Pruning

Chia-Chi Tsai (蔡家齊)
cctsai@gs.ncku.edu.tw
AI System Lab
Department of Electrical Engineering
National Cheng Kung University

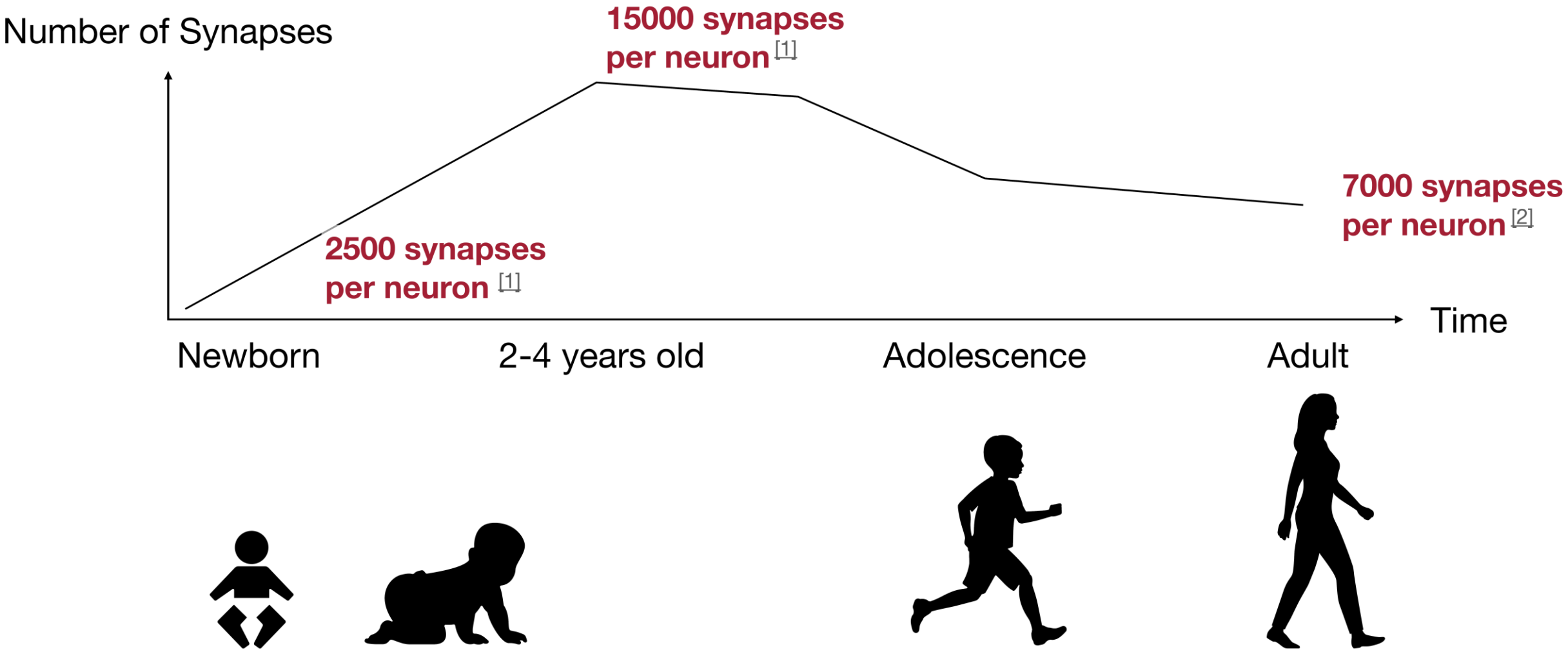
Outline

- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

Outline

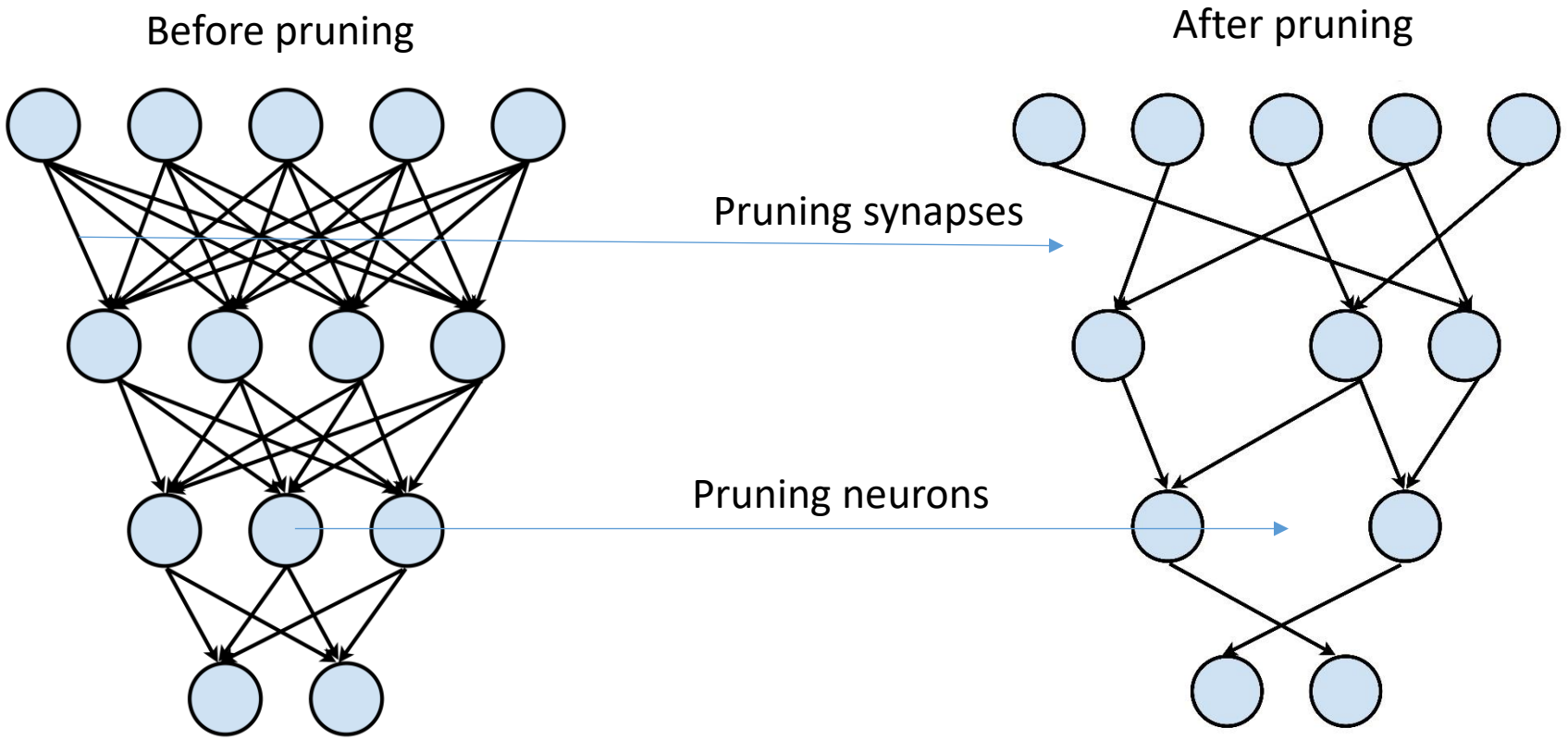
- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

