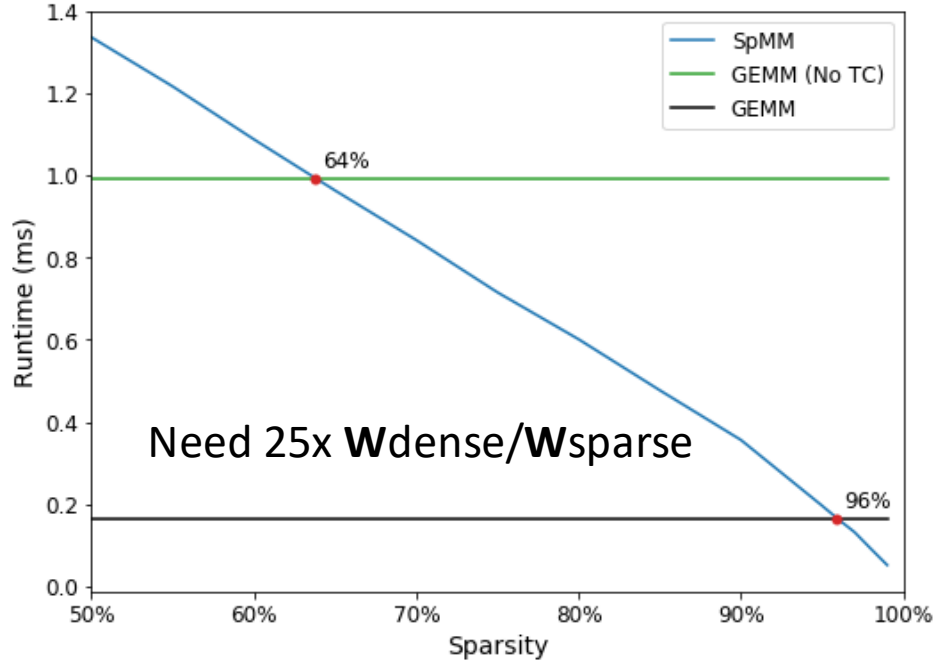
1. The compute workload reduction of sparse must be large 1/**W**compute,sparse to capture **E**compute,sparse
2. Increase HW capabilities for sparse **P**compute,sparse



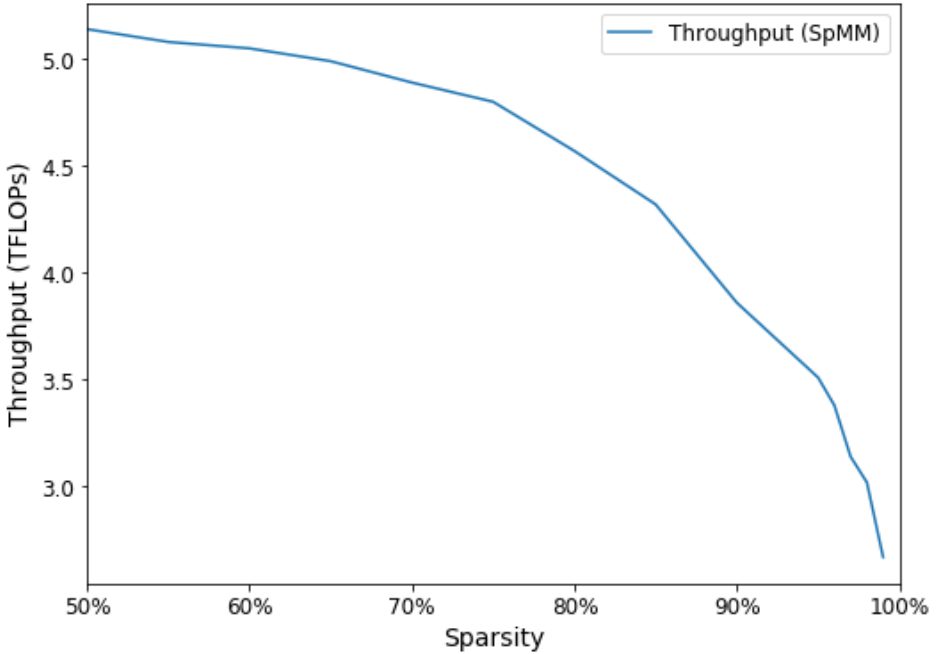
TU/e

The speed of sparse kernels in practice

Runtime of sparse kernels versus dense



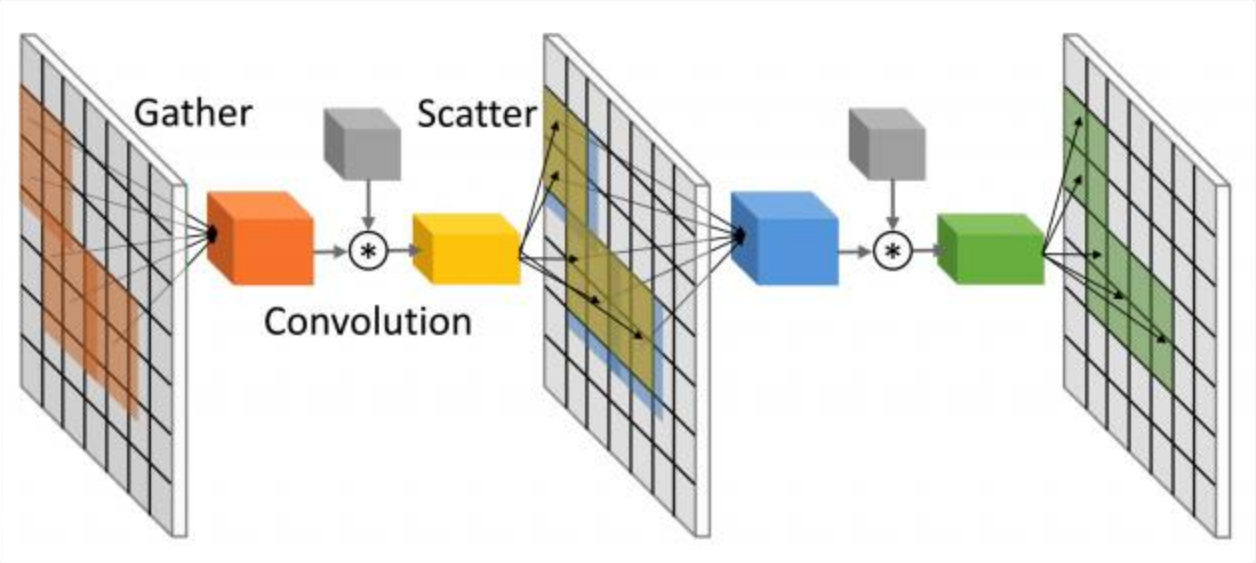
Throughput decrease of SpMM with more sparsity



TU/e

Need to find more sparsity

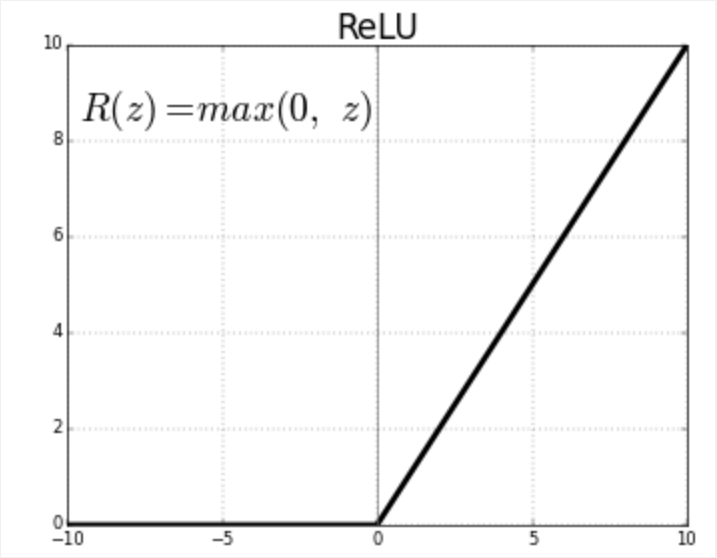
Sparsity in Activations



Activation Sparsity

- Modern network contain ReLU activations
- During inference plenty of these are 0

- Activation Sparsity is high
  - Resnet50 | 53% Natural AS
  - Resnet101 | 57% Natural AS

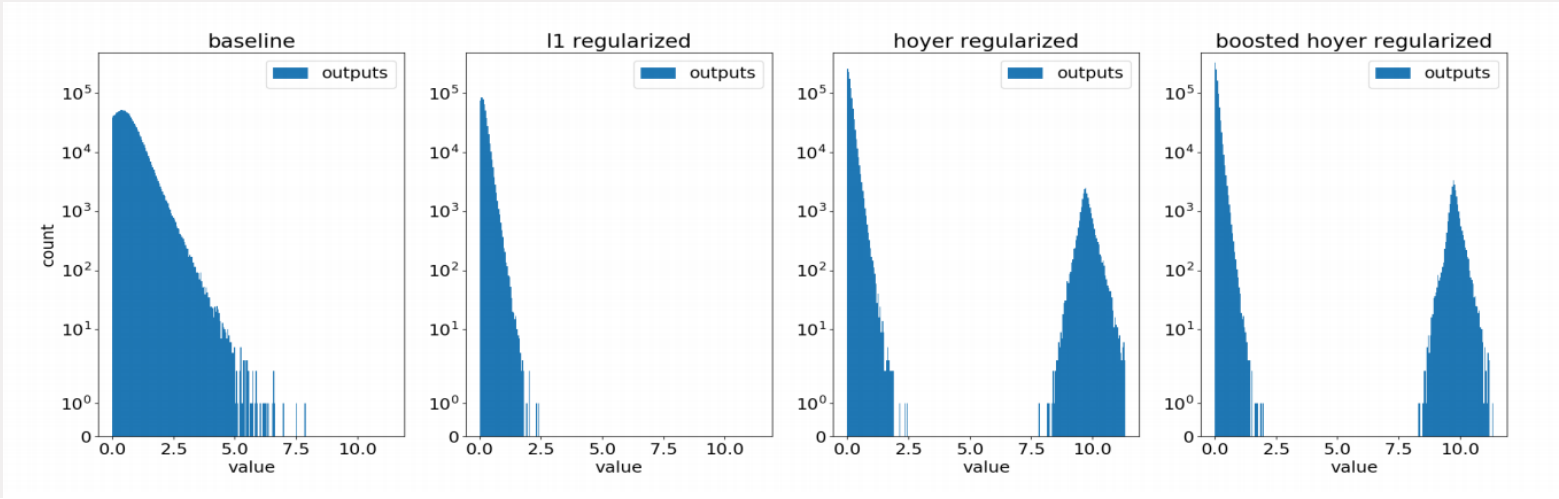


Boost activation sparsity

Instead of L1 regularization

Use Hoyer regularization

$$H(\vec{v}) = \frac{\left(\sum_{i=1}^{d} |v_i|\right)^2}{\sum_{i=1}^{d} v_i^2}$$

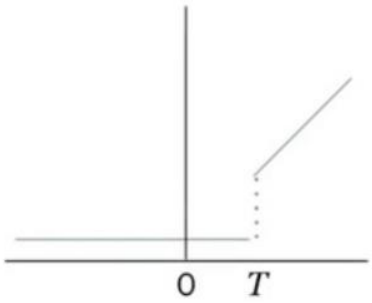


Boosted Hoyer regularization

Use a threshold to force more values to 0

$$FATReLU_T(x) = \begin{cases} x & \text{when } x \geq T; \\ 0 & \text{otherwise.} \end{cases}$$