Pruning Happens in Human Brain



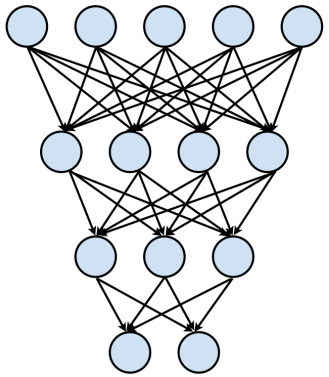
Neural Network Pruning

- Make neural network smaller by removing synapses and neurons

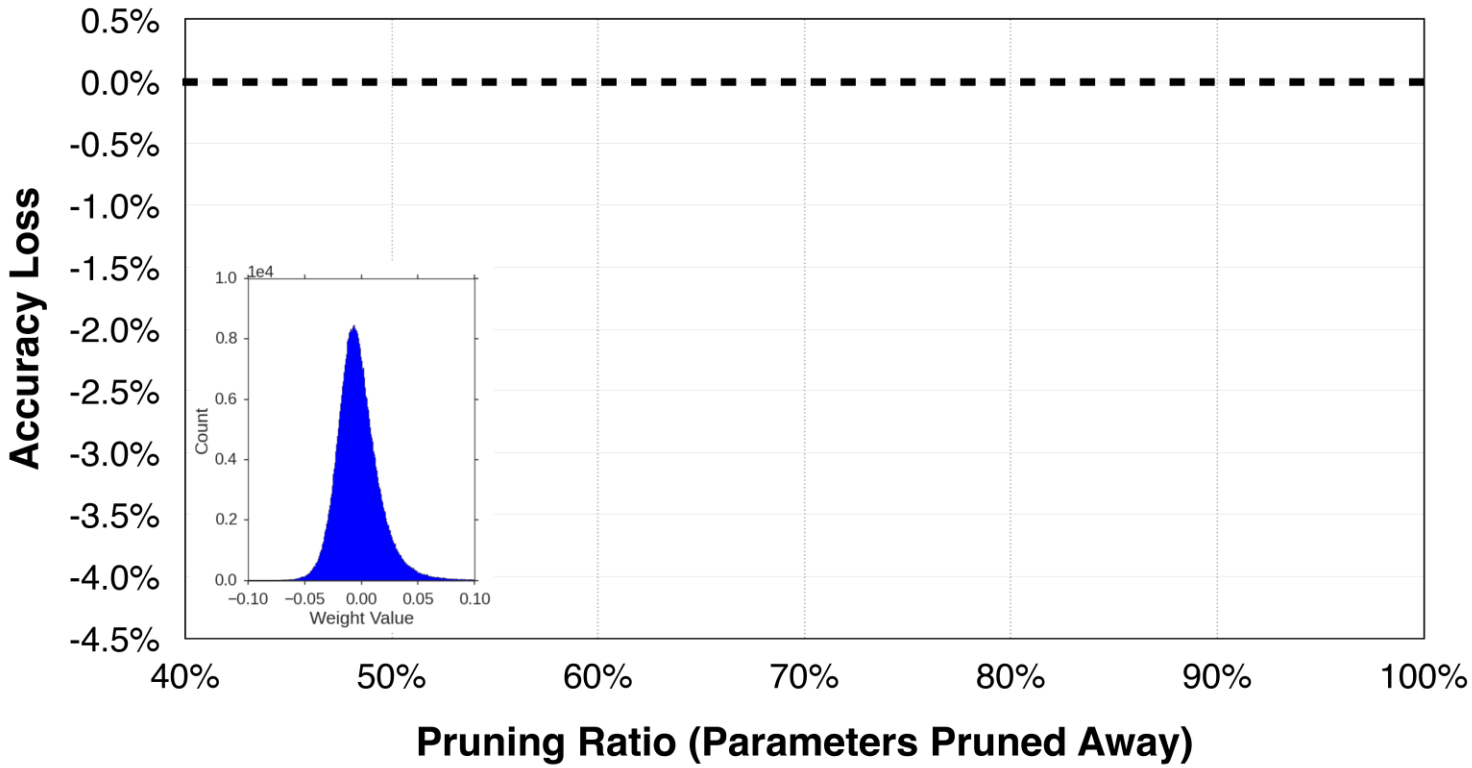


Neural Network Pruning

- Make neural network smaller by removing synapses and neurons

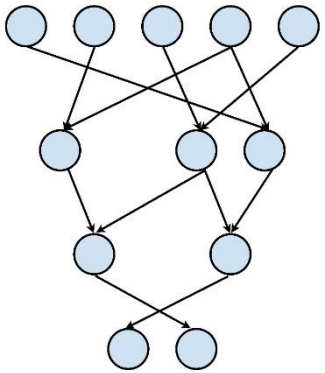


Train Connectivity



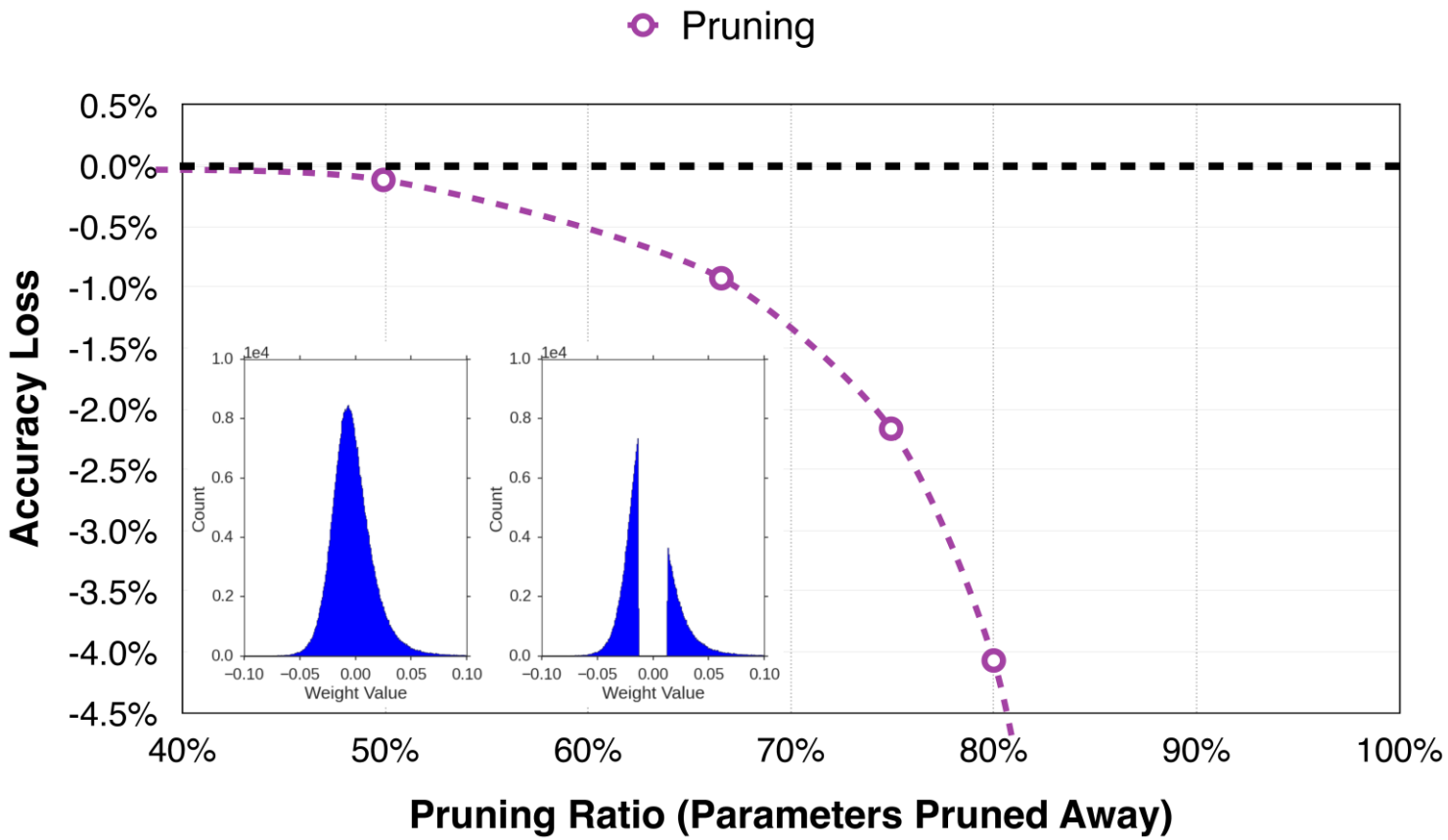
Neural Network Pruning

- Make neural network smaller by removing synapses and neurons



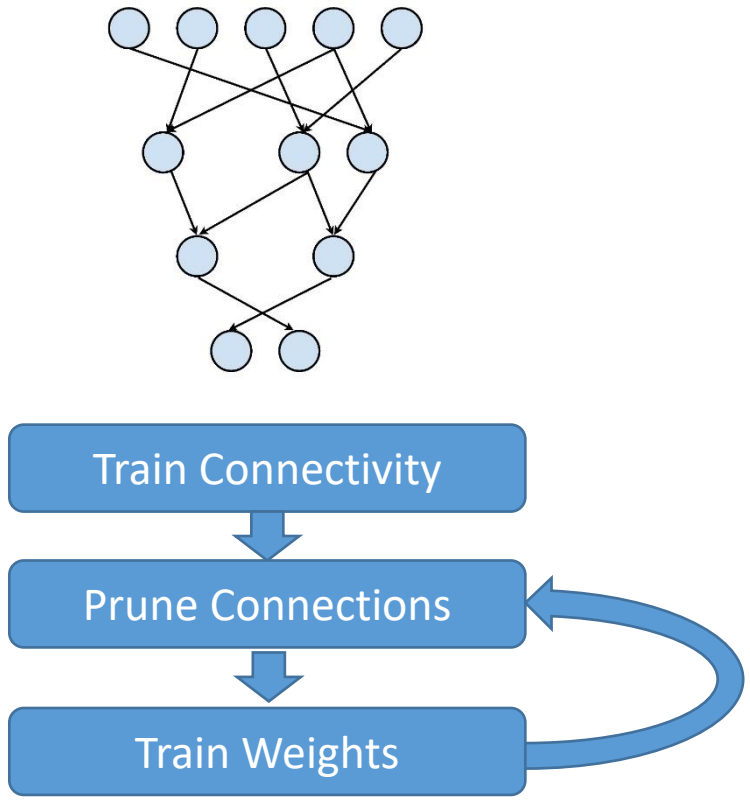
Train Connectivity

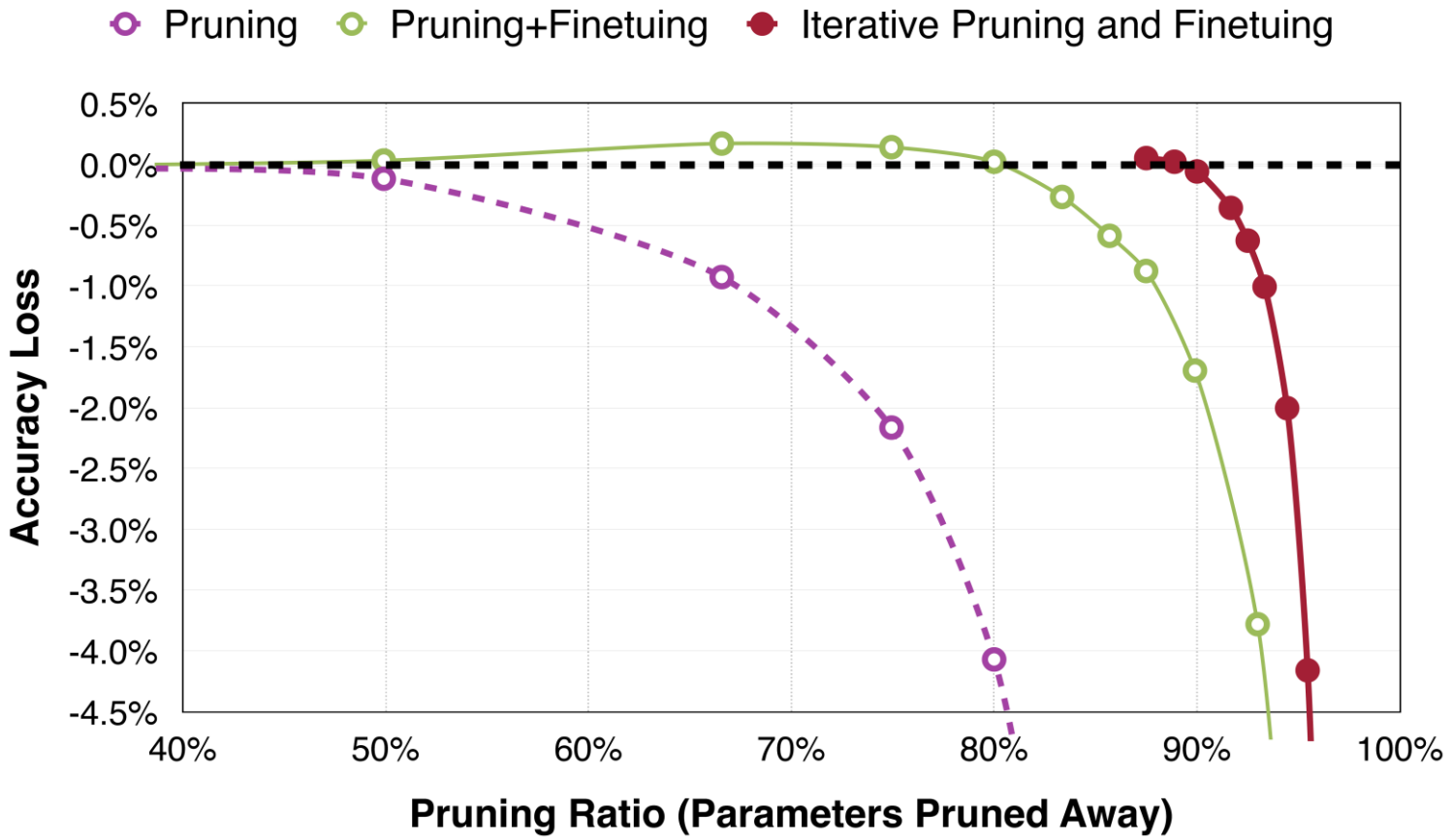
Prune Connections



Neural Network Pruning

- Make neural network smaller by removing synapses and neurons





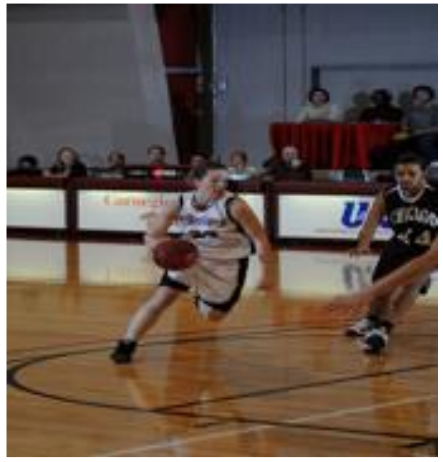
Pruning CNNs

- Make neural network smaller by removing synapses and neurons

Model	Number of Parameters			MACs
	Before Pruning (M)	After Pruning (M)	Reduction (times)	Reduction (times)
Alexnet	61	6.7	9	3
VGG16	138	10.3	12	5
GoogLeNet	7	2.0	3.5	5
ResNet50	26	7.47	3.4	6.3
SqueezeNet	1	0.38	3.2	3.5

Pruning LSTM Model

- Pruning the NeuralTalk LSTM does not hurt image caption quality



Baseline: a basketball player in a white uniform is playing with a ball.

Pruned 90%: a basketball player in a white uniform is playing with a basketball.

Baseline: a brown dog is running through a grassy field.

Pruned 90%: a brown dog is running through a grassy area.

Baseline: a man is riding a surfboard on a wave.

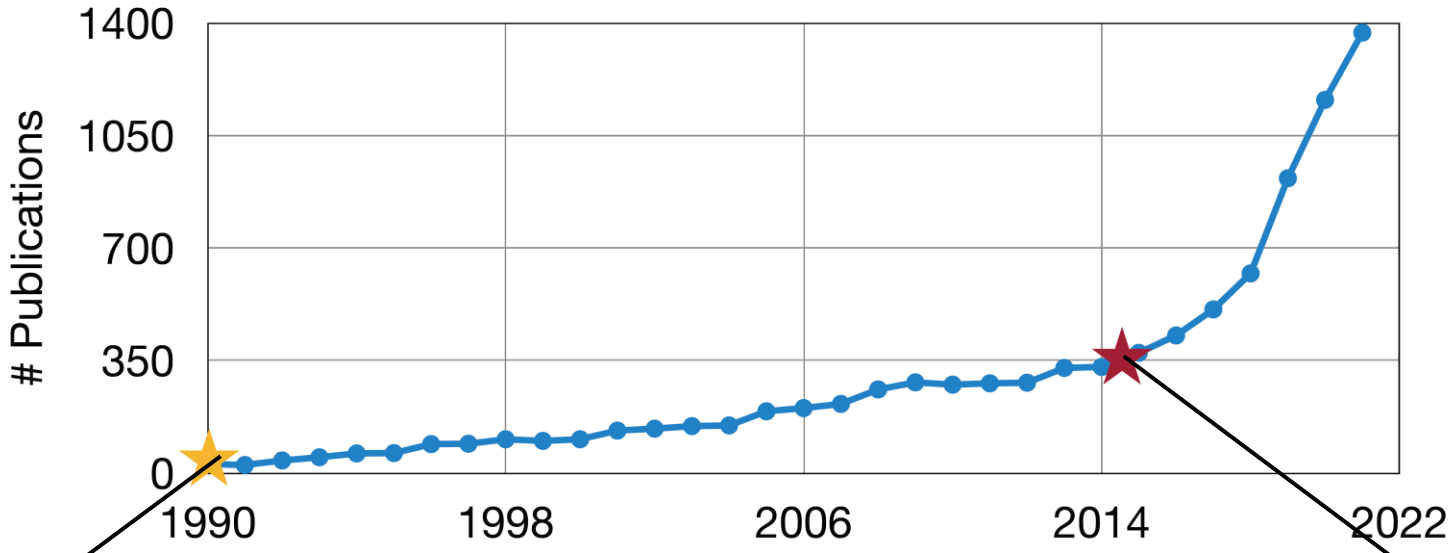
Pruned 90%: a man in a wetsuit is riding a wave on a beach.

Baseline: a soccer player in red is running in the field.

Pruned 95%: a man in a red shirt and black and white black shirt is running through a field.

Trends of Pruning

- Make neural network smaller by removing synapses and neurons



Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

Learning both Weights and Connections for Efficient Neural Networks

Song Han
Stanford University
songhan@stanford.edu

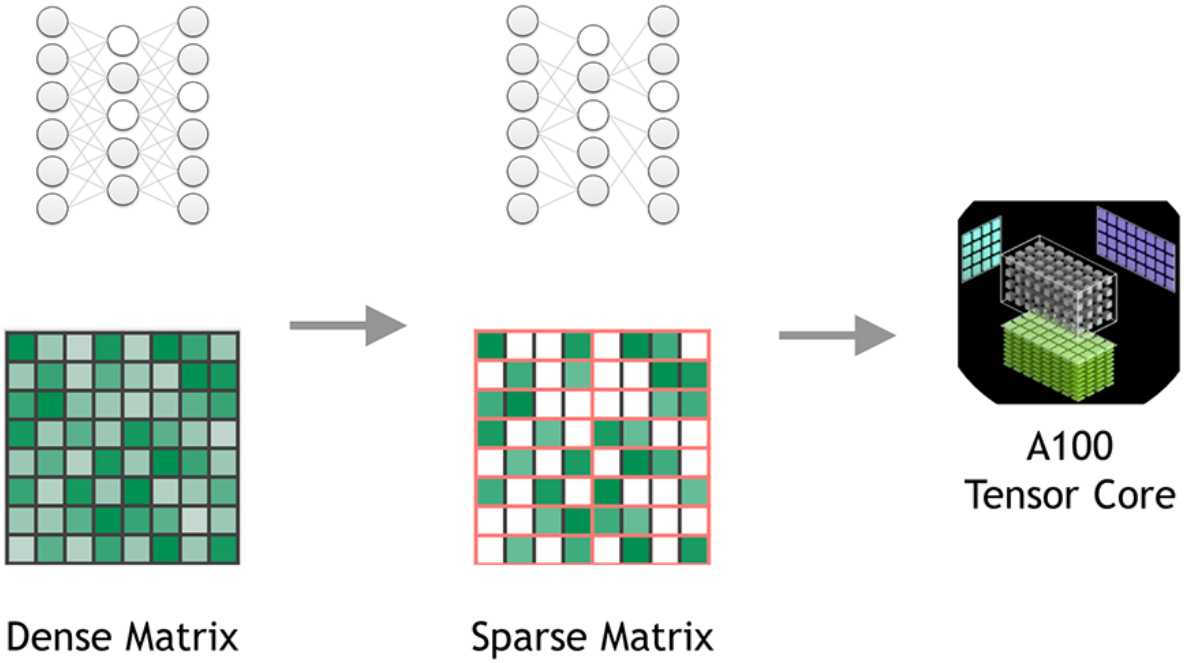
Jeff Pool
NVIDIA
jpool@nvidia.com

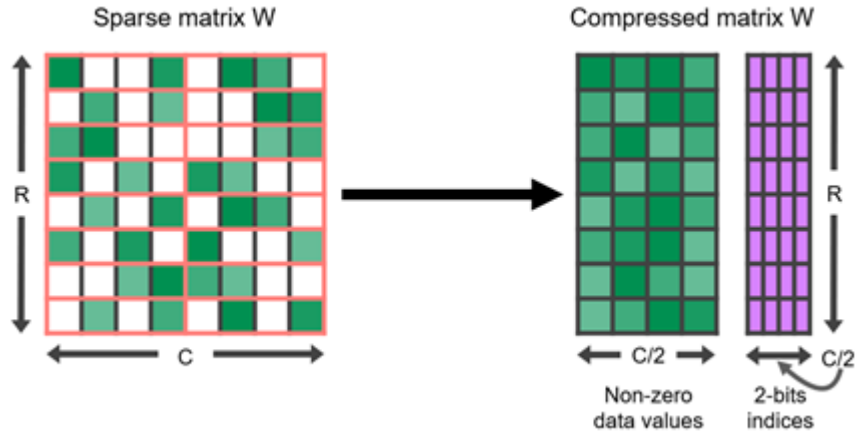
John Tran
NVIDIA
johntran@nvidia.com

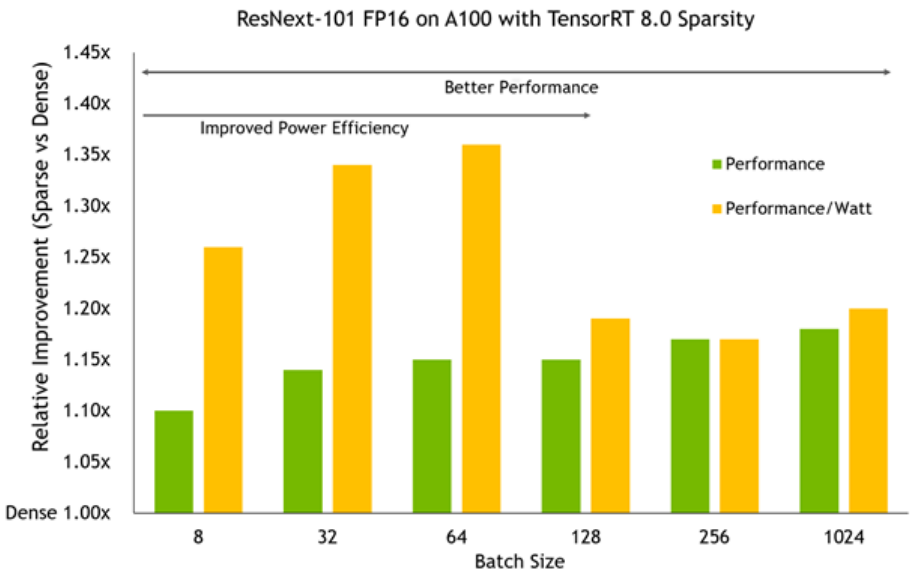
William J. Dally
Stanford University
NVIDIA
dally@stanford.edu

Hardware Support for Sparsity

- NVIDIA sparse tensor cores

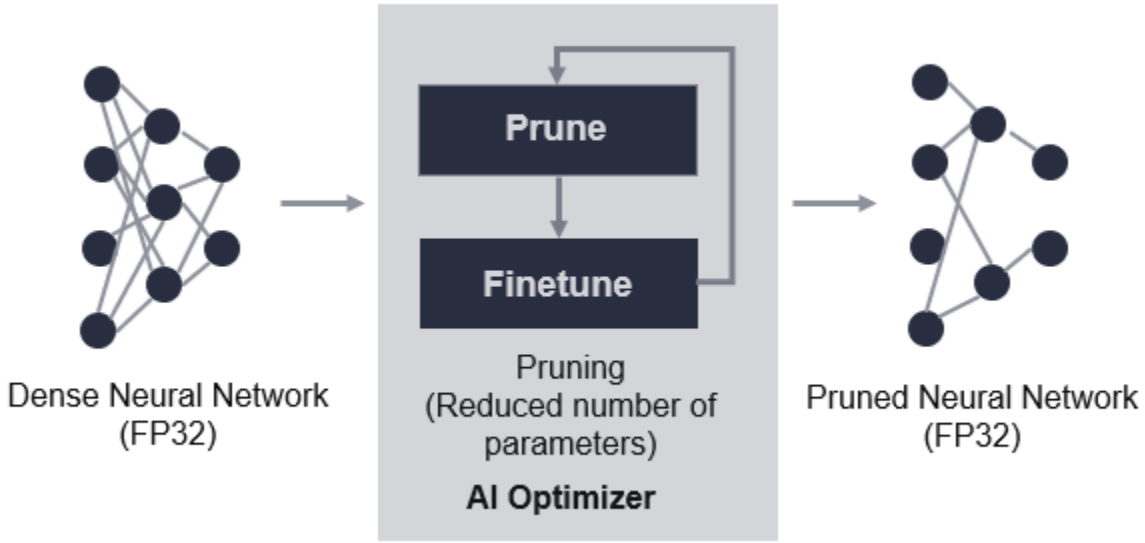


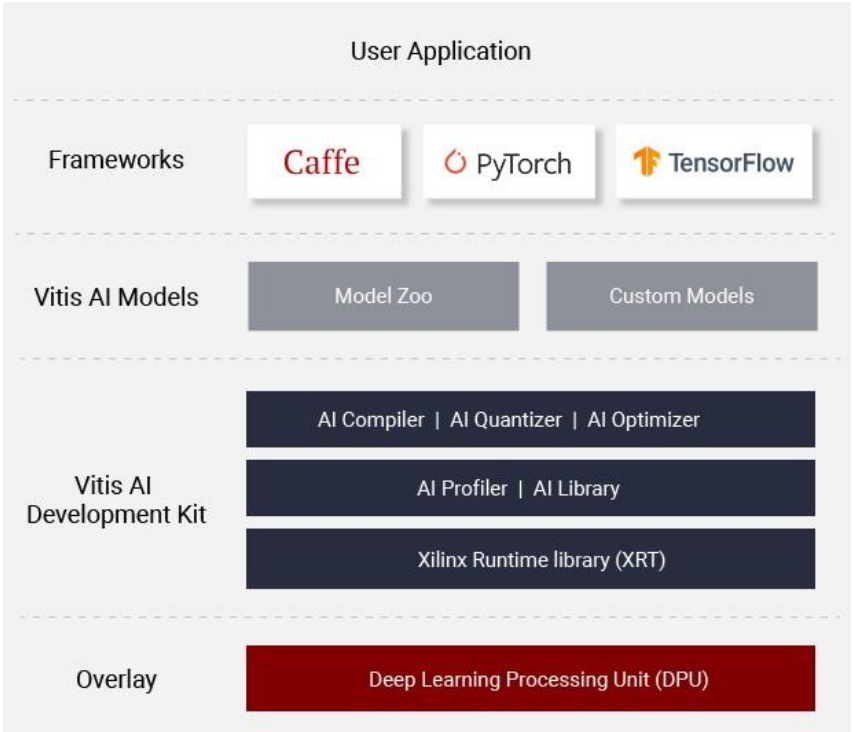




Hardware Support for Sparsity

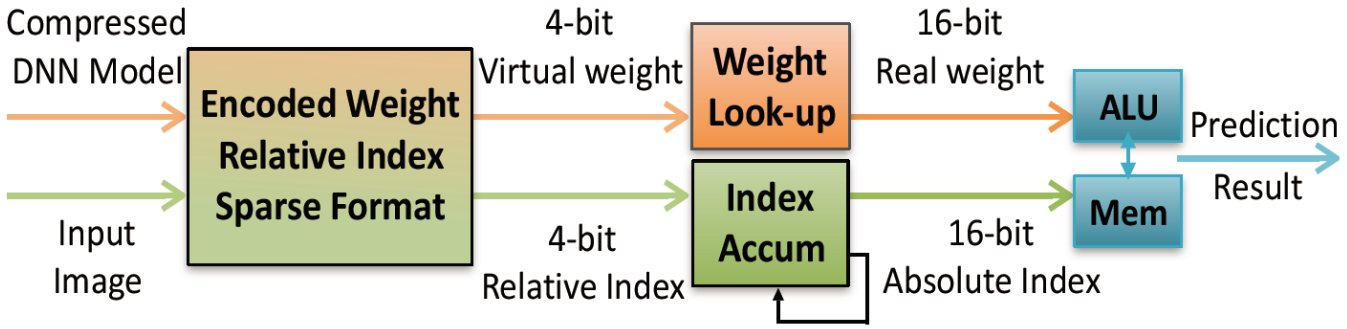
- Xilinx Vitis AI Optimizer
  - Reduce model complexity by 5x to 50x with minimal accuracy impact

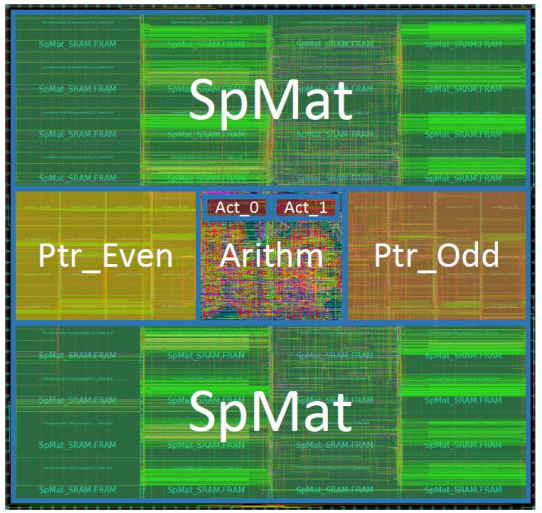


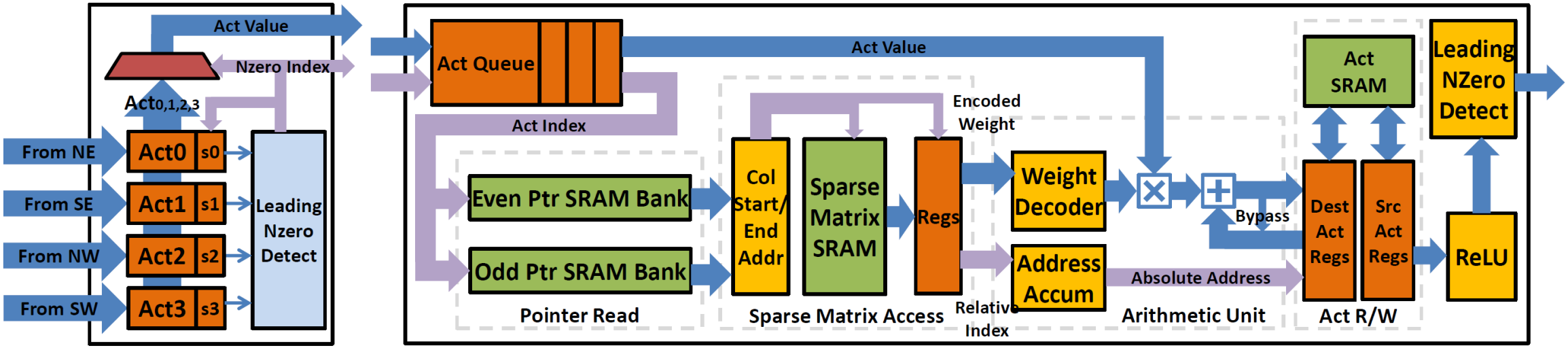


Sparse Accelerator for Images

- Efficient Inference Engine (EIE)