FATReLU equation

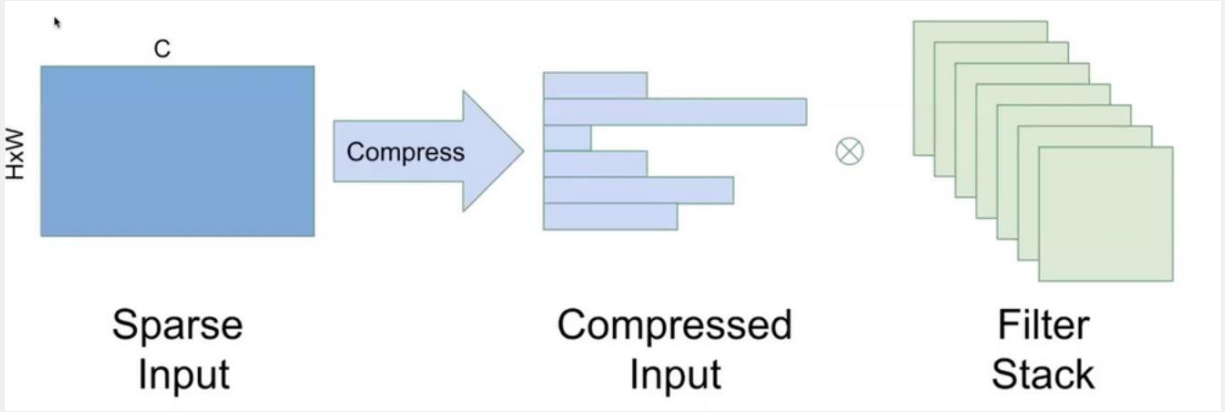


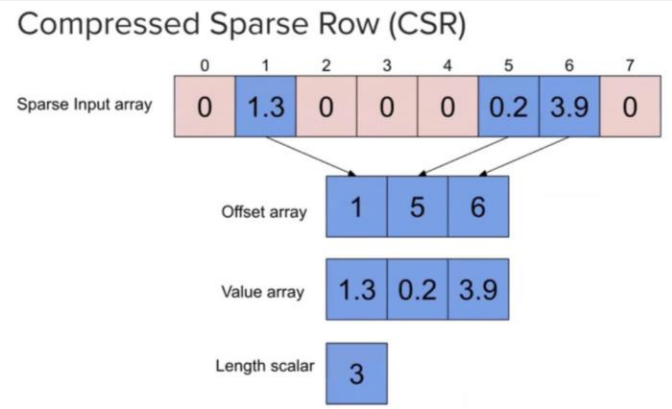
FATReLU activation function

Model	Baseline	L1	Hoyer	Boosted Hoyer
ResNet18	53%	55%	62 %	**67** %
ResNet50	53%	54%	61 %	**65** %
Mobilenet	48%	52 %	58 %	**60** %

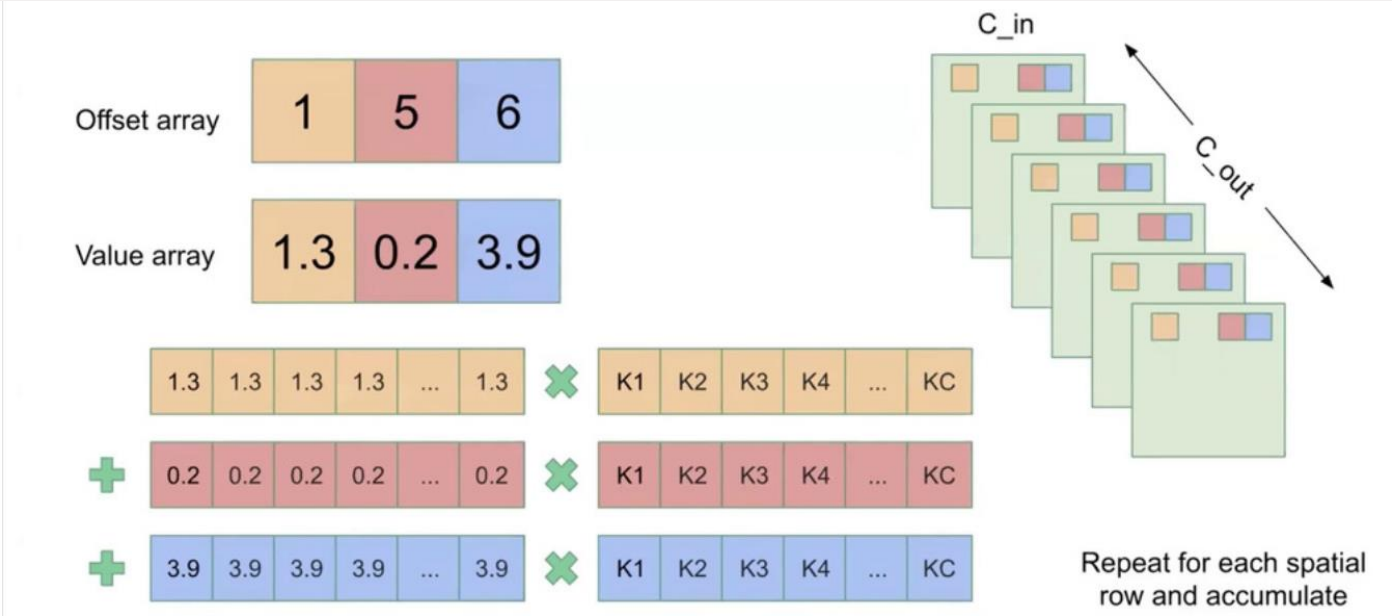
Implement: Compress your feature maps and reuse

CSR format





Implement Convolutions as Sparse SIMD operations

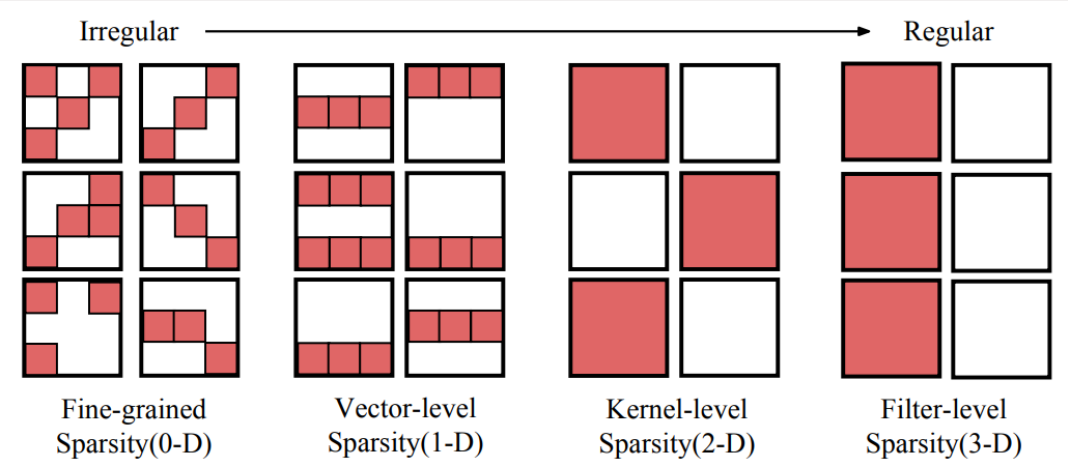


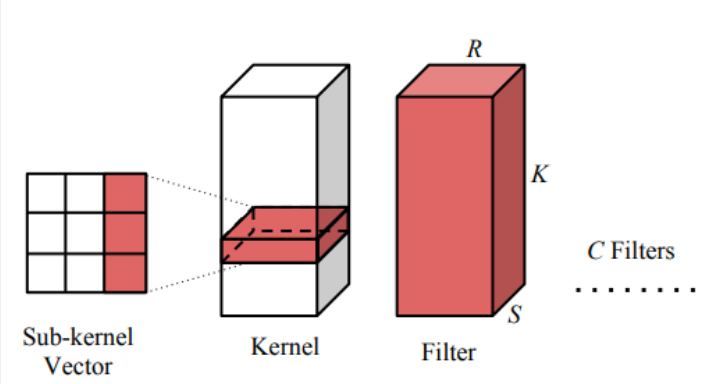
Results of activation sparsity

Model	MXNet+MKL-DNN	NVIDIA K80	Dense	Natural Sparsity	Hoyer Reg.	Boosted Hoyer
ResNet18	113.41	100.16	107.25	68.40	63.67	**60.92 (1.86x)**
ResNet50	317.49	350.2	256.06	194.86	183.21	**180.5 (1.75x)**
Mobilenet	88.55	114.3	62.64	58.93	51.80	**49.77 (1.78x)**

From Fine to Coarse-Grained Pruning

- Prune to match the underlying data-parallel hardware
  - E.g. prune by eliminating entire filter planes

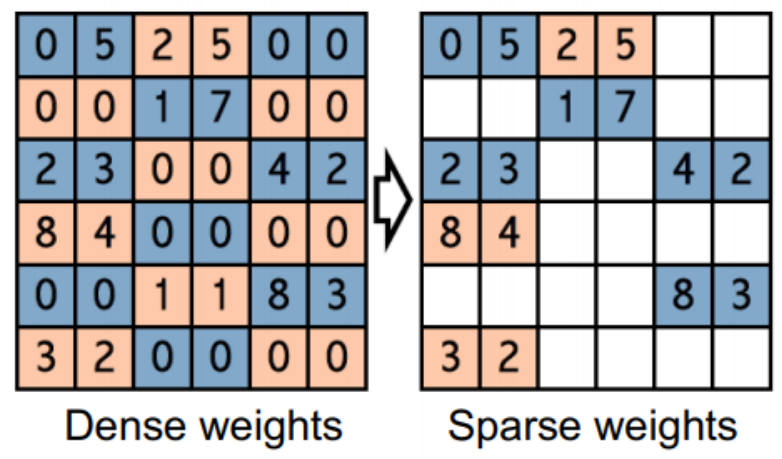


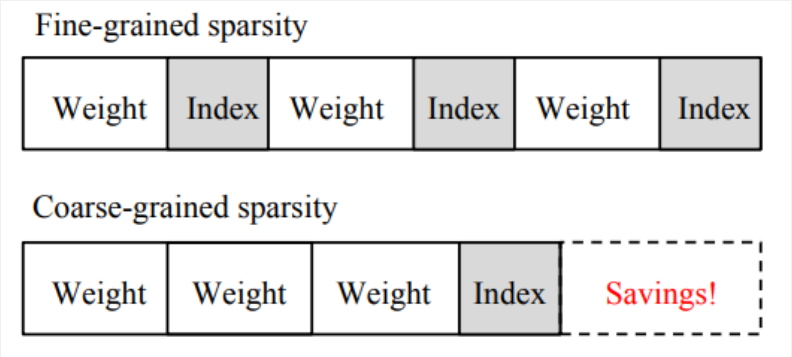


H. Mao et al. "Exploring the regularity of sparse structure in ConvNets" (CVPR 2017)

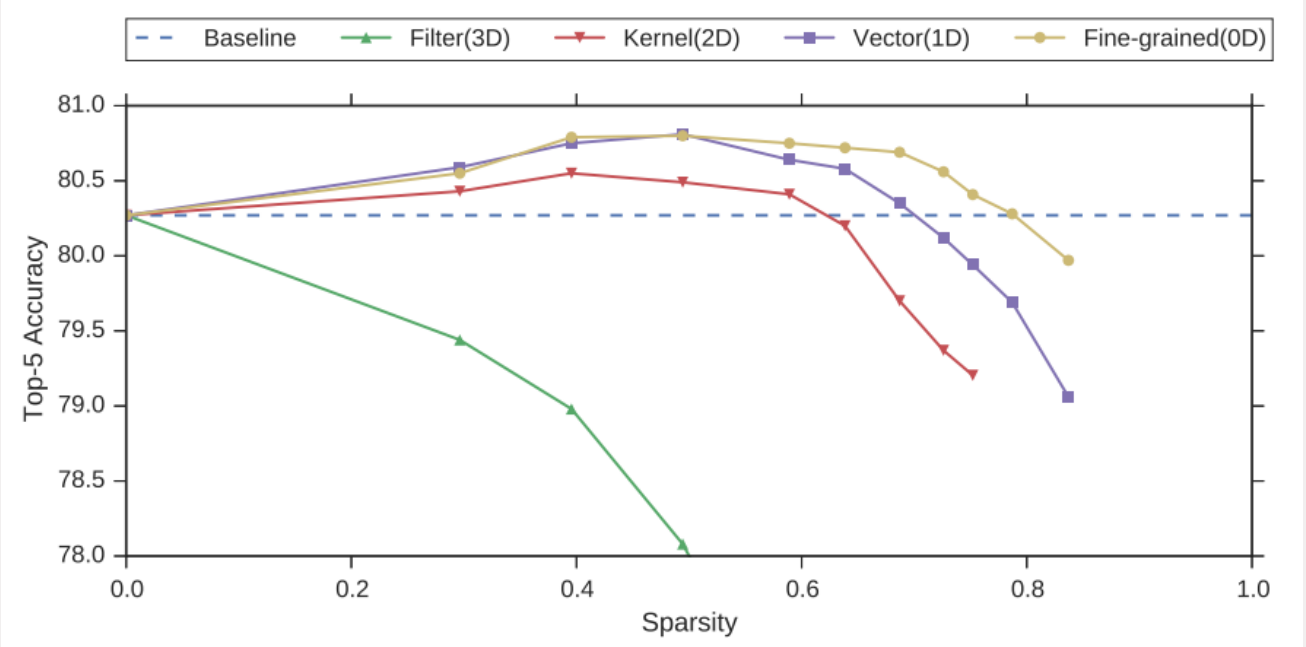
Structured Pruning

- Example 2-way SIMD
- Less storage overhead





How does coarse grained pruning translate in accuracy?