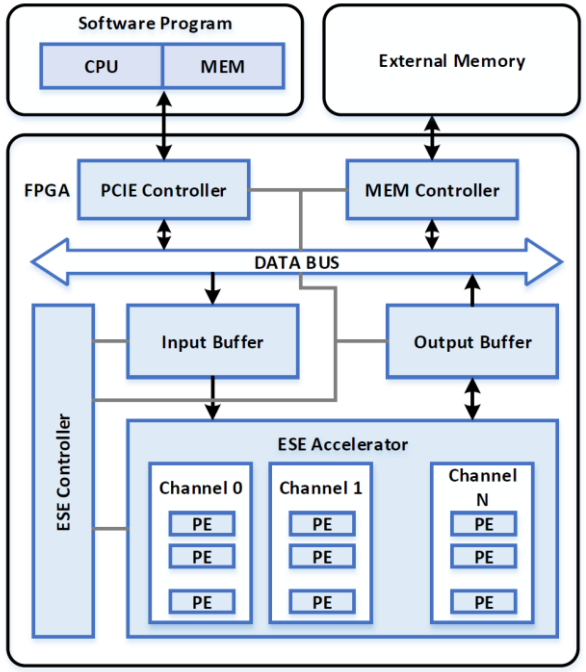


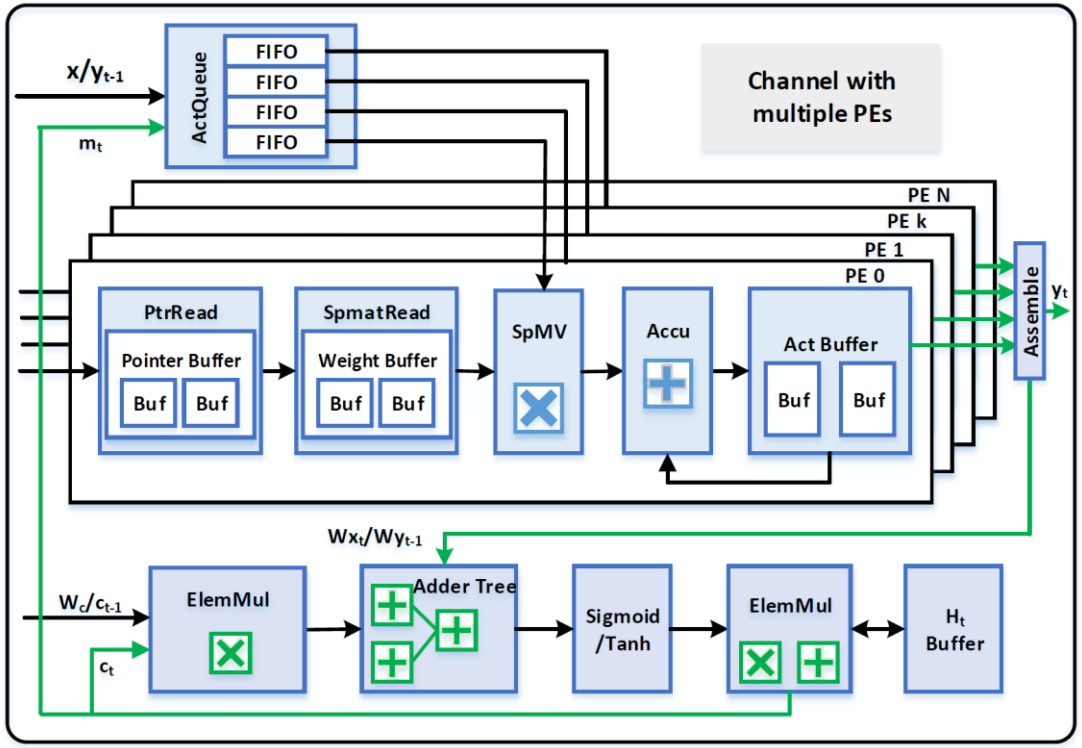


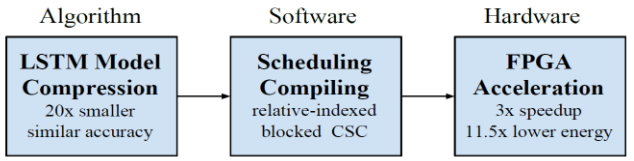
Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., & Dally, W. J. (2016). EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, *44*(3), 243-254.

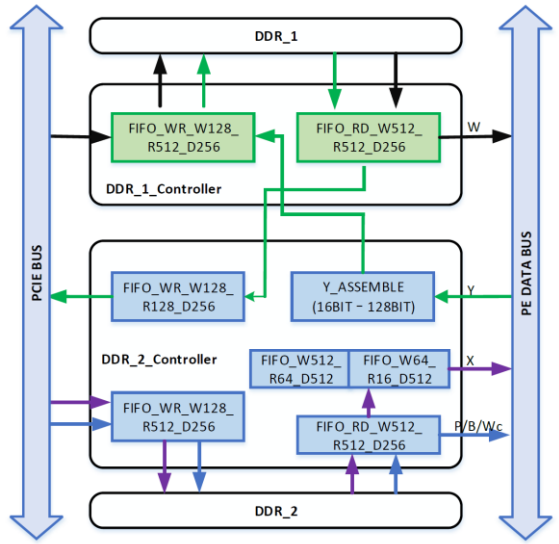
Sparse Accelerator for LSTM

- Efficient Speech Recognition Engine (ESE)





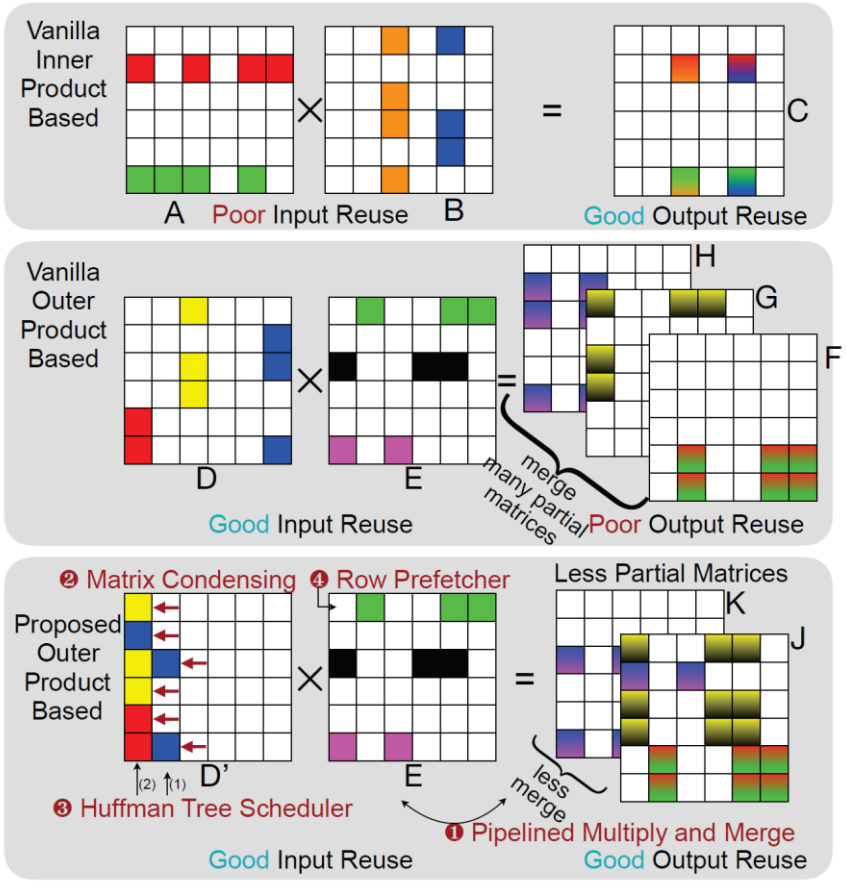


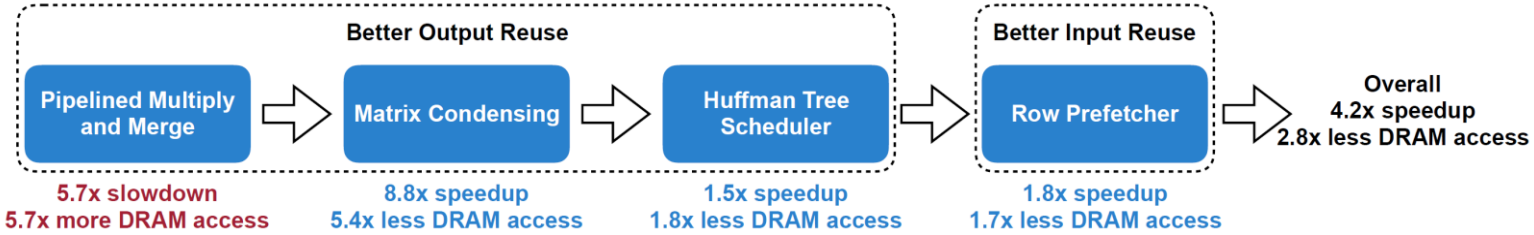


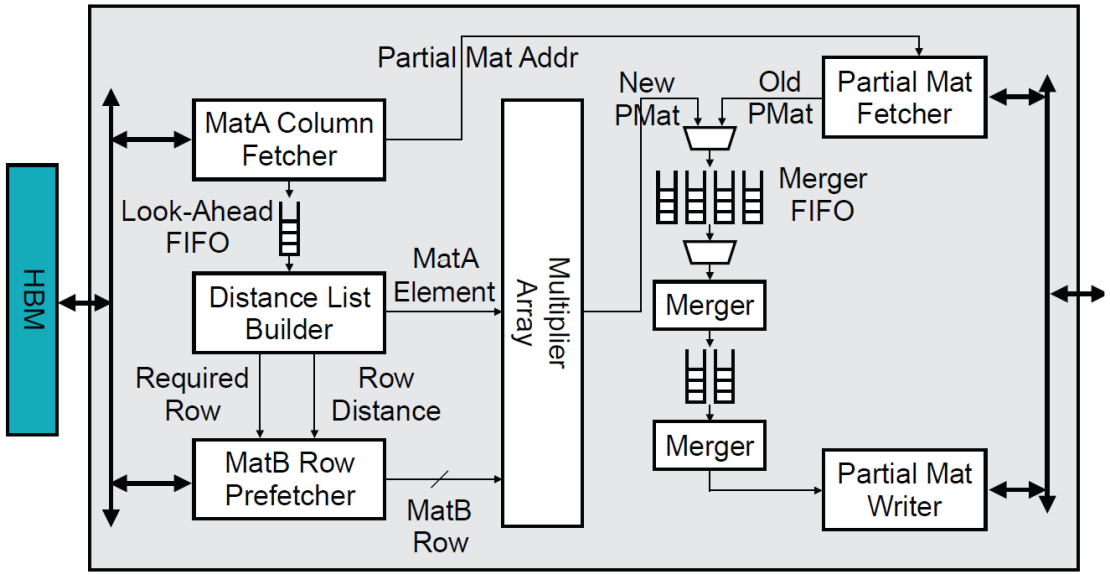
Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., ... & Dally, W. B. J. (2017, February). Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 75-84).

Sparse Accelerator for GEMM

- SpArch



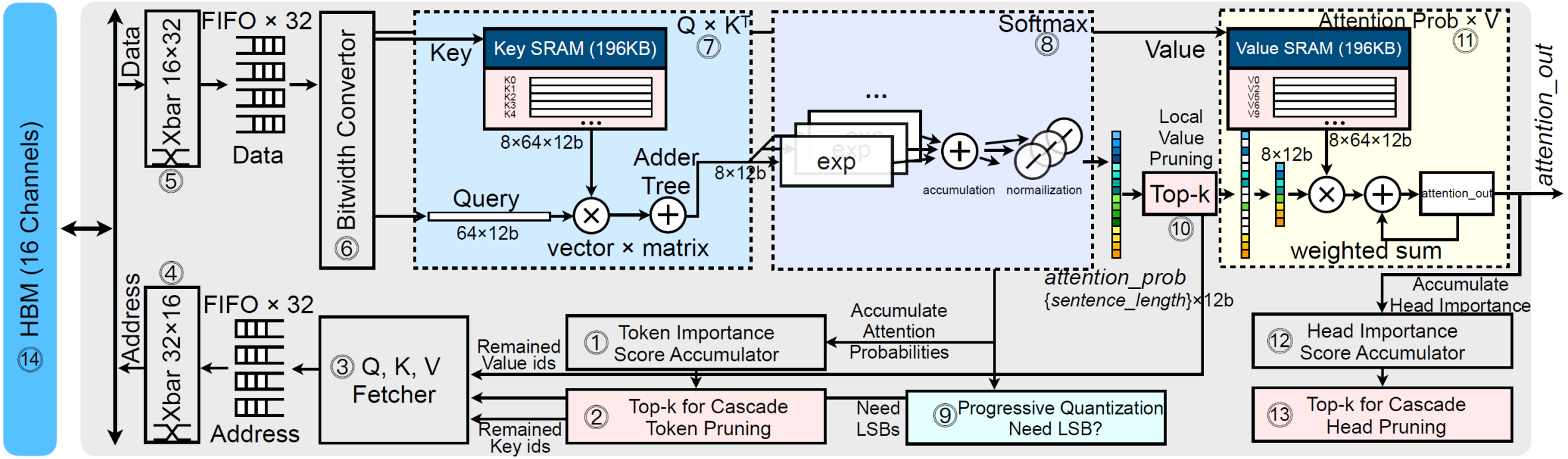




Zhang, Z., Wang, H., Han, S., & Dally, W. J. (2020, February). Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 261-274). IEEE.

Sparse Accelerator for Attention

- SpAtten



Wang, H., Zhang, Z., & Han, S. (2021, February). Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (pp. 97-110). IEEE.

Neural Network Pruning

- In what pattern should we prune the neural network?
  - **Determine the Pruning Granularity**
- What synapses/neurons should we prune?
  - **Determine the Pruning Criterion**
- What should target sparsity be for each layer?
  - **Determine the Pruning Ratio**
- How should we improve performance of pruned models?
  - **Fine-tune/Train Pruned Neural Network**

Pruning Formulation

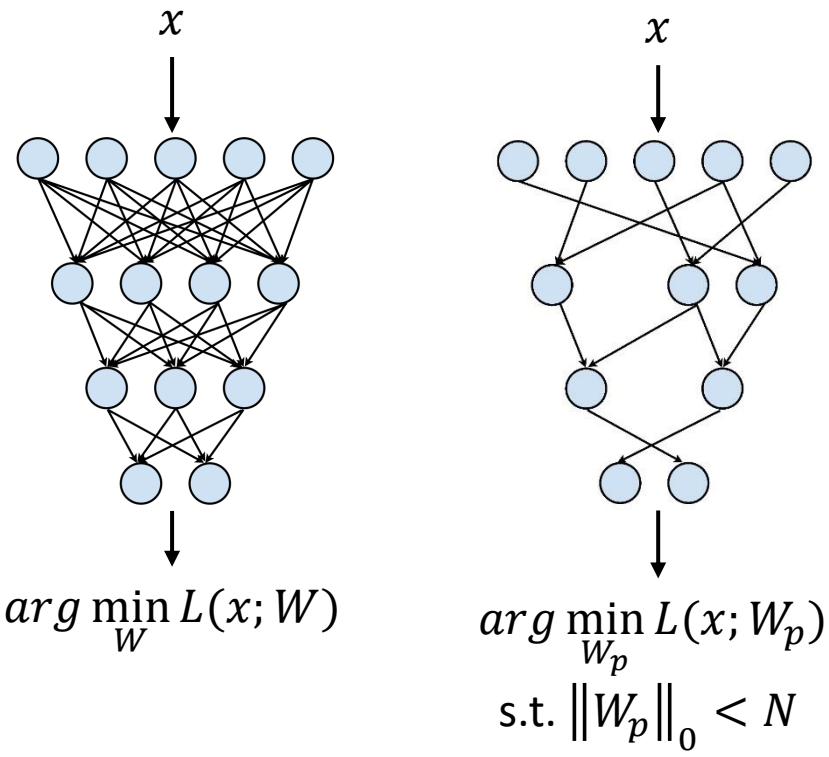
- In general, we could formulate the pruning as follows

$$arg\min_{W_p} L(x; W_p)$$

  - Subject to

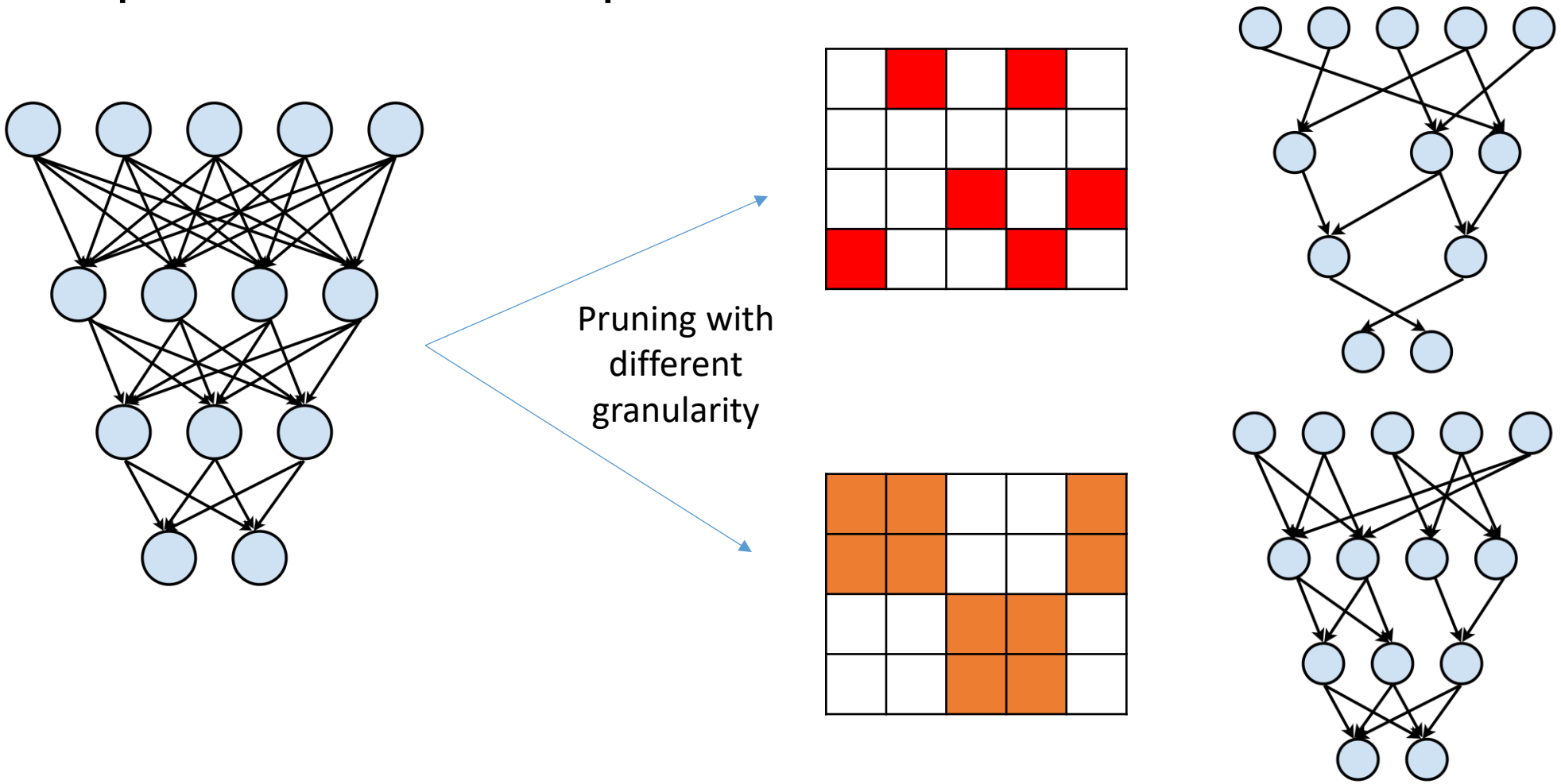
$$\|W_p\|_0 < N$$

  - L: objective function for neural network training:
  - x: input, W: original weights, W_p: pruned weights
  - $\|W_p\|_0$ calculates the number of non-zeroes in W_p
  - N: the target number of non-zeroes