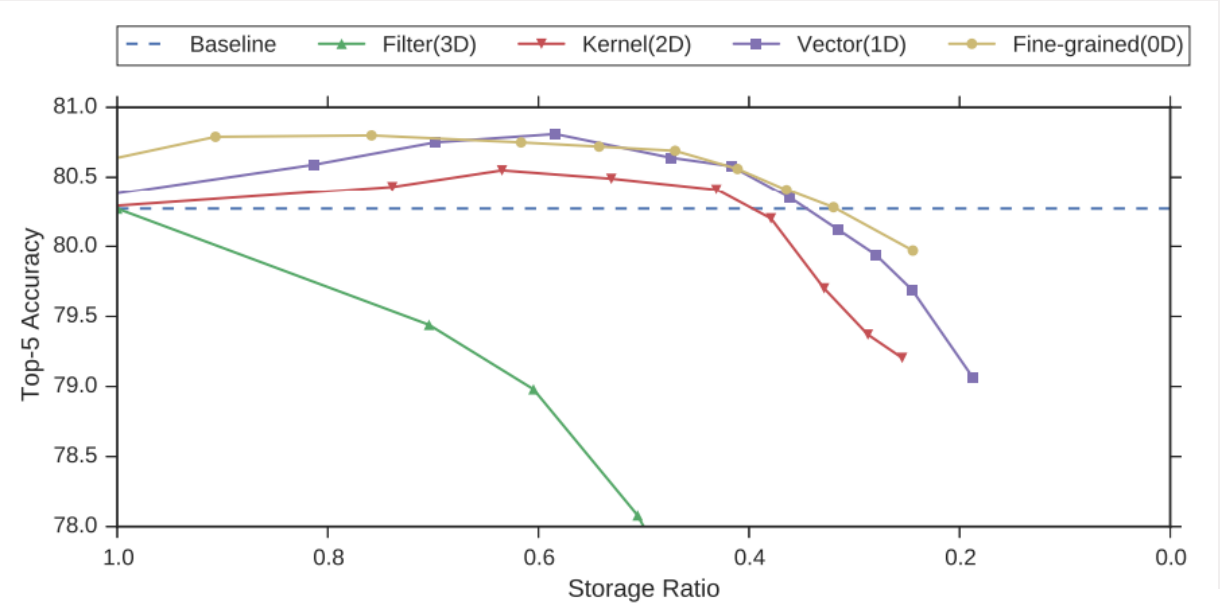


TU/e

Accuracy versus storage requirements

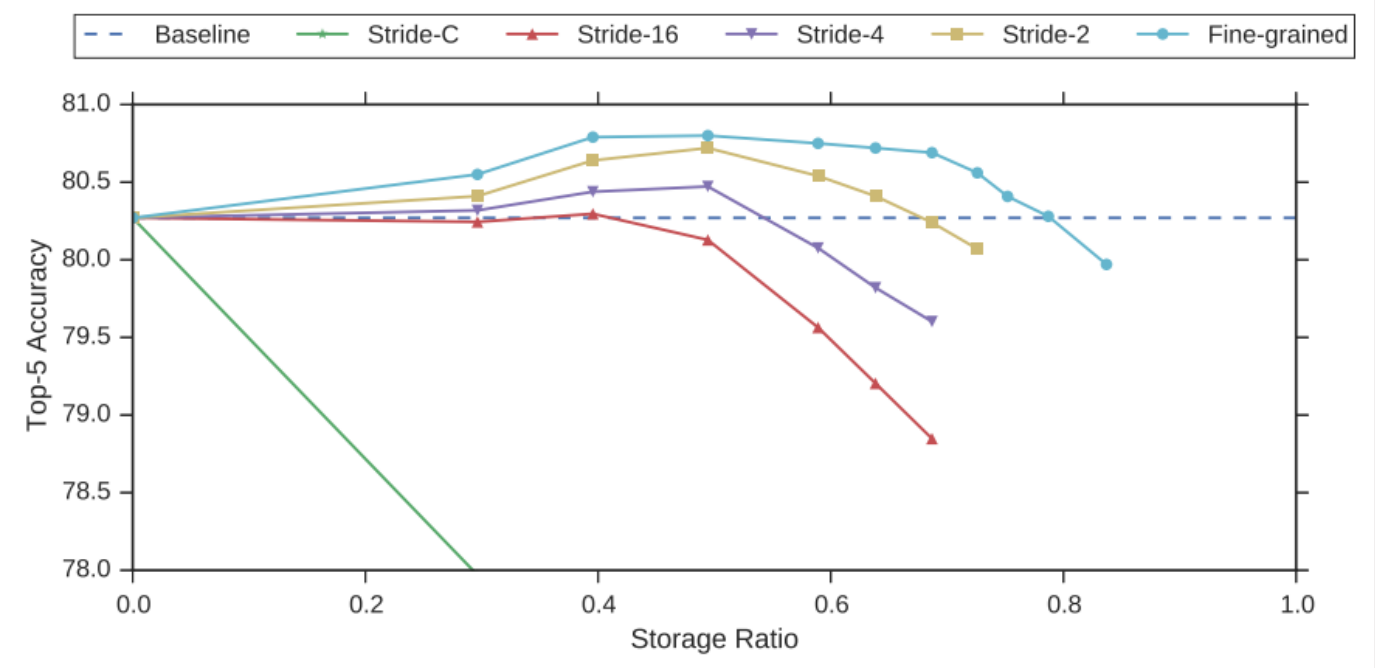
Fine, vector and kernel are close in storage ratio

- Less overhead for coarse pruning



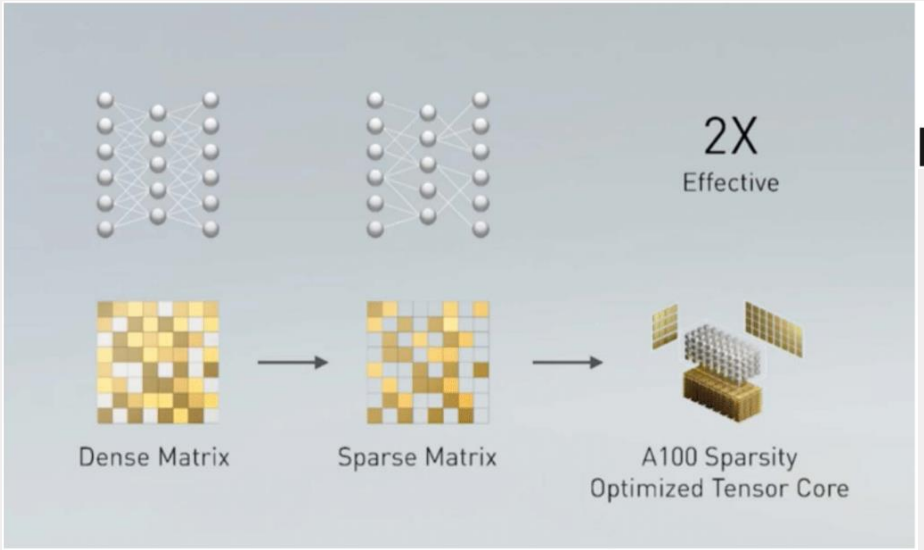
TU/e

Stride based pruning



TU/e

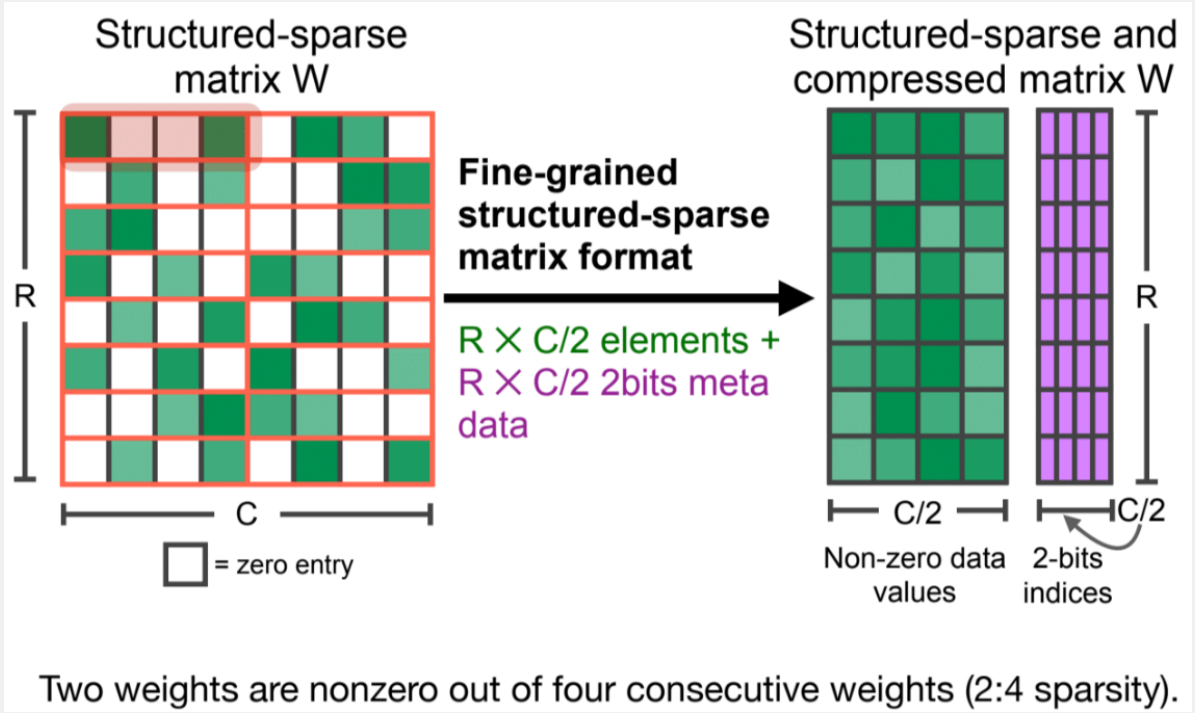
Pruning is now supported on Nvidia GPUs



2:4 sparsity in A100 GPU
2X peak performance, 1.5X measured BERT speedup

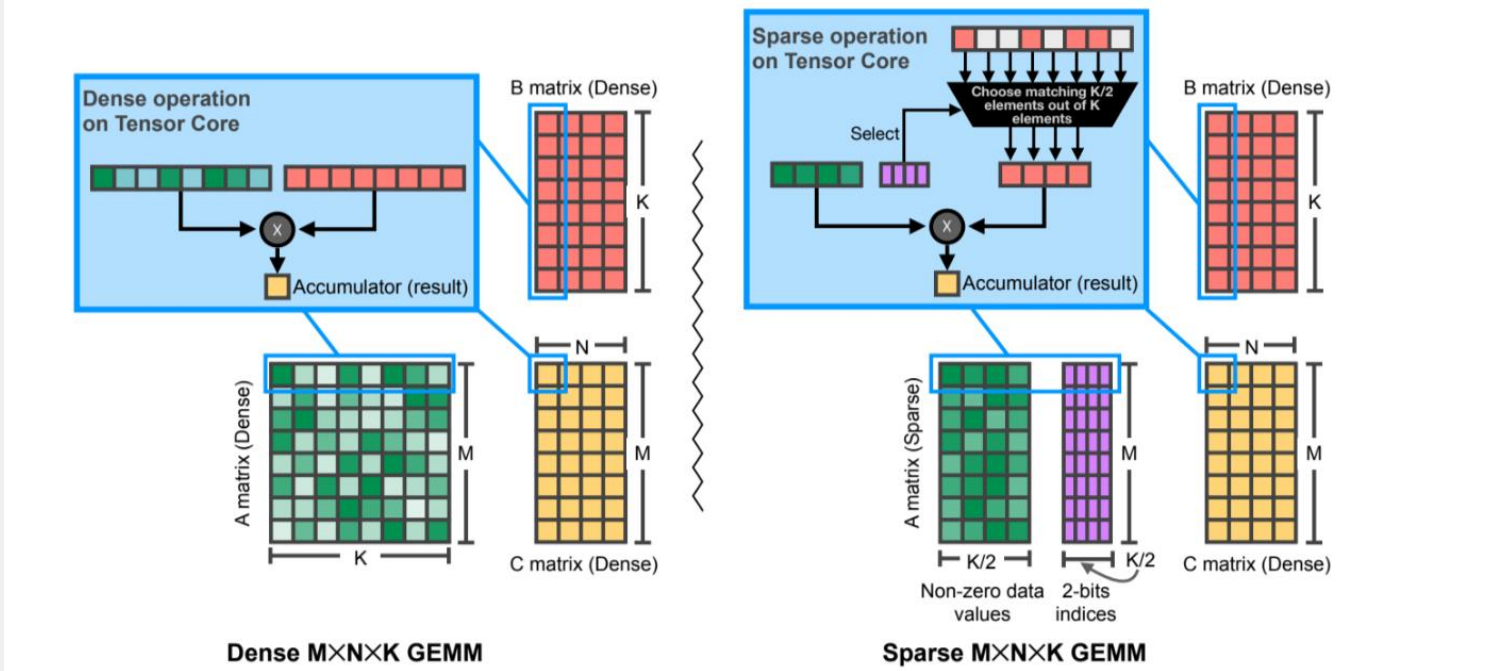
*Accelerating Sparse Deep Neural Networks [Mishra et al. ArXiv 2021]

M:N Sparsity support in Nvidia Tensor Cores



*Accelerating Sparse Deep Neural Networks [Mishraet al. ArXiv 2021]

Hardware support: Map M:N matrices to tensor cores



The indices are used to mask out the inputs. Only 2 multiplications will be done out of four.

Benchmarks on 2:4 Sparsity Support

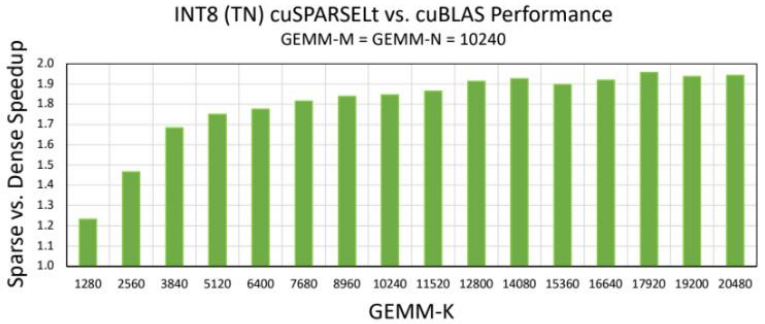


Fig. 3. Comparison of sparse and dense INT8 GEMMs on NVIDIA A100 Tensor Cores. Larger GEMMs achieve nearly a 2× speedup with Sparse Tensor Cores.