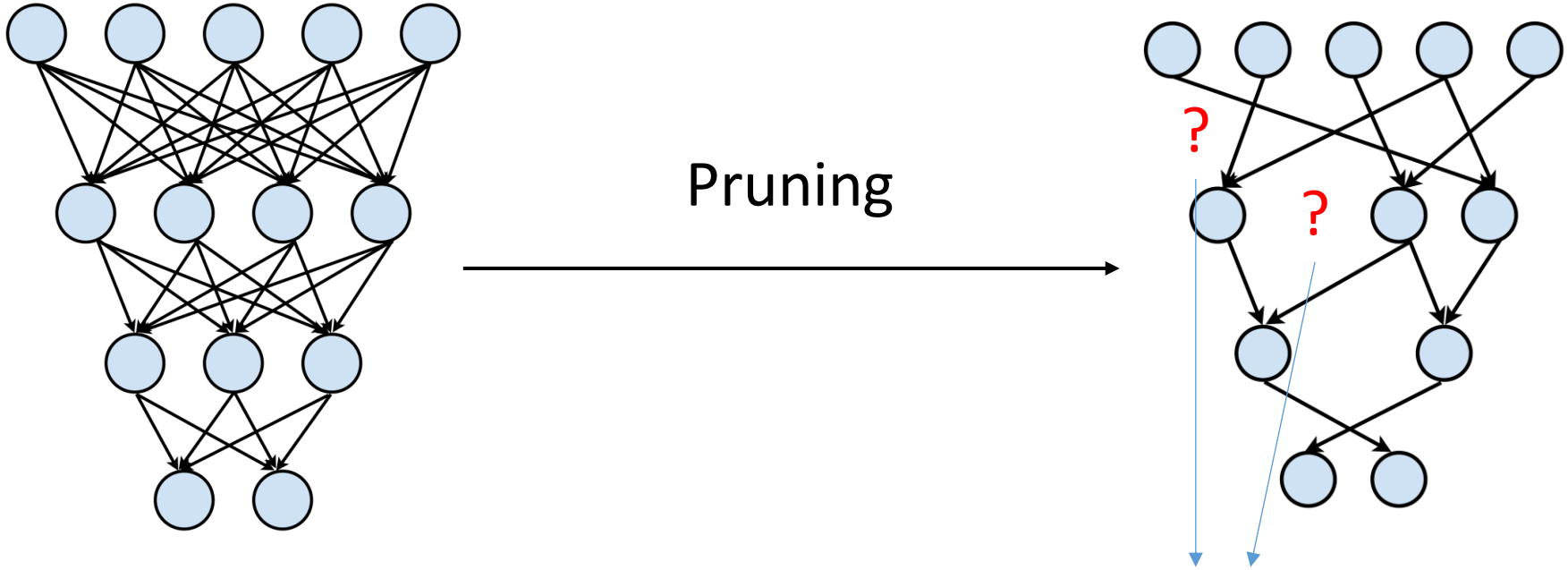
Determine the Pruning Granularity

- In what pattern should we prune the neural network?



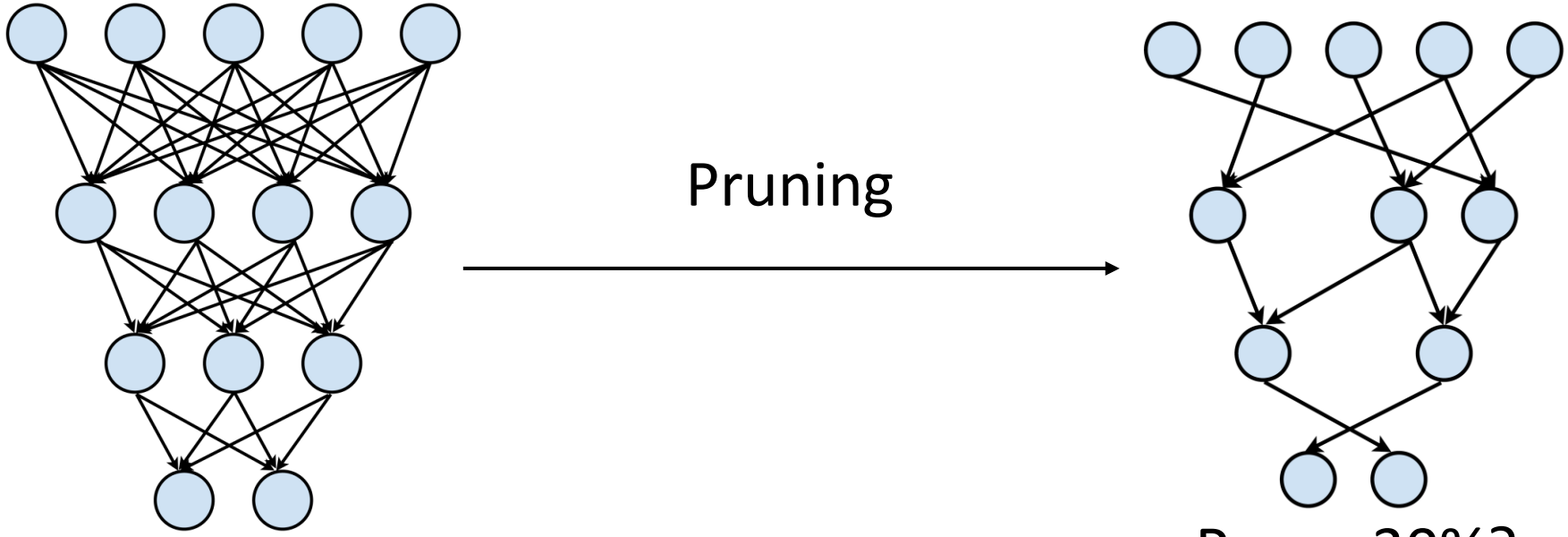
Determine the Pruning Criterion

- What synapses/neurons should we prune?



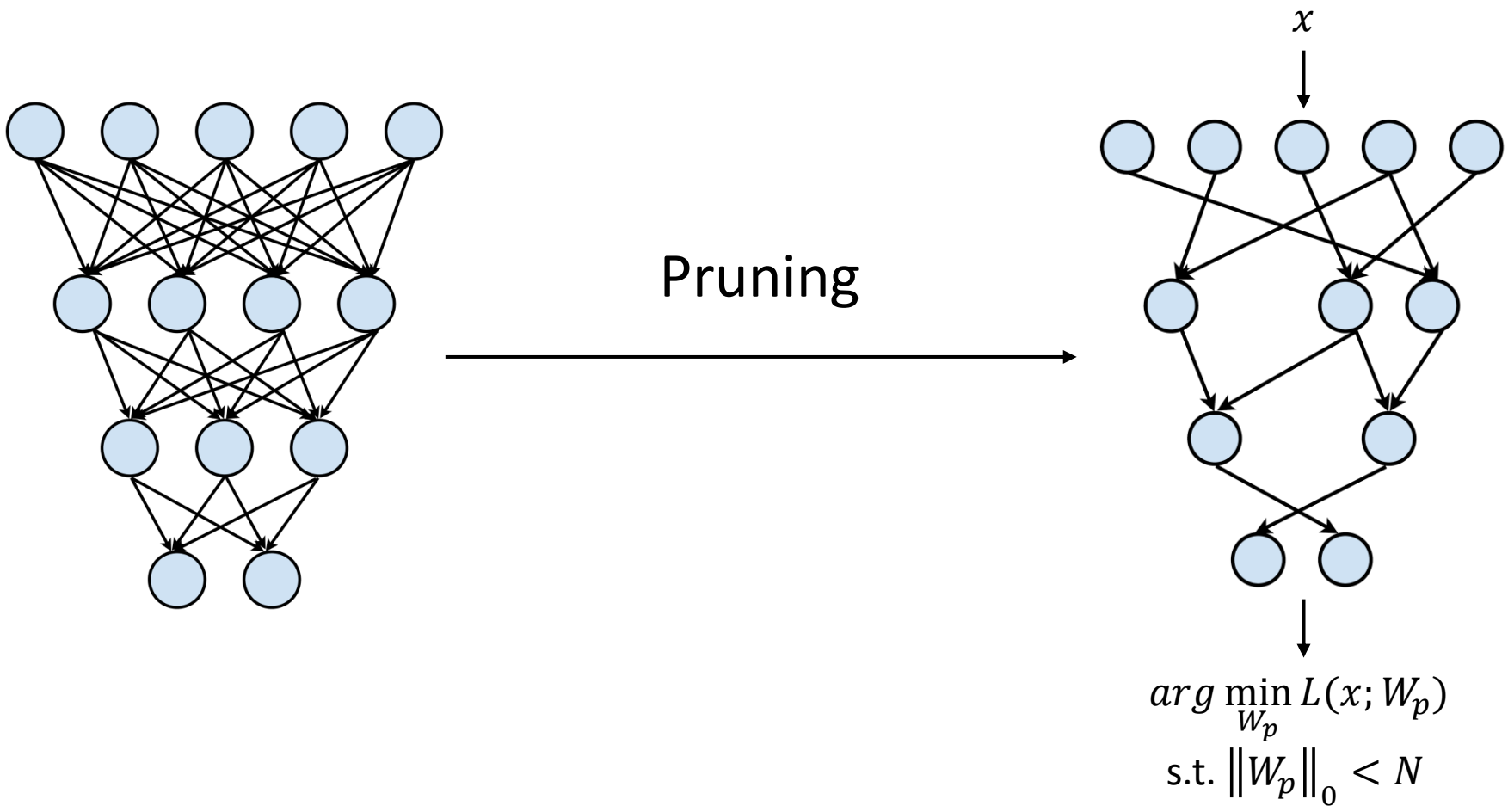
Determine the Pruning Ratio

- What should target sparsity be for each layer?



Fine-tune/Train Pruned Neural Network

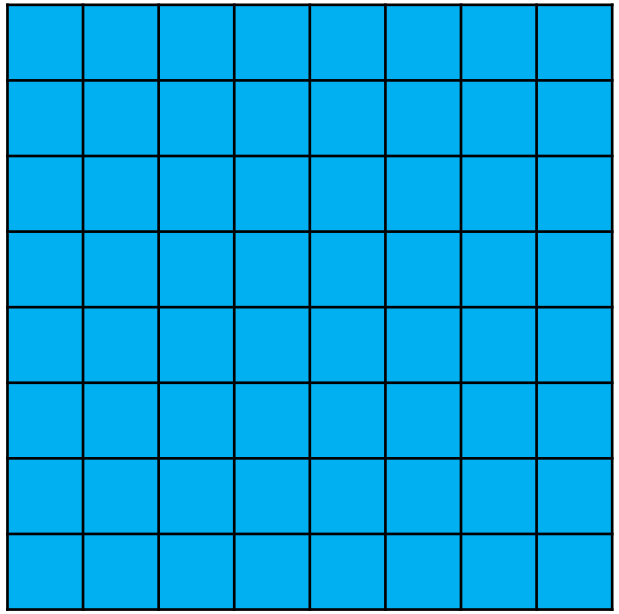
- How should we improve performance of pruned models?



Outline

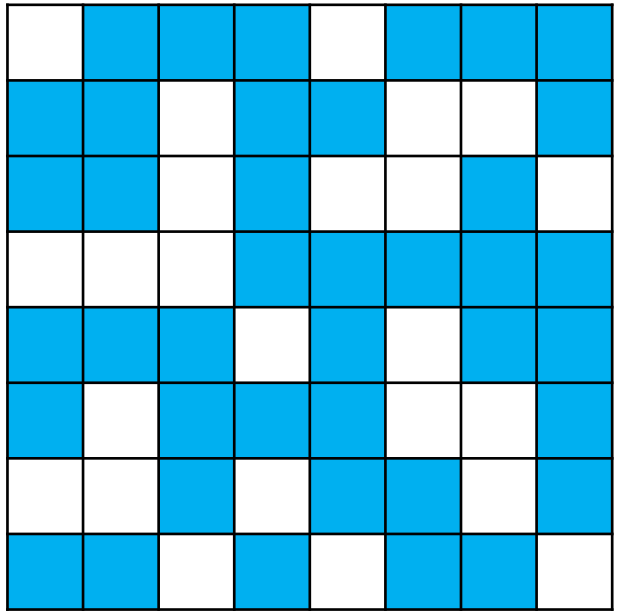
- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

A Simple Example of 2D Weight Matrix



- Pruning can be performed at different granularities
  - From structured to non-structured

A Simple Example of 2D Weight Matrix



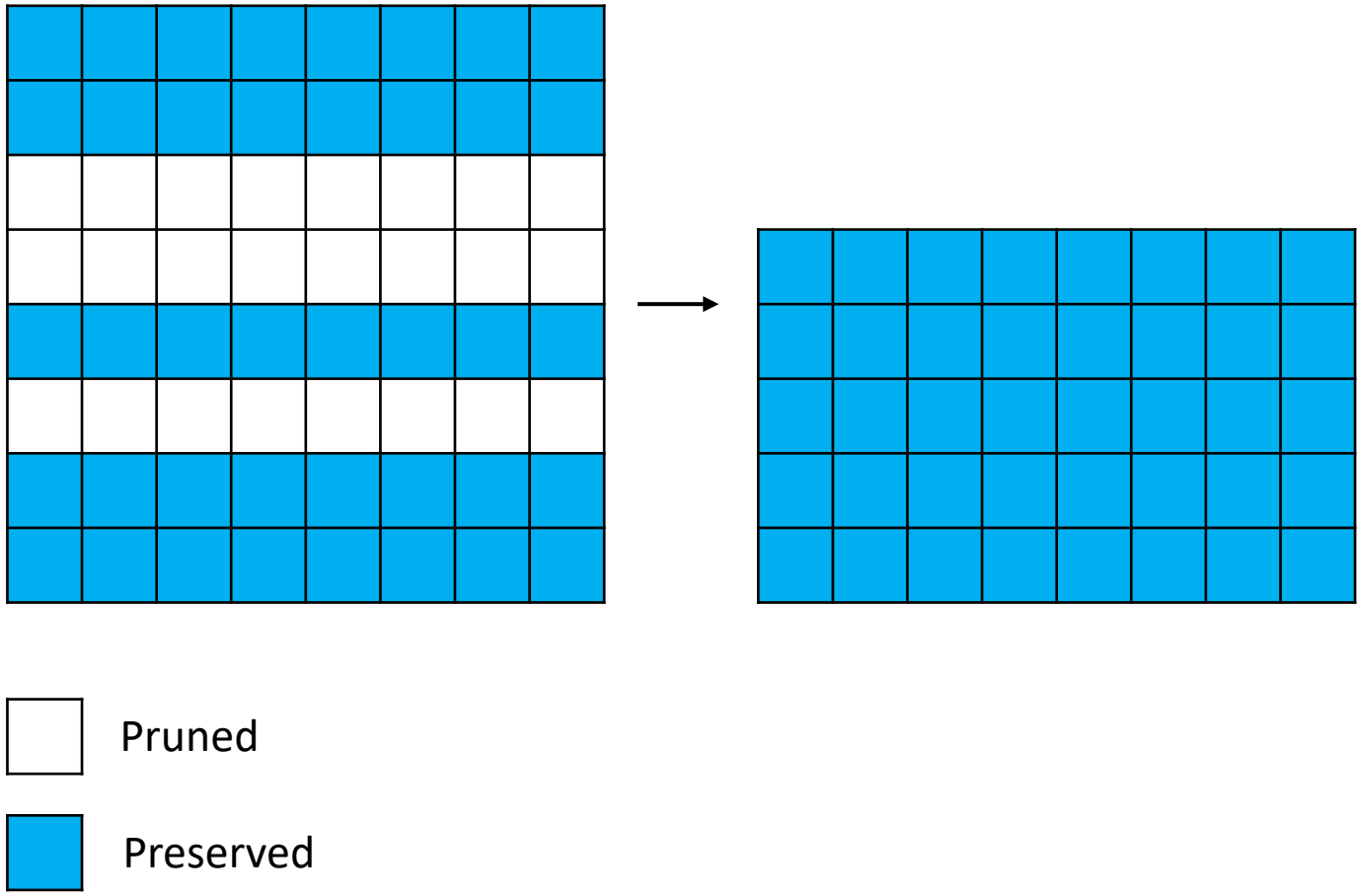
Pruned

Preserved

Fine-grained/Unstructured

- More flexible pruning index choice
- Hard to accelerate (irregular data expression)
- Need specialized hardware