Network	Accuracy		
	Dense FP16	Sparse FP16	Sparse INT8
ResNet-34	73.7	73.9	73.7
ResNet-50	76.1	76.2	76.2
ResNet-50 (SWSL)	81.1	80.9	80.9
ResNet-101	77.7	78.0	77.9
ResNeXt-50-32x4	77.6	77.7	77.7
ResNeXt-101-32x16	79.7	79.9	79.9
ResNeXt-101-32x16 (WSL)	84.2	84.0	84.2
DenseNet-121	75.5	75.3	75.3
DenseNet-161	78.8	78.8	78.9
Wide ResNet-50	78.5	78.6	78.5
Wide ResNet-101	78.9	79.2	79.1
Inception v3	77.1	77.1	77.1
Xception	79.2	79.2	79.2
VGG-11	70.9	70.9	70.8
VGG-16	74.0	74.1	74.1
VGG-19	75.0	75.0	75.0
SUNet-128	75.6	76.0	75.4
SUNet-7-128	76.4	76.5	76.3
DRN26	75.2	75.3	75.3
DRN-105	79.4	79.5	79.4

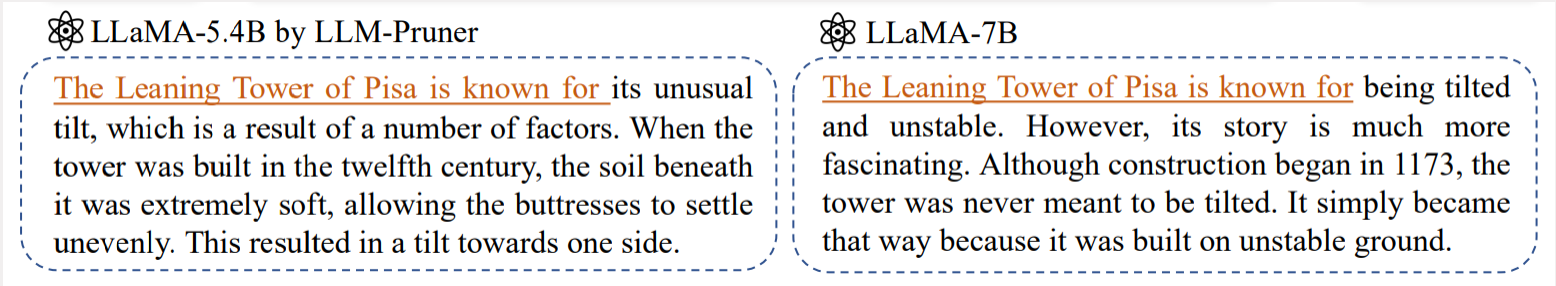
Pruning CNNs with 2:4 sparsity gives up to 2X speedup for GEMM and will not result in an accuracy drop

TU/e

Pruning is desired to deploy Large Language Models

Challenges:

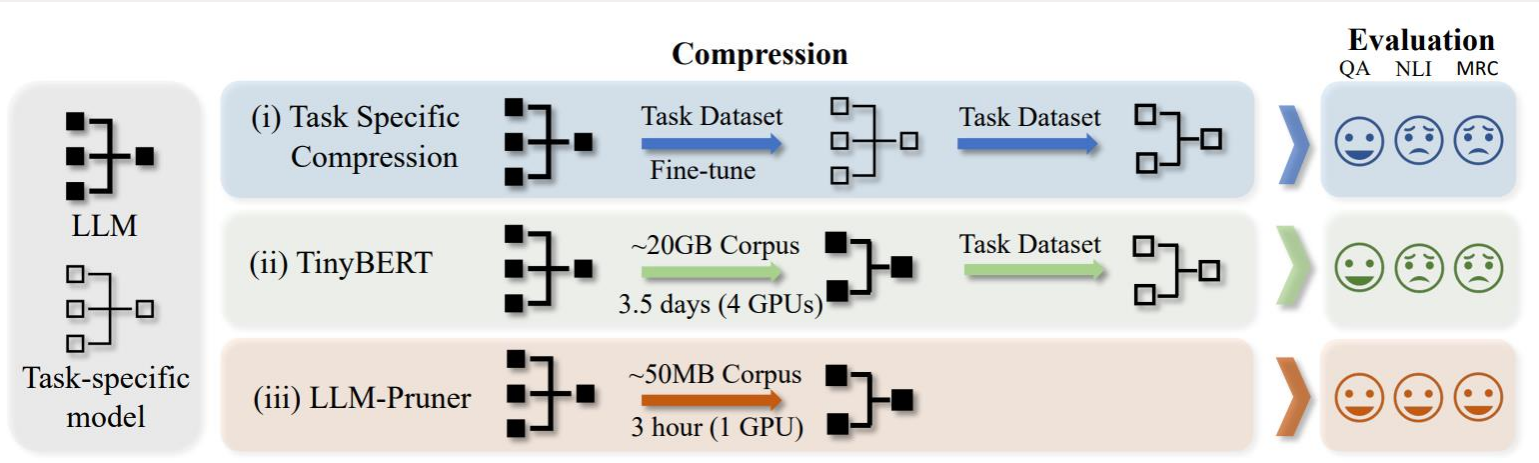
- The size of the training corpus of LLM is enormous
- Sometimes the training corpus is unavailable
- The unacceptably long duration for the post-training of the compressed LLM



X. Ma et al. “LLM-Pruner: On the Structural pruning of Large Language Models” NeurIPS 2023

What makes a good pruning method for LLM

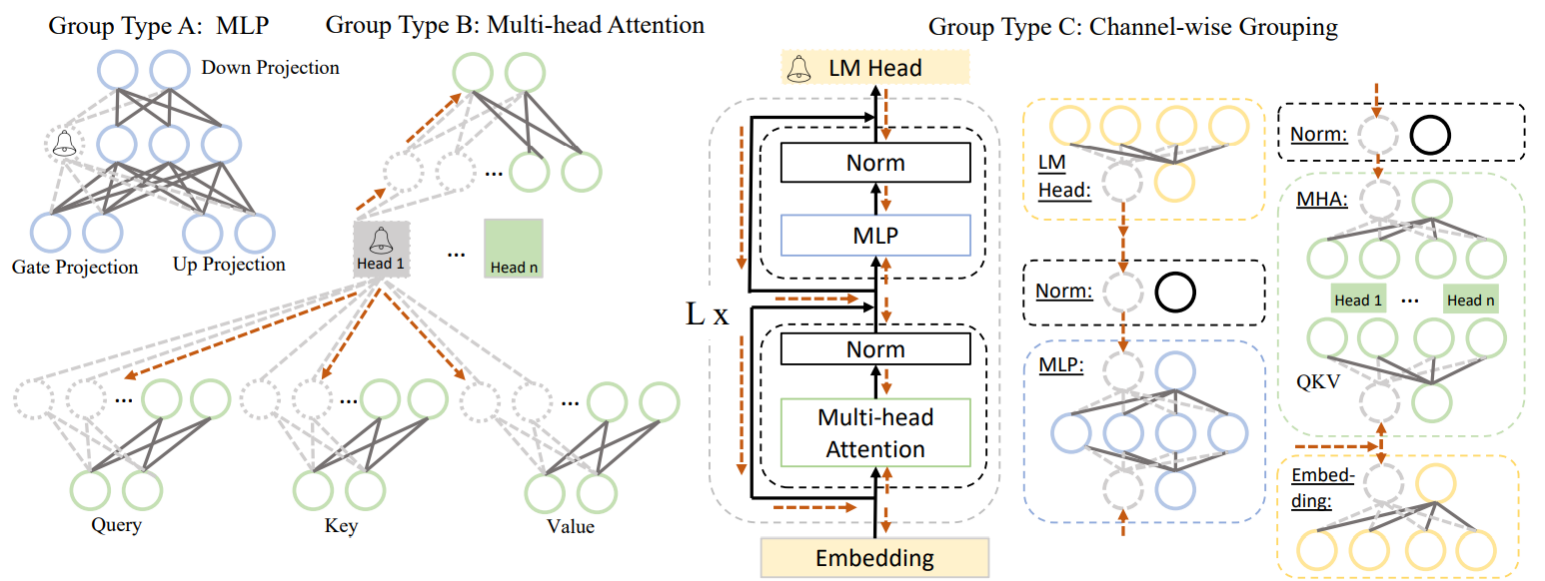
- Task-agnostic compression: LLM should retain its original ability as multi-task solver
- Less training corpus: E.g. only 50k public available samples to post-train
- Efficient compression afterwards
- Automatic approach: Minimal human effort required



Define Dependency Groups and Estimate Importance

Only make pruning decisions based upon dependency groups

Use retraining to repair functionality



Results on LLM pruning

Good results while reducing 20% of the parameters

The domain of LLM pruning is in developing quickly: All datacenters want pruned networks

Pruning Ratio	Method	WikiText2↓	PTB↓	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Ratio = 0%	LLaMA-7B[49]	-	-	76.5	79.8	76.1	70.1	72.8	47.6	57.2	68.59
	LLaMA-7B*	12.62	22.14	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
Ratio = 20% w/o tune	L2	582.41	1022.17	59.66	58.00	37.04	52.41	33.12	28.58	29.80	42.65
	Random	27.51	43.19	61.83	71.33	56.26	54.46	57.07	32.85	35.00	52.69
	Channel	74.63	153.75	62.75	62.73	41.40	51.07	41.38	27.90	30.40	45.38
	Vector	22.28	41.78	61.44	71.71	57.27	54.22	55.77	33.96	38.40	53.25
	Element^2	19.77	36.66	59.39	75.57	65.34	61.33	59.18	37.12	39.80	56.82
	Element^1	19.09	34.21	57.06	75.68	66.80	59.83	60.94	36.52	**40.00**	56.69
Ratio = 20% w/ tune	Channel	22.02	38.67	59.08	73.39	64.02	60.54	57.95	35.58	38.40	55.57
	Vector	18.84	33.05	65.75	74.70	64.52	59.35	60.65	36.26	39.40	57.23
	Element^2	**17.37**	30.39	**69.54**	76.44	68.11	**65.11**	63.43	**37.88**	**40.00**	**60.07**
	Element^1	17.58	**30.11**	64.62	**77.20**	**68.80**	63.14	**64.31**	36.77	39.80	59.23

TU/e

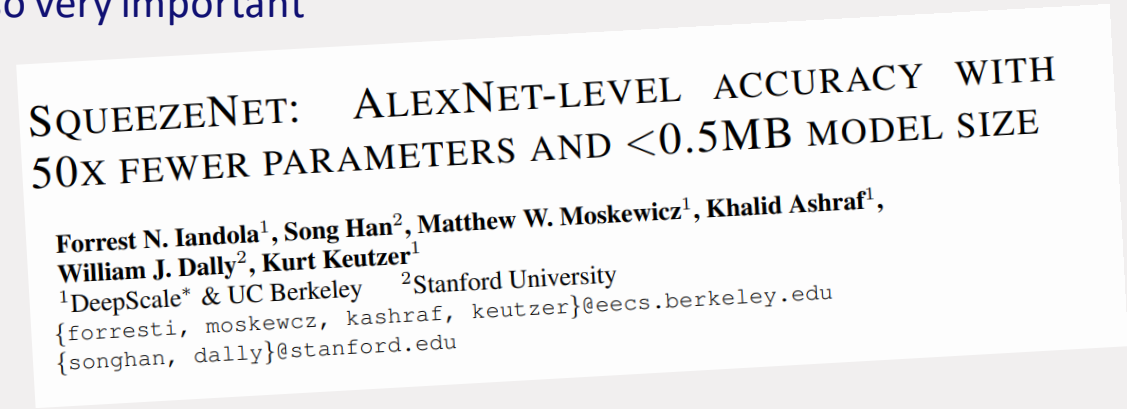
Summary: Pruning

- Networks with many parameters are not efficient: Storage, Speed, Energy
- Data-Free pruning: up to 1.5x reduction
- L1 regularization: up to 4x reduction
- Retraining: up to 6.7x reduction
- Iterative retraining up to 10x reduction
- Speed and energy advantages are lower 3 and 6x due to irregularity
- Use a model for Sparsity
- Activation Sparsity
- Structured pruning can help
  - Less storage overhead and much more regular
  - Architecture dependent

Conclusions

- Pruning does bring a lot to the complete tool box of optimizations
  - Pruning is not the only tool and it has disadvantages (irregularity)
  - It does not work well for all networks and layers!!!

- Correct Network design is also very important
  - Plus Quantization
  - Plus Weight sharing



SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

Forrest N. Iandola[1], Song Han[2], Matthew W. Moskewicz[1], Khalid Ashraf[1], William J. Dally[2], Kurt Keutzer[1]
[1]DeepScale* & UC Berkeley [2]Stanford University
{forresti, moskewcz, kashraf, keutzer}@eecs.berkeley.edu
{songhan, dally}@stanford.edu

TU/e