A Simple Example of 2D Weight Matrix



Coarse-grained/Structured

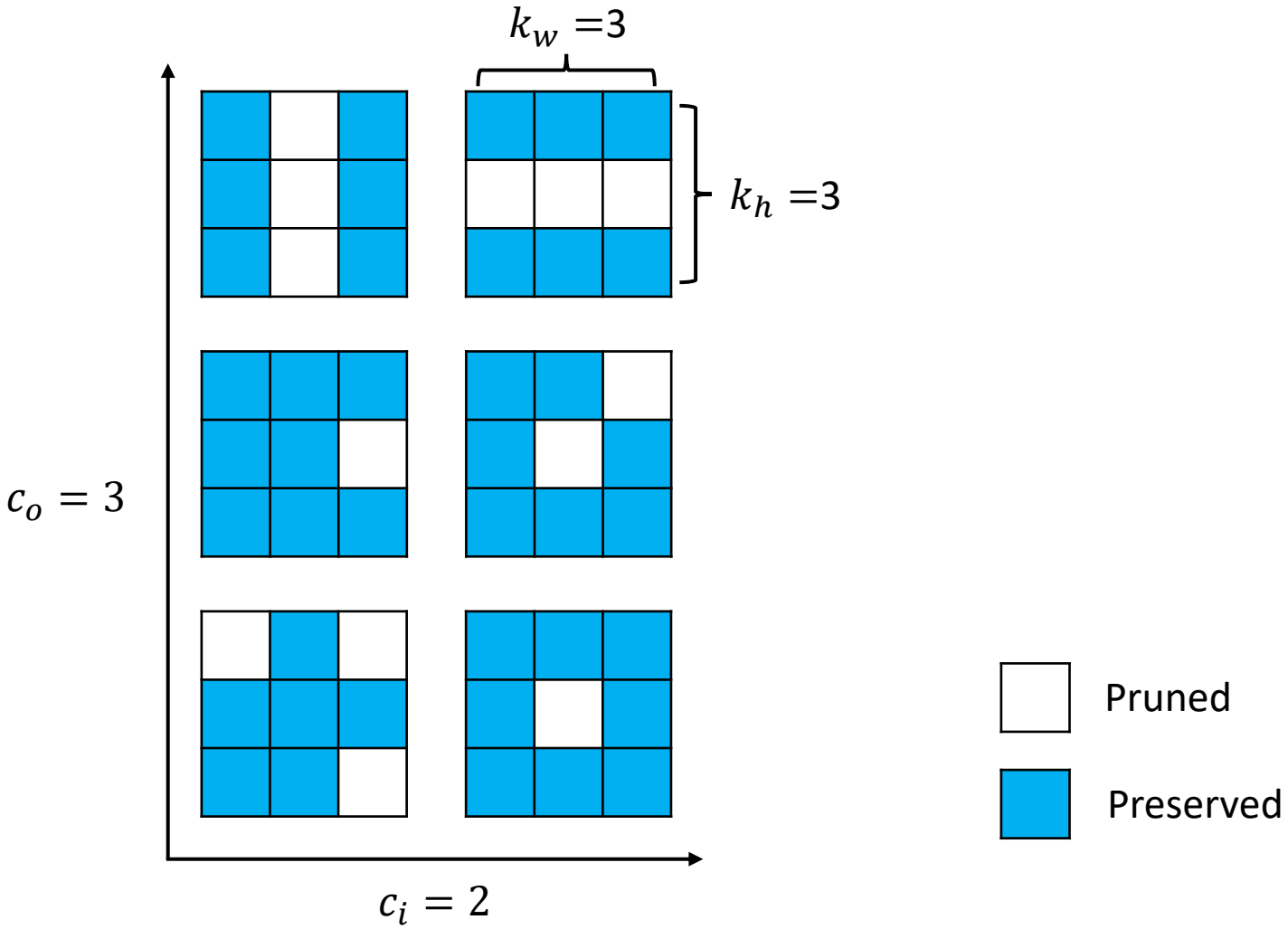
- Less flexible pruning index choice
  - A subset of the fine-grained case
- Easy to accelerate
  - Just a smaller matrix!

The Case of Convolutional Layers

- The weights of convolutional layers have 4 dimensions
  - $[c_o, c_i, k_h, k_w]$
  - c_i: input channels (or channels)
  - c_o: output channels (or filters)
  - k_h: kernel size height
  - h_w: kernel size width
- The 4 dimensions give us more choices to select pruning granularities

Commonly Used Pruning Granularities

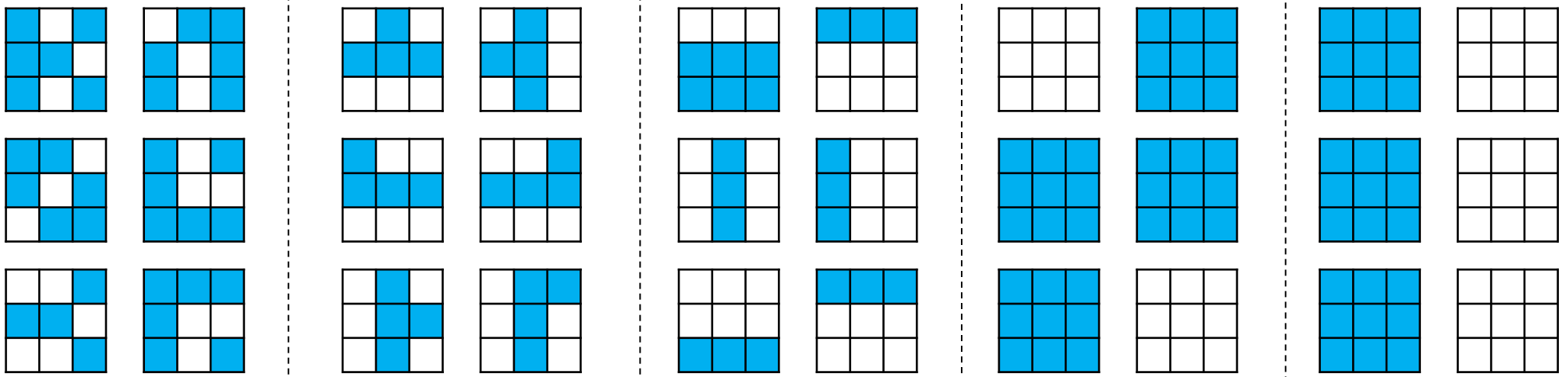
- Notations



Commonly Used Pruning Granularities

Pro and Cons?

Irregular ⟷ Regular



Fine-grained Pruning

Pattern-based Pruning (Like Tetris)

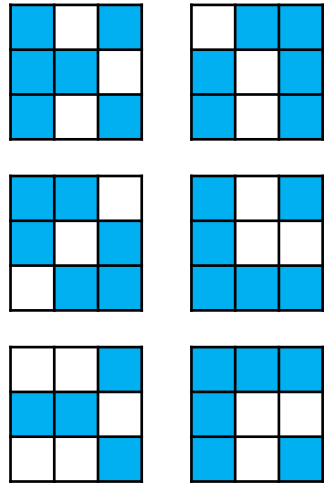
Vector-level Pruning

Kernel-level Pruning

Channel-level Pruning

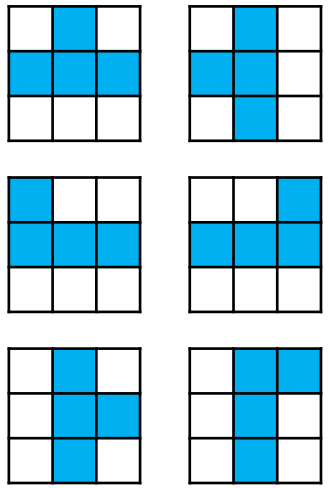
Fine-grained Pruning

- Flexible pruning indices
- Usually larger compression ratio since we can flexibly find "redundant" weights
  - We will later discuss how we find them
- Can deliver speed up on some custom hardware
  - e.g., EIE



Neural Network	Parameters		
	Before Pruning	After Pruning	Reduction
AlexNet	61 M	6.7 M	9x
VGG-16	138 M	10.3 M	12x
GoogLeNet	7 M	2.0 M	3.5x
ResNet50	26 M	7.47 M	3.4x

Pattern-based Pruning

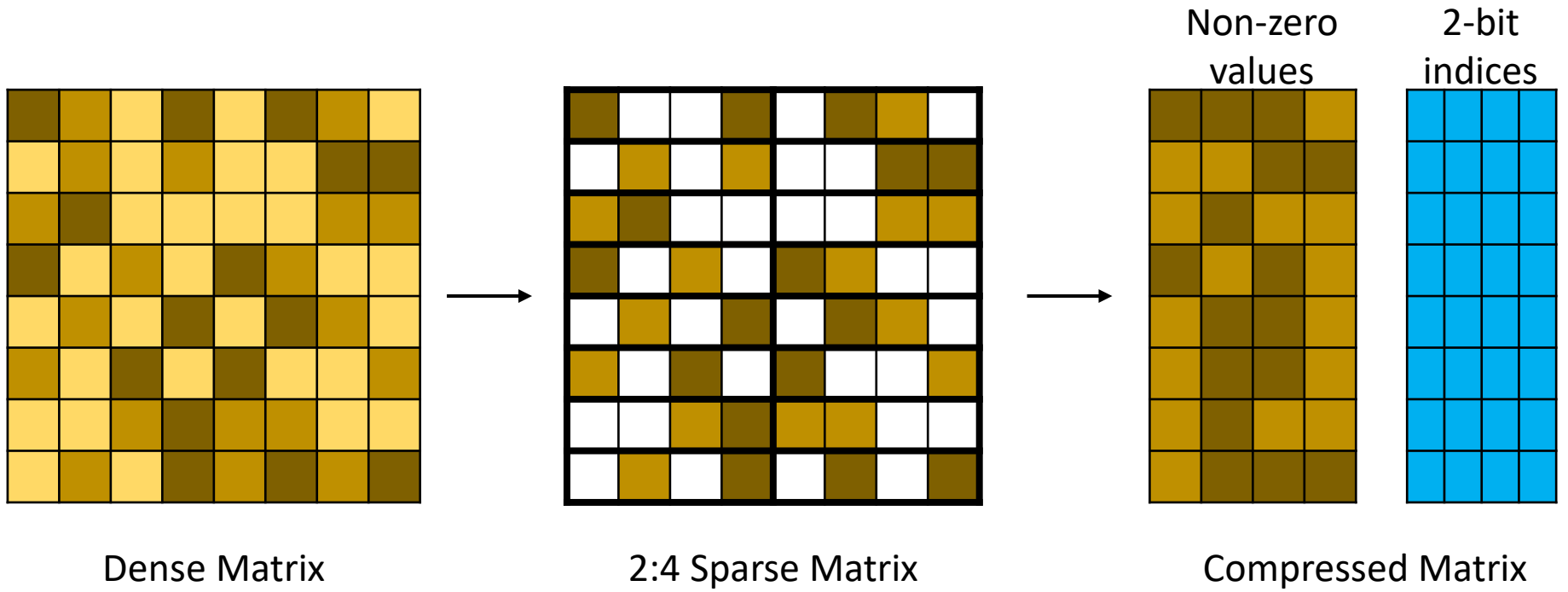


- N:M sparsity
  - Means that in each contiguous M elements, N of them is pruned
  - A classic case is 2:4 sparsity (50% sparsity)
  - Supported by NVIDIA's Ampere GPU Architecture, which delivers up to 2x speed up
  - Usually maintains accuracy

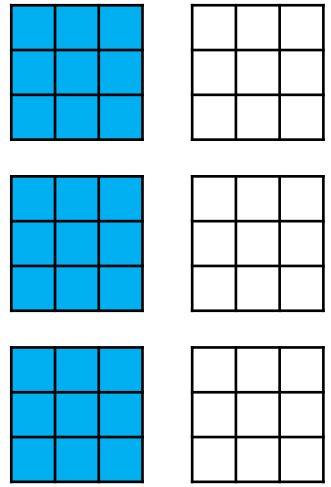
Network	Data Set	Metric	Dense FP16	Sparse FP16
ResNet-50	ImageNet	Top-1	76.1	76.2
ResNeXt-101_32x8d	ImageNet	Top-1	79.3	79.3
Xception	ImageNet	Top-1	79.2	79.2
SSD-RN50	COCO2017	bbAP	24.8	24.8
MaskRCNN-RN50	COCO2017	bbAP	37.9	37.9
FairSeq Transformer	EN-DE WMT'14	BLEU	28.2	28.5
BERT-Large	SQuAD v1.1	F1	91.9	91.9

Pattern-based Pruning

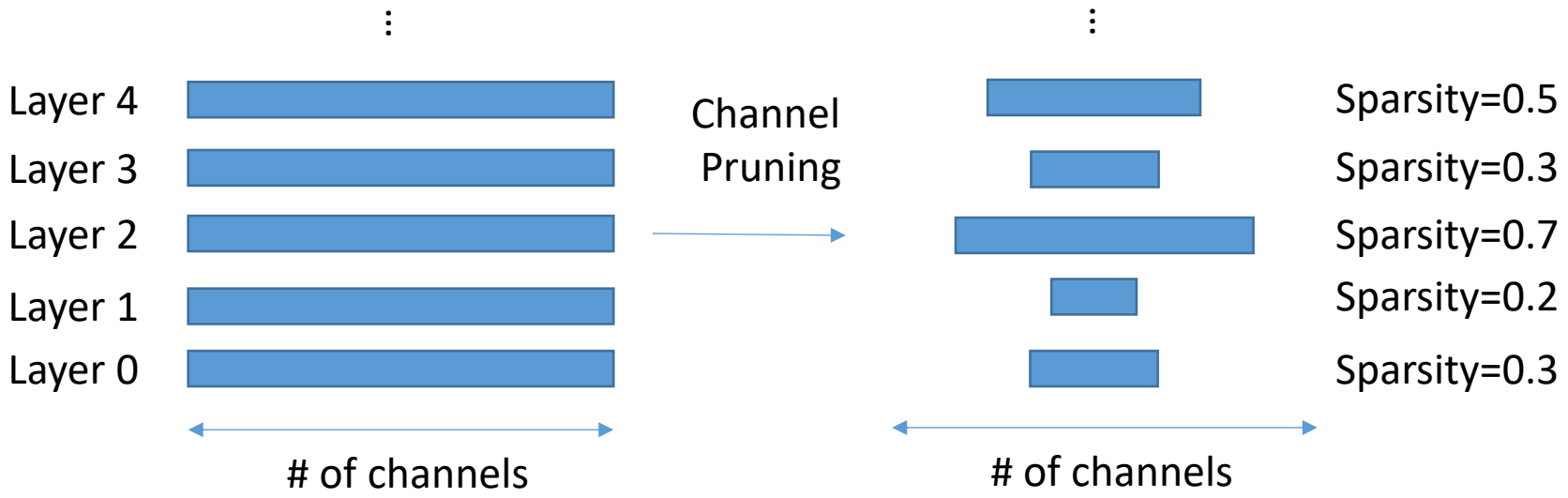
- 2:4 sparsity example

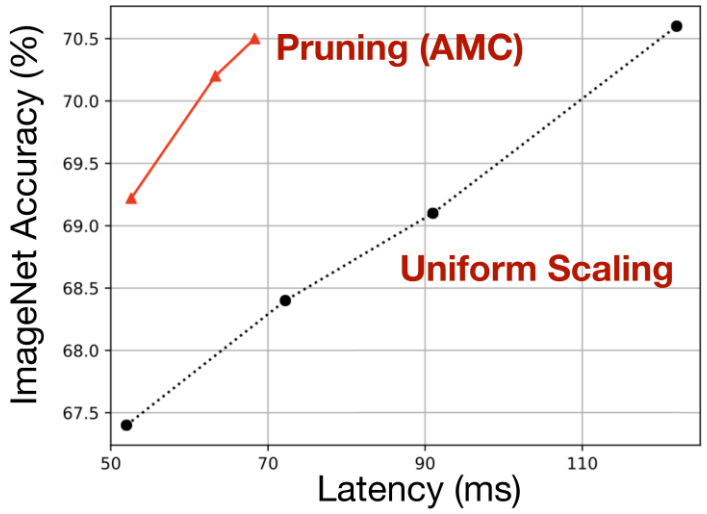


Channel-level Pruning



- Pro
  - Direct speed up due to reduced channel numbers
  - Leading to an NN with smaller number of channels
- Con
  - Smaller compression ratio





Outline

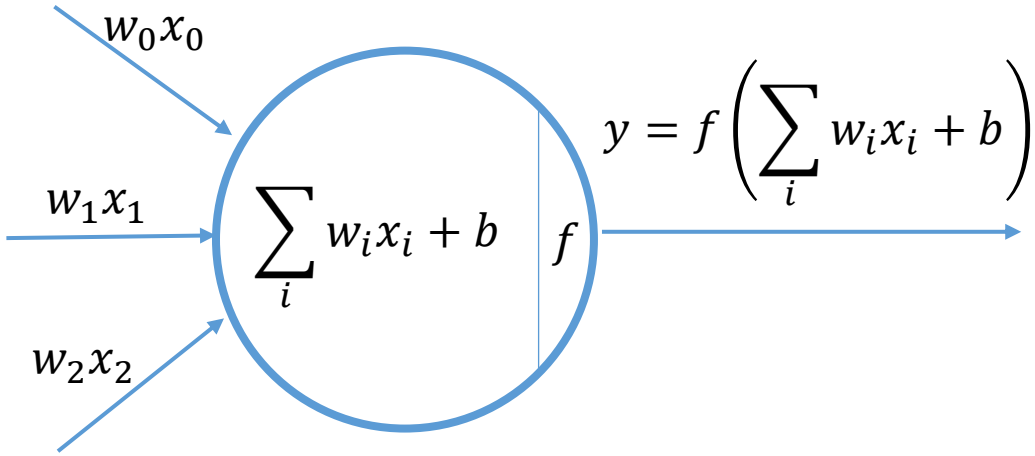
- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

Selection of Synapses to Prune

- When removing parameters from a neural network model
  - **The less important** the parameters being removed are, the better the performance of pruned neural network is.
- Example

$$f(\cdot) = ReLU(\cdot), W = [10, -8, 0.1]$$
$$y = ReLU(10x_0 - 8x_1 + 0.1x_2)$$

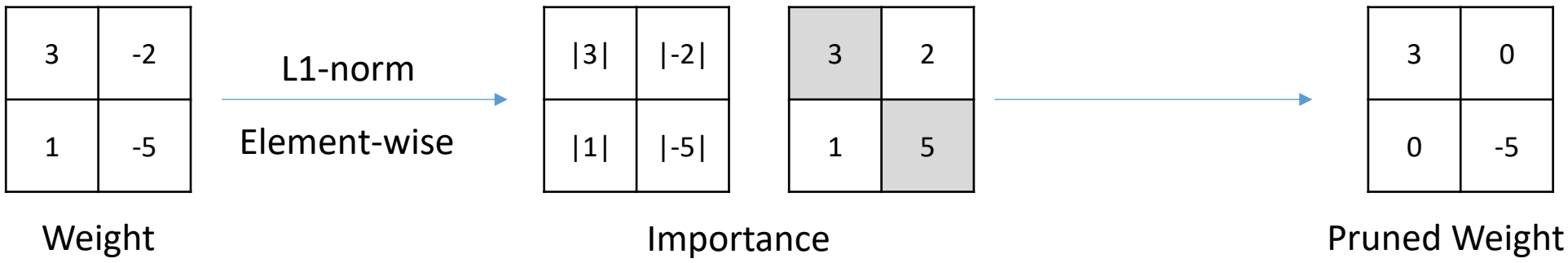
  - If only one weight will be removed
    - Which one? Why?
  - If only one weight will be kept
    - Which one? Why?



Magnitude-based Pruning

- A heuristic pruning criterion
- Considers weights with **larger absolute values** are more important than other weights
- For element-wise pruning

$$Importance = |W|$$

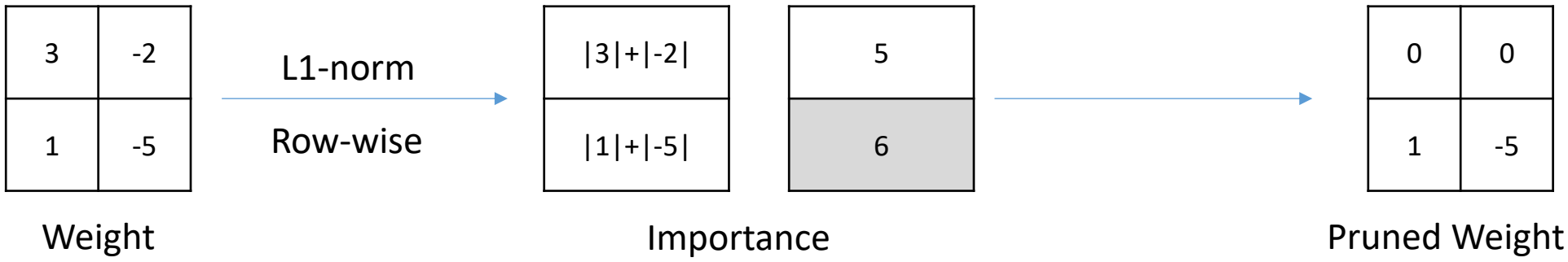


Magnitude-based Pruning

- A heuristic pruning criterion
- Considers weights with **larger absolute values** are more important than other weights
- For row-wise pruning, the L1-norm magnitude can be defined as

$$Importance = \sum_{i \in \mathbb{S}} |W_i|$$

$$, where\ W^{(\mathbb{S})}\ is\ the\ structural\ set\ \mathbb{S}\ of\ parameters\ W$$

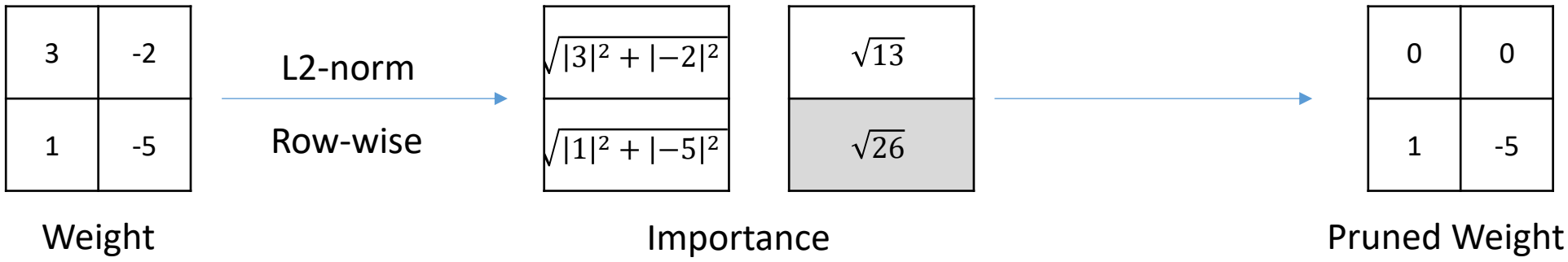


Magnitude-based Pruning

- A heuristic pruning criterion
- Considers weights with **larger absolute values** are more important than other weights
- For row-wise pruning, the L2-norm magnitude can be defined as

$$Importance = \sqrt{\sum_{i \in \mathbb{S}} |W_i|^2}$$

$$, where\ W^{(\mathbb{S})}\ is\ the\ structural\ set\ \mathbb{S}\ of\ parameters\ W$$

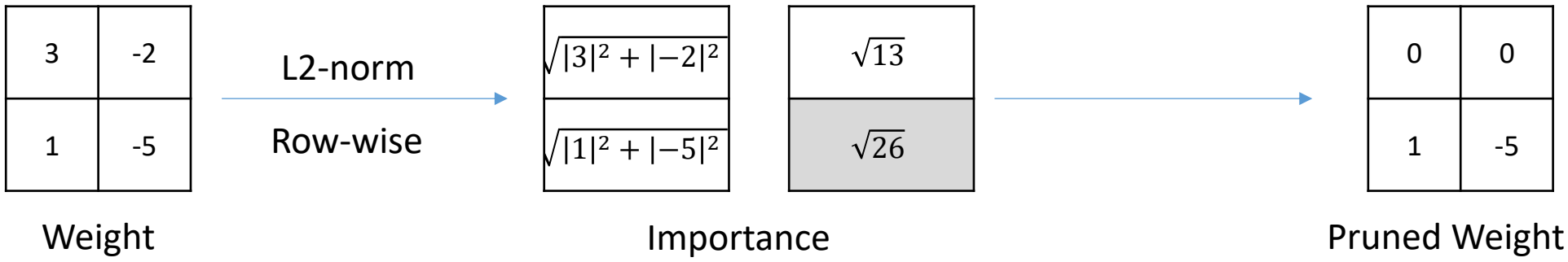


Magnitude-based Pruning

- A heuristic pruning criterion
- Considers weights with **larger absolute values** are more important than other weights
- Magnitude is also known as L_p-norm defined as

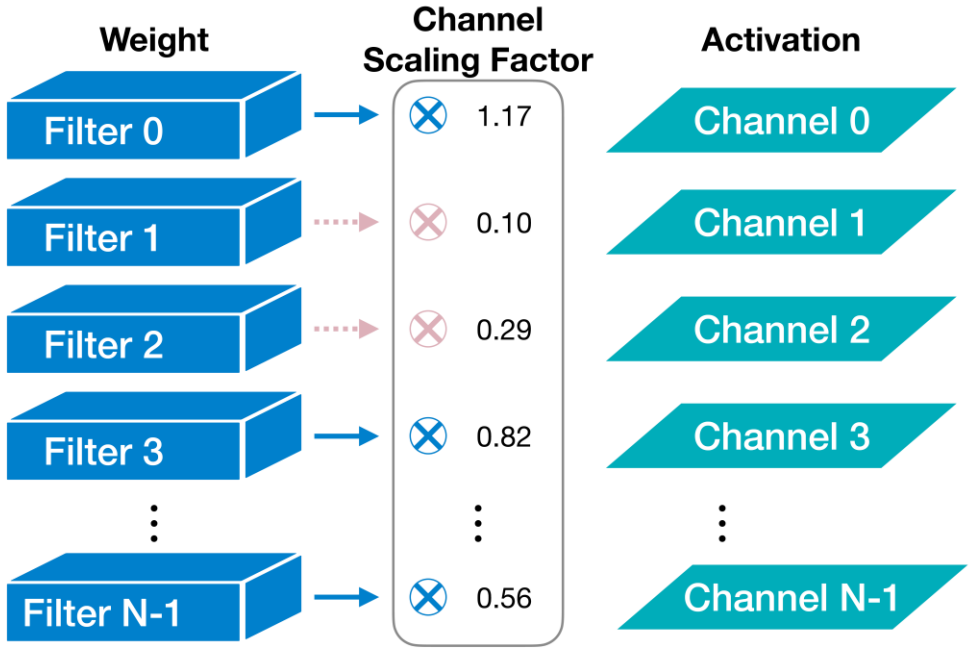
$$Importance = \left(\sum_{i \in \mathbb{S}} |W_i|^p \right)^{\frac{1}{p}}$$

$$, where\ W^{(\mathbb{S})}\ is\ the\ structural\ set\ \mathbb{S}\ of\ parameters\ W$$



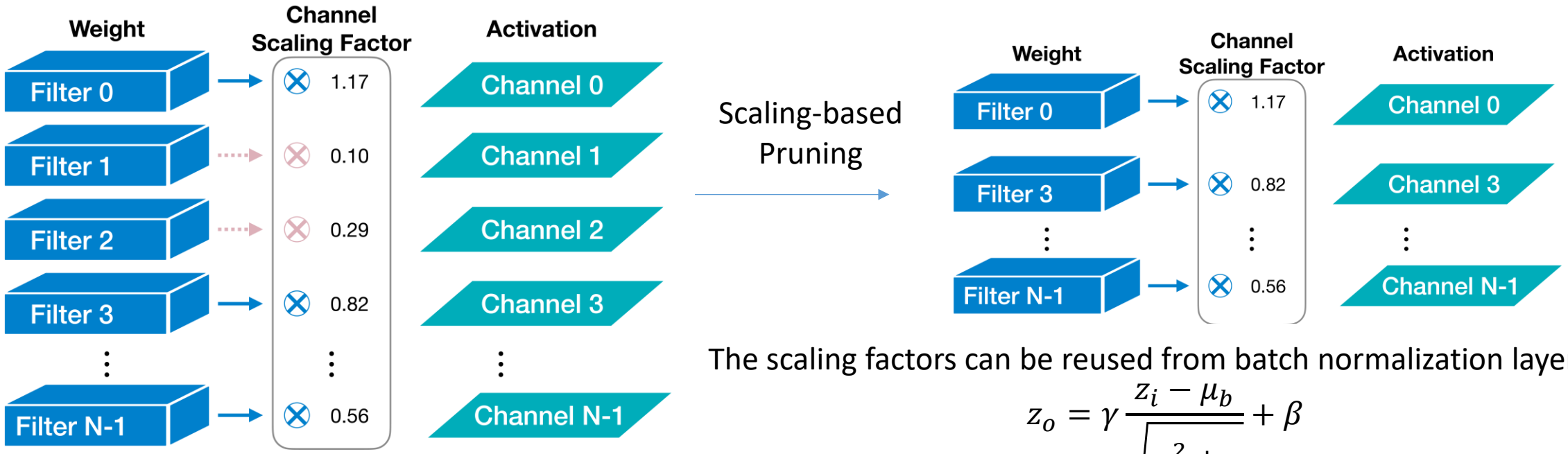
Scaling-based Pruning

- Pruning criterion for filter pruning
- A scaling factor is associated with each filter (i.e., output channel) in convolutional layers
  - The scaling factor is multiplied to the output of that channel
  - The scaling factors are trainable parameters



Scaling-based Pruning

- Pruning criterion for filter pruning
- A scaling factor is associated with each filter (i.e., output channel) in convolutional layers
  - The scaling factor is multiplied to the output of that channel
  - The scaling factors are trainable parameters
- The filters/output channels with small scaling factor magnitude will be pruned



The scaling factors can be reused from batch normalization layer

$$z_o = \gamma \frac{z_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta$$

Taylor Expansion Analysis on Pruning Error

- Evaluate pruning error induced by pruning synapses
- The task of training neural network is to minimize the objective function $L(x; W)$
  - The importance of a parameter can be quantified by the error induced by removing it
- The induced error can be approximated by a Taylor series

$$\delta L = L(x; W) - L\left(x; \mathrm{W}_p = W - \delta W\right)$$

$$= \sum_i g_i \delta w_i + \frac{1}{2} \sum_i h_{ii} \delta w_i^2 + \frac{1}{2} h_{ij} \delta w_i \delta w_j + O(\|\delta W\|^3)$$

  - Where

$$g_i = \frac{\partial L}{\partial w_i}, h_{i,j} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

- There are two ways to simplify these approximation

Second-Order-based Pruning

Minimize the error on loss function introduced by pruning synapses

- The induced error can be approximated by a Taylor series

$$\delta L = L(x; W) - L\left(x; \mathrm{W}_p = W - \delta W\right)$$

$$= \sum_{\cancel{i}} \cancel{g_i \delta w_i} + \frac{1}{2}\sum_{i} h_{ii}\delta w_i^2 + \frac{1}{2}\sum_{\cancel{i \neq j}} \cancel{h_{ij}\delta w_i \delta w_j} + \cancel{O(\|\delta W\|^3)}$$

  - Where

$$g_i = \frac{\partial L}{\partial w_i}, h_{i,j} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

- Optimal Brain Damage assumes that
  - The objective function L is nearly quadratic → **the last term is neglected**
  - The neural network training has converged → **first-order terms are neglected**
  - The error caused by deleting each parameter is independent → **cross terms are neglected**

$$\delta L_i = L(x; W) - L\left(x; W_p \middle| w_i = 0\right) \approx \frac{1}{2} h_{ii} w_i^2$$

where $h_{ii} = \frac{\partial^2 L}{\partial w_i \partial w_i}$

Second-Order-based Pruning

- The synapses with smaller induced error $|\delta L_i|$ will be removed
- That is to say

$$Importance_{w_i} = |\delta L_i| = \frac{1}{2} h_{ii} w_i^2$$

h_{ii} is non-negative

Hessian Matrix H is difficult to compute

First-Order-based Pruning

Minimize the error on loss function introduced by pruning synapses

- The induced error can be approximated by a Taylor series

$$\delta L = L(x; W) - L\left(x; \mathrm{W}_p = W - \delta W\right)$$

$$= \sum_i g_i \delta w_i + \frac{1}{2}\sum_i \cancel{h_{ii}\delta w_i^2} + \frac{1}{2}\sum_{i \neq j} \cancel{h_{ij}\delta w_i \delta w_j} + \cancel{O(\|\delta W\|^3)}$$

  - Where

$$g_i = \frac{\partial L}{\partial w_i}, h_{i,j} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

- If only first-order expansion is considered, under an i.i.d assumption,

$$\delta L_i = L(x; W) - L\left(x; W_p \middle| w_i = 0\right) \approx g_i w_i$$

  - Where $g_i = \frac{\partial L}{\partial w_i}$

First-Order-based Pruning

- The synapses with smaller induced error $|\delta L_i|$ will be removed; that is to say

$$importance_{w_i} = |\delta L_i| = |g_i w_i|$$

- Or

$$importance_{w_i} = |\delta L_i|^2 = (g_i w_i)^2$$

- For coarse-grained pruning, we have

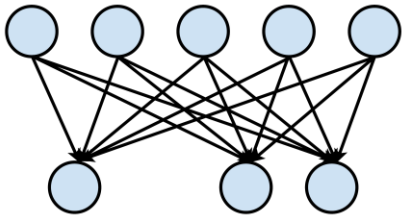
$$importance_{w^{(\mathbb{S})}} = \sum_{i \in \mathbb{S}} |\delta L_i|^2 = \sum_{i \in \mathbb{S}} (g_i w_i)^2$$

  - Where $W^{(\mathbb{S})}$ is the structural set of parameters

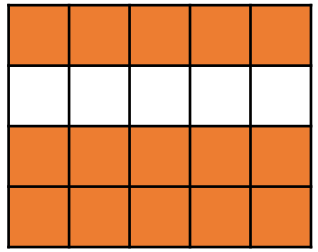
Selection of Neurons to Prune

- When removing neurons from a neural network model
  - **The less useful** the neurons being removed are, the better the performance of pruned neural network is.
- Recall: Neuron pruning is coarse-grained pruning indeed

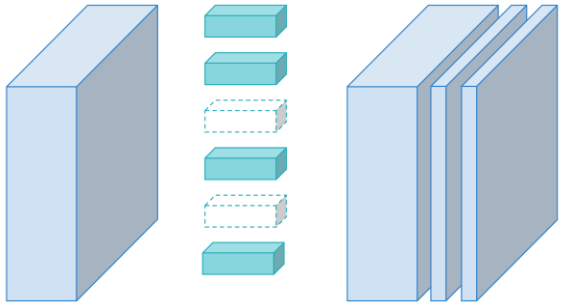
Neuron Pruning
in Linear Layer

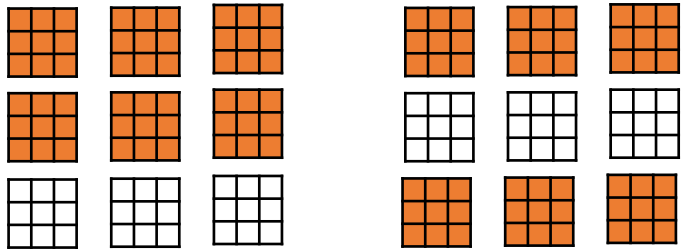


Weight Matrix



Channel Pruning
in Convolution Layer



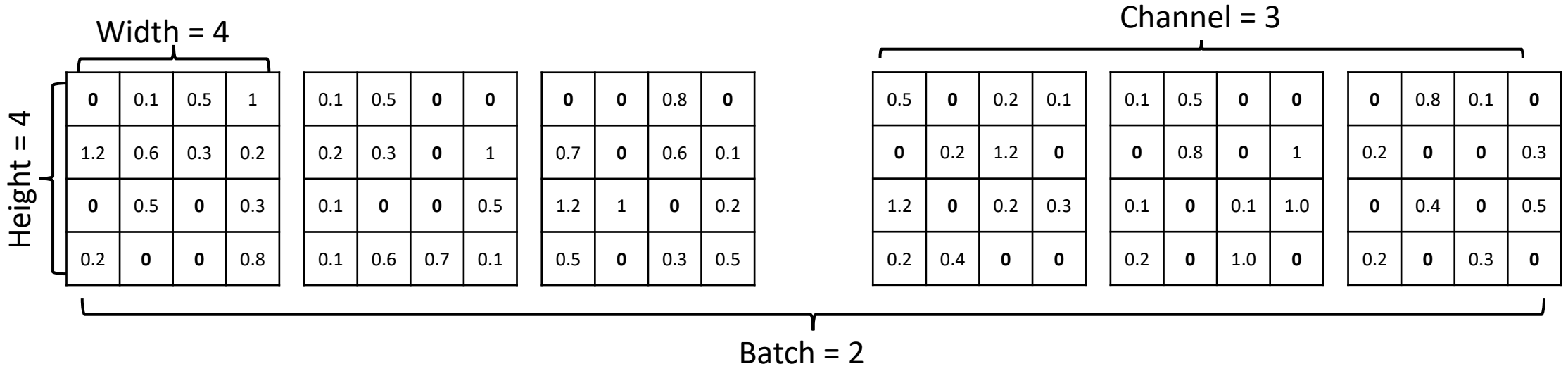


Percentage-of-Zero-Based Pruning

- ReLU activation will generate zeros in the output activation
- Similar to magnitude of weights, the **Average Percentage of Zero activations (APoZ)** can be exploited to measure the importance of the neurons
  - The smaller APoZ is, the more importance the neuron has

Example of APoZ

Output Activation



0	0.1	0.5	1
1.2	0.6	0.3	0.2
0	0.5	**0**	0.3
0.2	**0**	**0**	0.8

0.1	0.5	**0**	**0**
0.2	0.3	**0**	1
0.1	**0**	**0**	0.5
0.1	0.6	0.7	0.1

0	**0**	0.8	**0**
0.7	**0**	0.6	0.1
1.2	1	**0**	0.2
0.5	**0**	0.3	0.5

0.5	**0**	0.2	0.1
0	0.2	1.2	**0**
1.2	**0**	0.2	0.3
0.2	0.4	**0**	**0**

0.1	0.5	**0**	**0**
0	0.8	**0**	1
0.1	**0**	0.1	1.0
0.2	**0**	1.0	**0**

0	0.8	0.1	**0**
0.2	**0**	**0**	0.3
0	0.4	**0**	0.5
0.2	**0**	0.3	**0**

Average Percentage of Zeros (APoZ)

$= \frac{\mathbf{5+6}}{\mathbf{2\times4\times4}} = \frac{\mathbf{11}}{\mathbf{32}}$ Channel 0

$= \frac{5+7}{2\times4\times4} = \frac{12}{32}$ Channel 1

$= \frac{\mathbf{6+8}}{\mathbf{2\times4\times4}} = \frac{\mathbf{14}}{\mathbf{32}}$ Channel 2

First-Order-based Pruning

Minimize the error on loss function introduced by pruning neurons

- Similar to previous Taylor expansion on weights, the induced error $L(x; W)$ of the objective function can be approximated by a Taylor series expanded on activations

$$\delta L_i = L(x; W) - L(x|x_i = 0; W) \approx \frac{\partial L}{\partial x_i} x_i$$

- For a structural set of neurons $X^{(\mathbb{S})}$(e.g., a channel plane)

$$\left|\delta L_{X^{(\mathbb{S})}}\right| = \left|\sum_{i \in \mathbb{S}} \frac{\partial L}{\partial x_i} x_i\right|$$

Regression-based Pruning

Minimize reconstruction error of the corresponding layer's outputs

- Instead of considering the pruning error of the objective function $L(x; W)$
  - Regression-based pruning minimizes the reconstruction error of the corresponding layer's outputs.
- Let

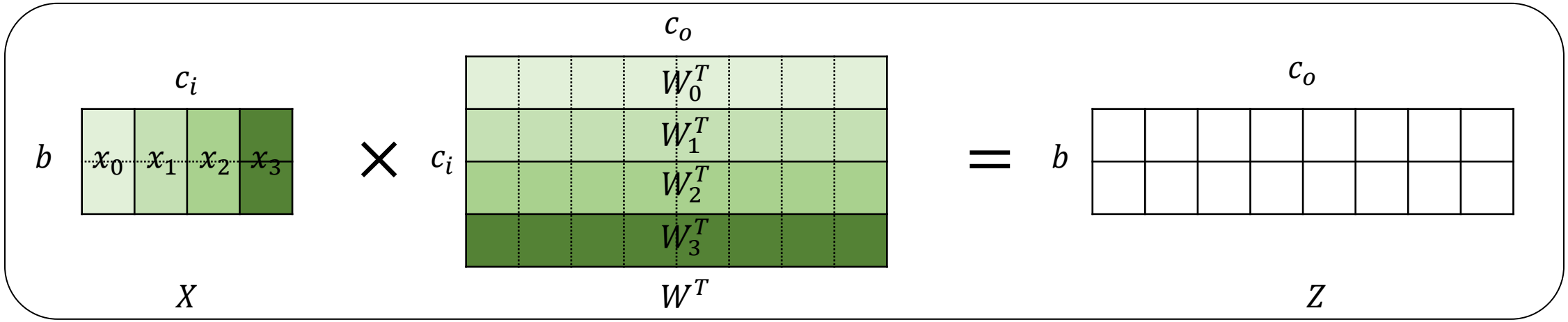
$$Z = XW^T = \sum_{c=0}^{c_i-1} X_c W_c^T$$

- The problem can be formulate as

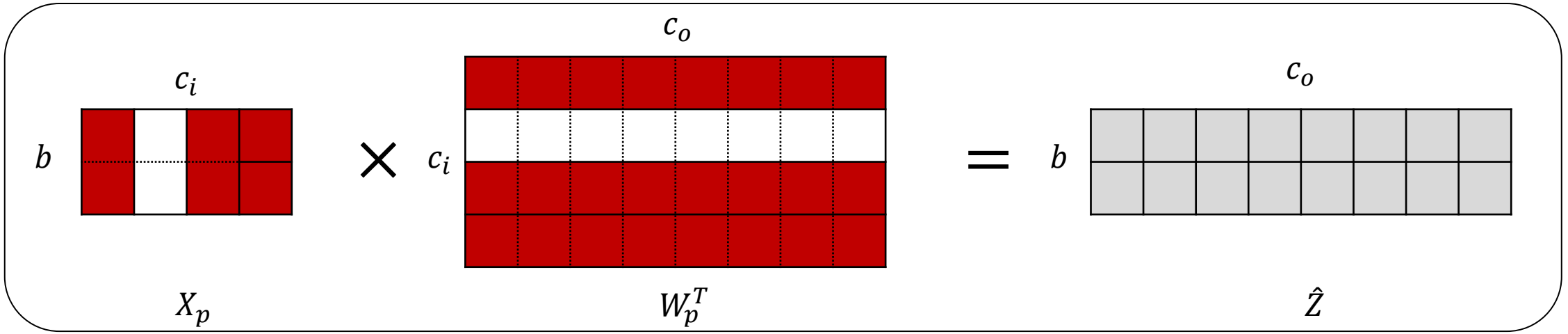
$$\arg\min_{W,\beta} \left\| Z - \hat{Z} \right\|_F^2 = \left\| Z - \sum_{c=0}^{c_i-1} \beta_c X_c W_c^T \right\|_F^2$$

  - Subject to $\|\beta\|_0 \leq N_c$
  - β: coefficient vector of length c_i for channel selection.
    - $\beta_c = 0$ means channel c is pruned
  - N_c: The number of non-zero channel

Regression-based Pruning



Minimize the error between Z and $\hat{Z}$



Regression-based Pruning

- Solve the problem in two folds:
  - Fix W, solve β for channel selection (**NP-hard**)
  - Fix β, solve W to minimize reconstruction error
- The objective can be rewritten as

$$\arg\min_{\beta} \left\| Z - \sum_{c=0}^{c_i-1} \beta_c X_c W_c^T \right\|_F^2 = \left\| \sum_{c=0}^{c_i-1} X_c W_c^T - \sum_{c=0}^{c_i-1} \beta_c X_c W_c^T \right\|_F^2$$

$$= \left\| \sum_{c=0}^{c_i-1} (1-\beta_c) X_c W_c^T \right\|_F^2$$

Subject to $\|\beta\|_0 \leq N_c$

Exmple - ThiNet

- ThiNet proposed a greedy solution to optimize

$$\arg\min_{\mathbb{S}} \left\| \sum_{c \in \mathbb{S}} X_c W_c^T \right\|_F^2, s.t. card(\mathbb{S}) \leq N_c$$

  - Iteratively add only one channel into pruning set $\mathbb{S}$ leading to the smallest L2-norm

```
1: S = []
2: while len(S) < N:
3:     min_norm, min_c = +inf, 0
4:     for c in range(c_i):
5:         tmpS = S + [c]
6:         Z = X[:,tmpS] * W[:,tmpS].t()
7:         norm = Z.norm(2)
8:         if norm < min_norm:
9:             min_norm, min_c = norm, c
10:    S.append(min_c)
```

Luo, J. H., Wu, J., & Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision* (pp. 5058-5066).

Regression-based Pruning

- Back to

$$\arg\min_{\beta}\left\|Z-\hat{Z}\right\|_F^2=\left\|Z-\sum_{c=0}^{c_i-1}\beta_c X_c W_c^T\right\|_F^2$$

  - Subject to $\|\beta\|_0 \le N_c$
- Relax the l_0 to l_1 regularization (LASSO):

$$\arg\min_{\beta}\left\|Z-\sum_{c=0}^{c_i-1}\beta_c X_c W_c^T\right\|_F^2+\lambda\|\beta\|_1$$

  - λ: penalty coefficient
    - By increasing λ, there will be more zeros in β
- Gradually increase λ and solve the LASSO regression for β, until $\|\beta\|_0 == N_c$ is met

Regression-based Pruning

- Weight Reconstruction

$$\arg\min_{W}\left\|Z-\hat{Z}\right\|_F^2=\left\|Z-\sum_{c=0}^{c_i-1}\beta_c \mathrm{X}_c W_c^T\right\|_F^2$$

  - β: coefficient vector from previous step
- This is a classic linear regression problem, which has a unique closed-form solution using the least squares approach

$$\arg\min_{W}\left\|Z-\hat{Z}\right\|_F^2=\left\|Z-UW_F^2\right\|_F^2$$

  - Where

$$U=[\beta_0 X_0,\beta_1 X_1,\dots,\beta_c X_c,\dots,\beta_{c_i-1}X_{c_i-1}]$$

  - And thus

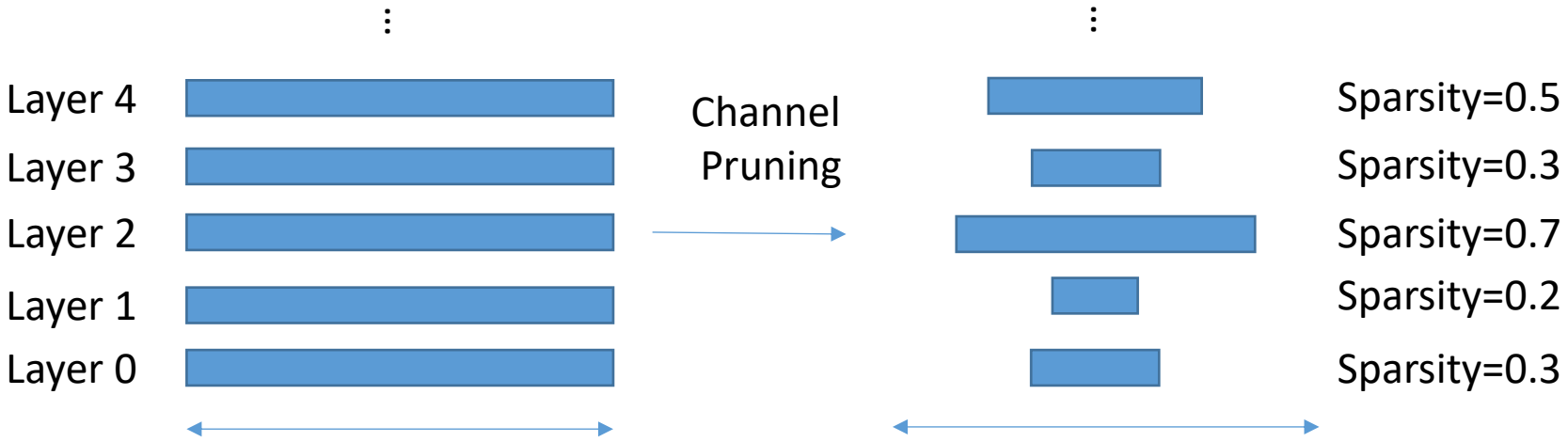
$$W^T=(U^TU)^{-1}U^TZ$$

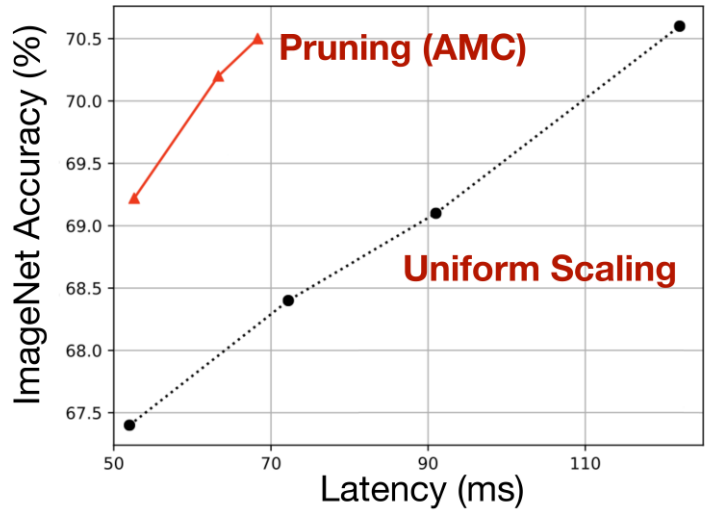
Outline

- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

Pruning Ratio

- Non-uniform pruning is better than uniform shrinking





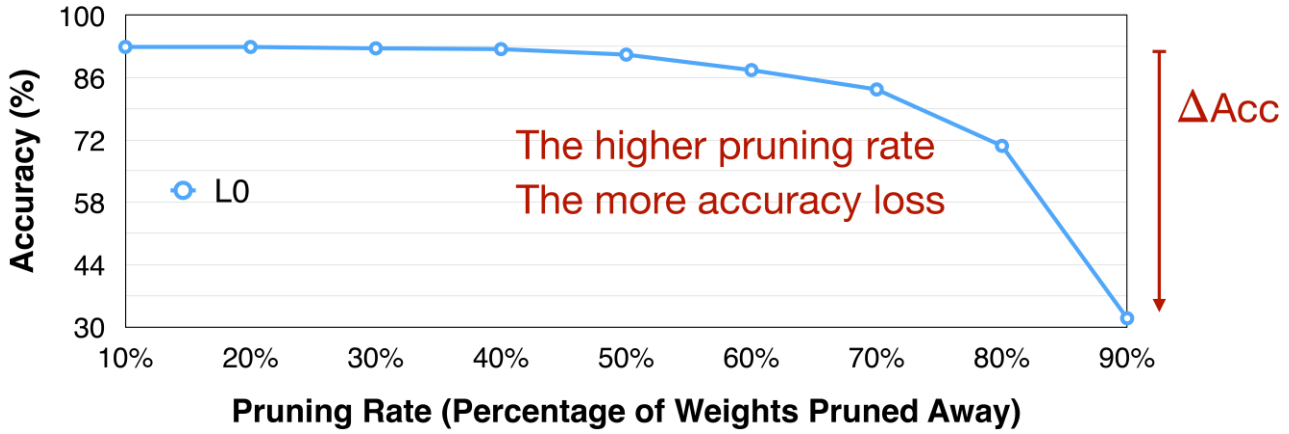
Question: how should we find ratios for each layer?

Finding Pruning Ratios

- We need different pruning ratios for each layer since different layers have different **sensitivity**
  - Some layers are more sensitive
    - e.g., first layer
  - Some layers are more redundant
- We can perform **sensitivity analysis** to determine the per-layer pruning ratio

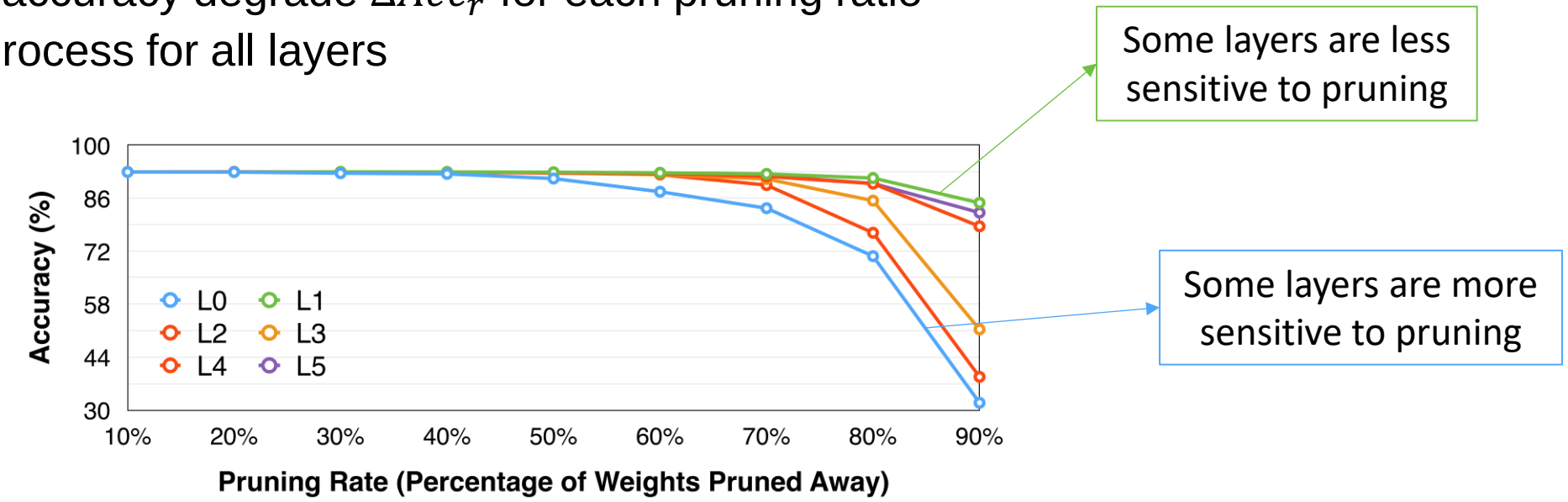
Sensitivity Analysis

- Analyze the sensitivity of each layer
- The process of Sensitivity Analysis
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, \ldots, 0.9\}$
      - Or other strides
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio



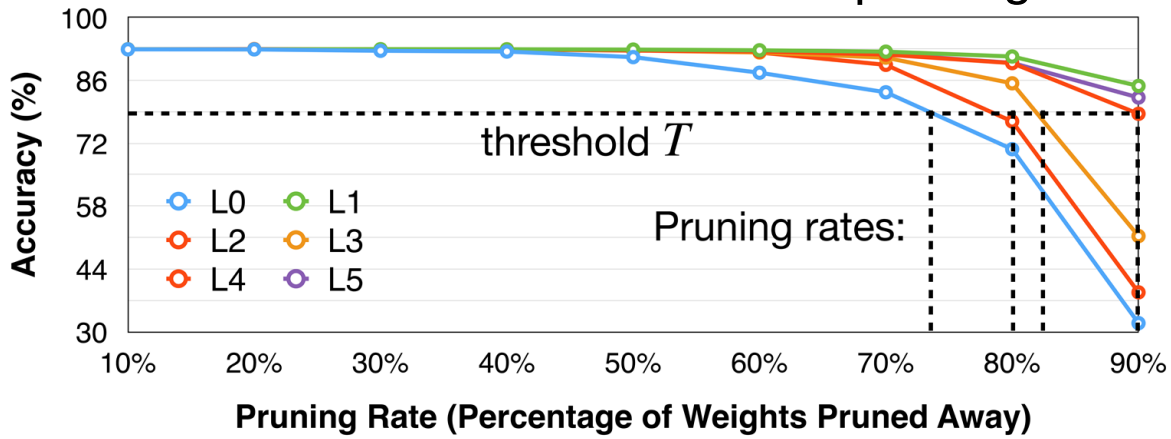
Sensitivity Analysis

- Analyze the sensitivity of each layer
- The process of Sensitivity Analysis
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, \ldots, 0.9\}$
      - Or other strides
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
    - Repeat the process for all layers



Sensitivity Analysis

- Analyze the sensitivity of each layer
- The process of Sensitivity Analysis
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, \ldots, 0.9\}$
      - Or other strides
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
    - Repeat the process for all layers
    - Pick a degradation threshold such that the overall pruning rate is desired

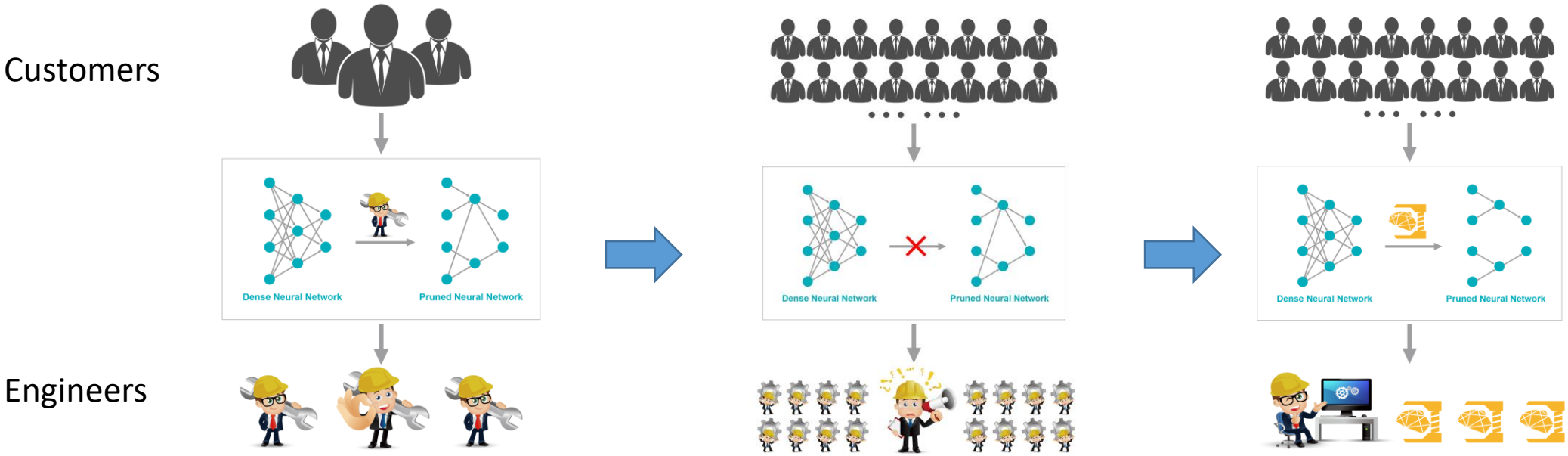


Sensitivity Analysis

- Is this optimal?
  - Maybe not
  - We do not consider the interaction between layers
- Can we go beyond the heuristics?
  - Yes!

Automatic Pruning

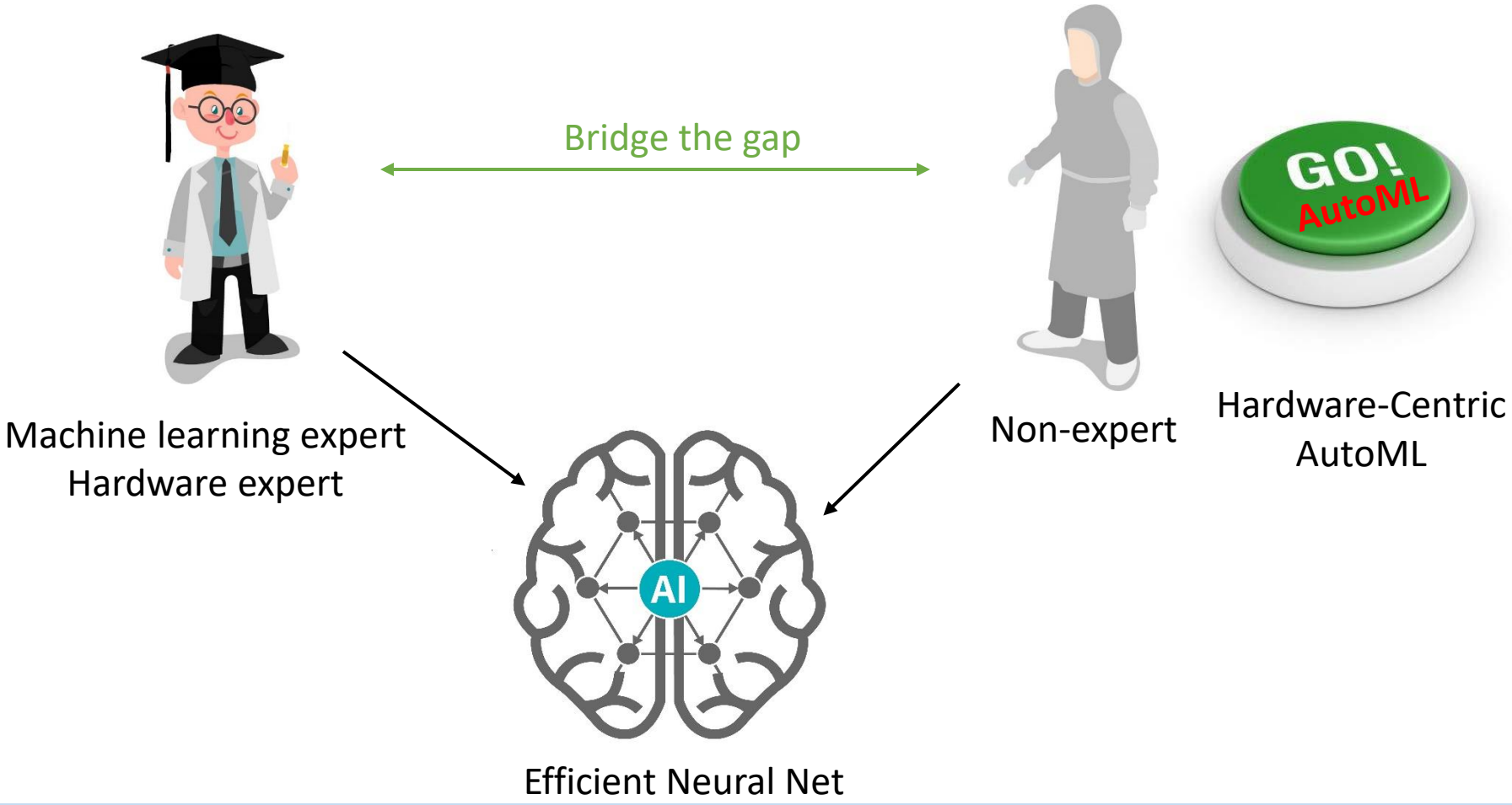
- Given an **overall** compression ratio, how do we choose **per-layer** pruning ratios?
  - Sensitivity analysis ignores the interaction between layers → sub-optimal
- Conventionally, such process relies on human expertise and trials and errors



He, Y., Lin, J., Liu, Z., Wang, H., Li, L. J., & Han, S. (2018). Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 784-800).

Automatic Pruning

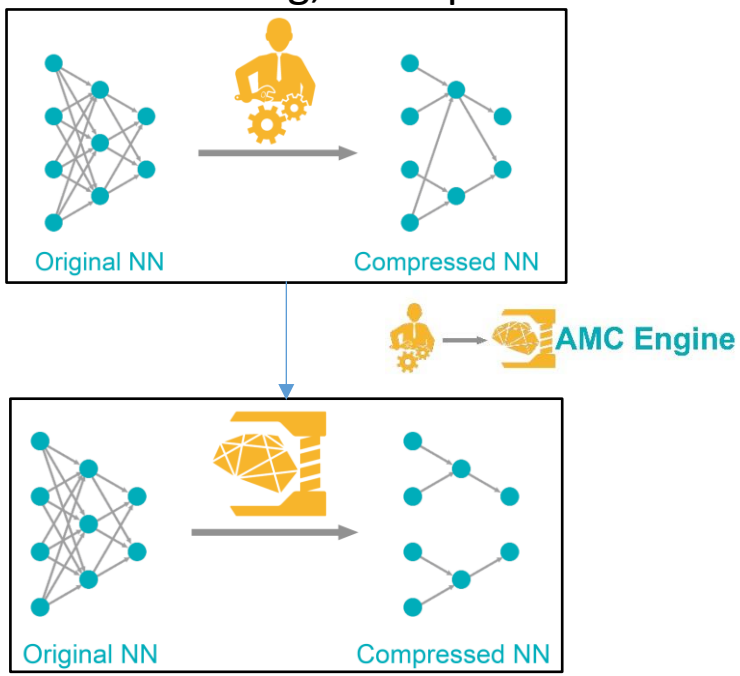
- Can we develop a push-the-button solution?



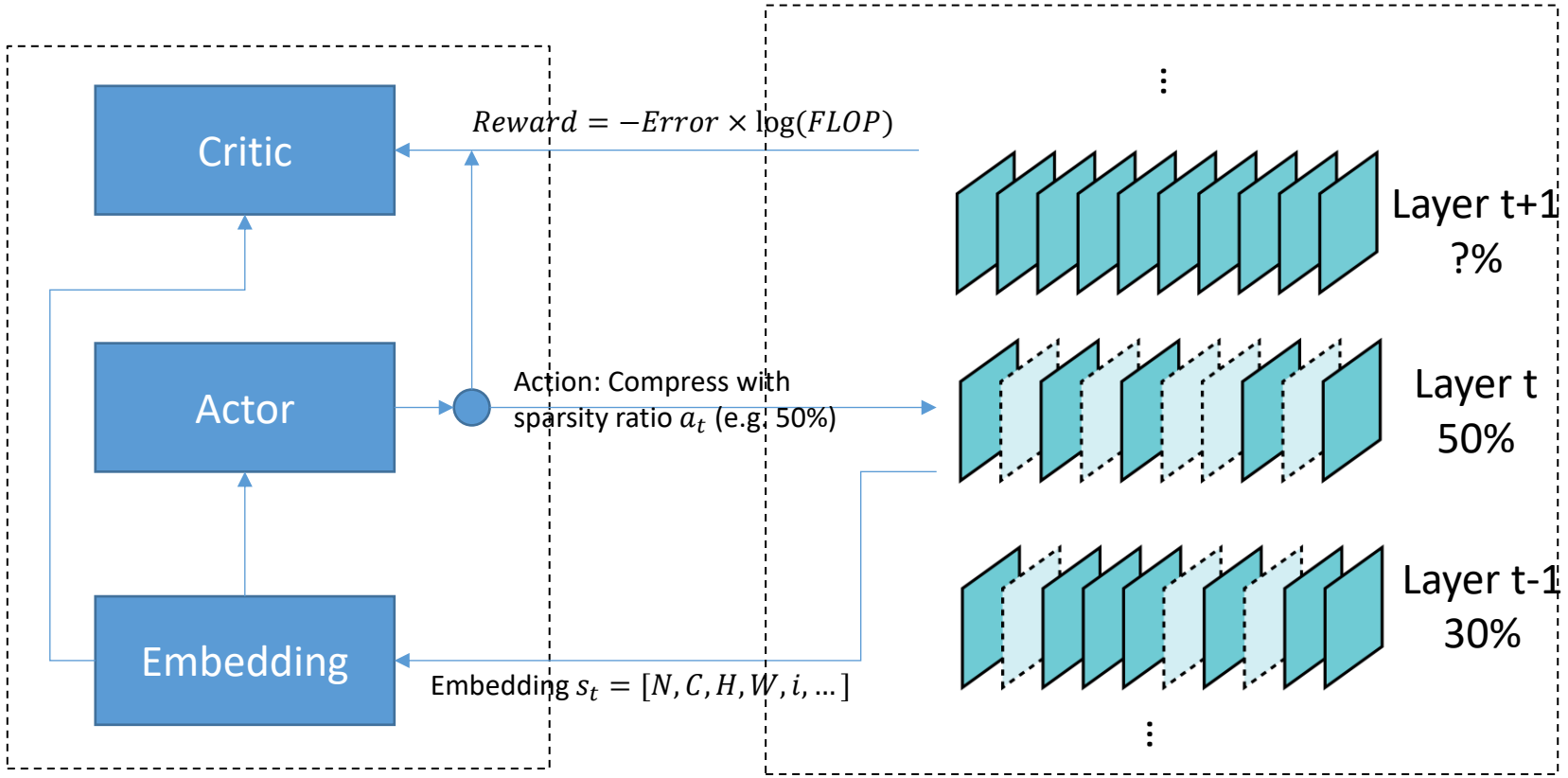
AMC: AutoML for Model Compression

- Pruning as a reinforcement learning problem

Model Compression by Human:
Labor Consuming, Sub-optimal



Model Compression by AI:
Automated, Higher
Compression Rate, Faster



Agent: DDPG

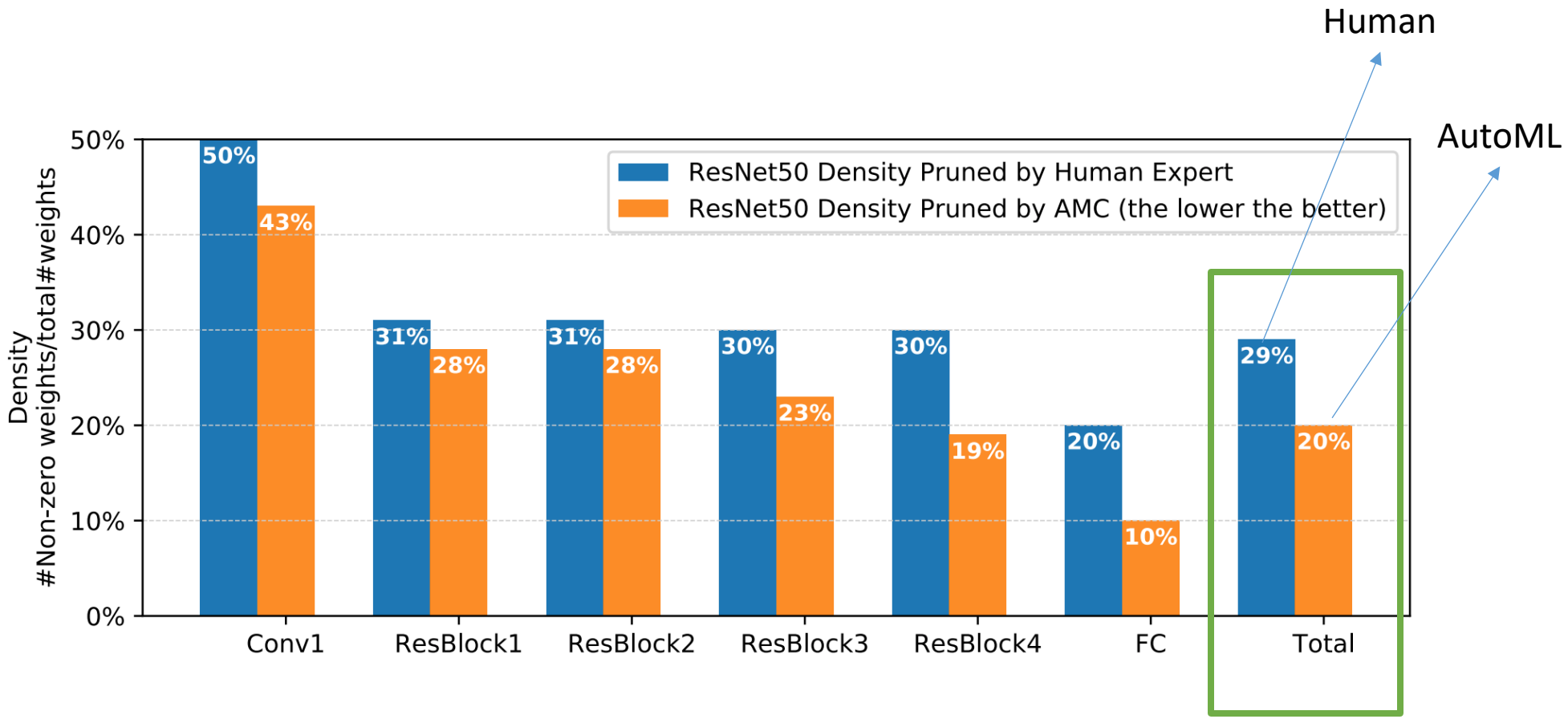
Environment: Channel Pruning

AMC: AutoML for Model Compression

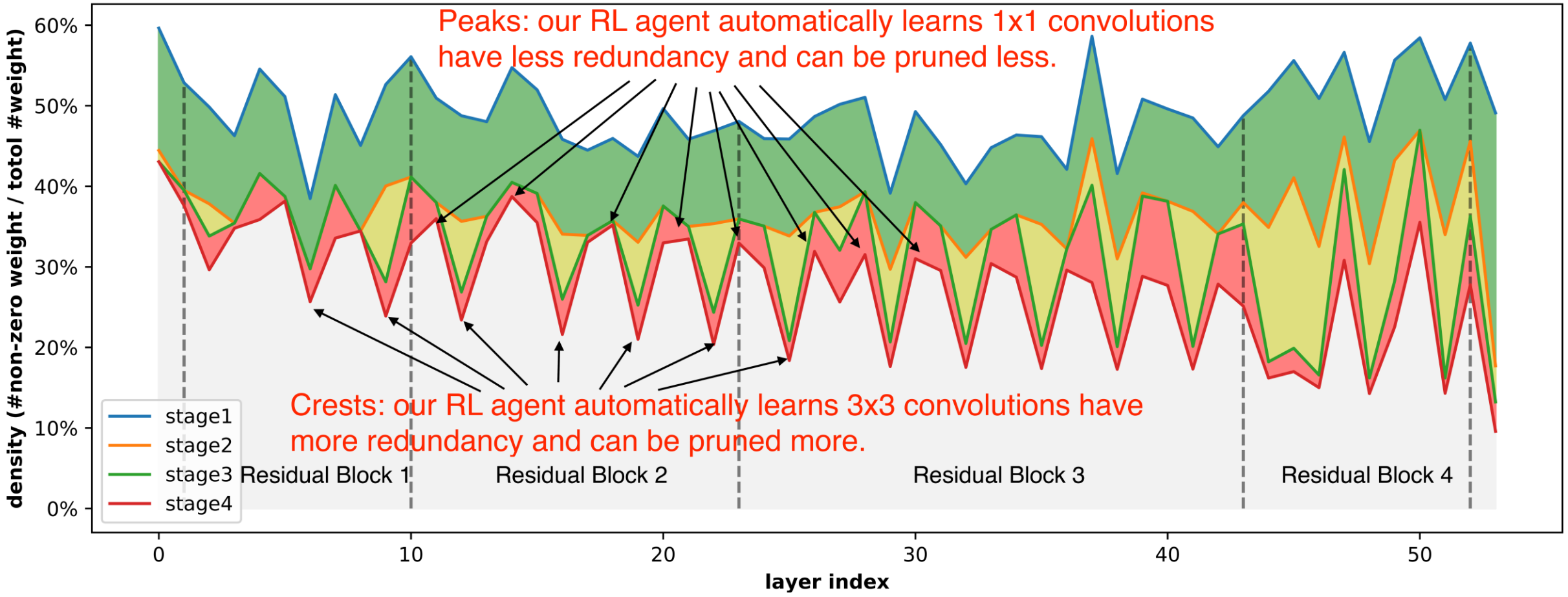
- AMC uses the following setups for the reinforcement learning problem
  - State
    - 11 features (including layer indices, channel numbers, kernel sizes, FLOPs, …)
  - Action
    - A continuous number (pruning ratio) $a \in [0,1)$
  - Agent
    - DDPG agent, since it supports continuous action output
  - Reward
    - $R = \begin{cases} -Error, & if\ satisfies\ constrains \\ -\infty, & if\ not \end{cases}$
    - We can also optimize latency constraints with a pre-built lookup table (LUT)

AMC: AutoML for Model Compression

Smaller is better



AMC: AutoML for Model Compression



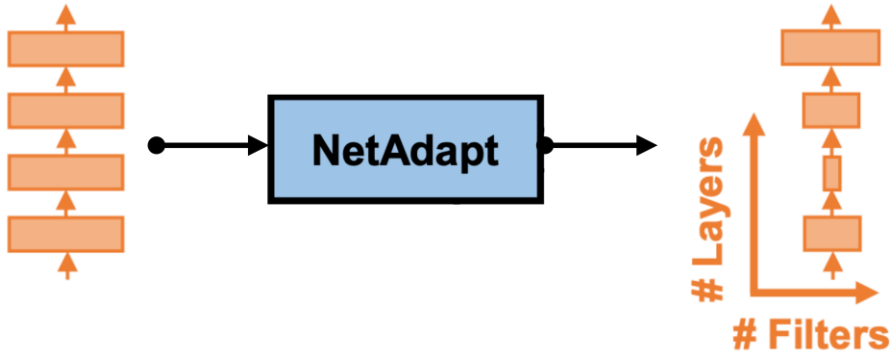
AMC: AutoML for Model Compression

Model	MAC (M)	Top-1	Latency* (ms)	Speedup	Memory (MB)
1.0 MobileNet	569	70.6%	119.0	1x	20.1
AMC(50% FLOPs)	285	70.5%	64.4	1.8x	14.3
AMC(50% Time)	272	70.2%	59.7	2.0x	13.2
0.75 MobileNet	325	68.4%	69.5	1.7x	14.8

* Measured with TF-Lite on Samsung Galaxy S7 Edge, which has Qualcomm Snapdragon SoC
Single core, Batch size = 1(mobile, latency oriented)

NetAdapt

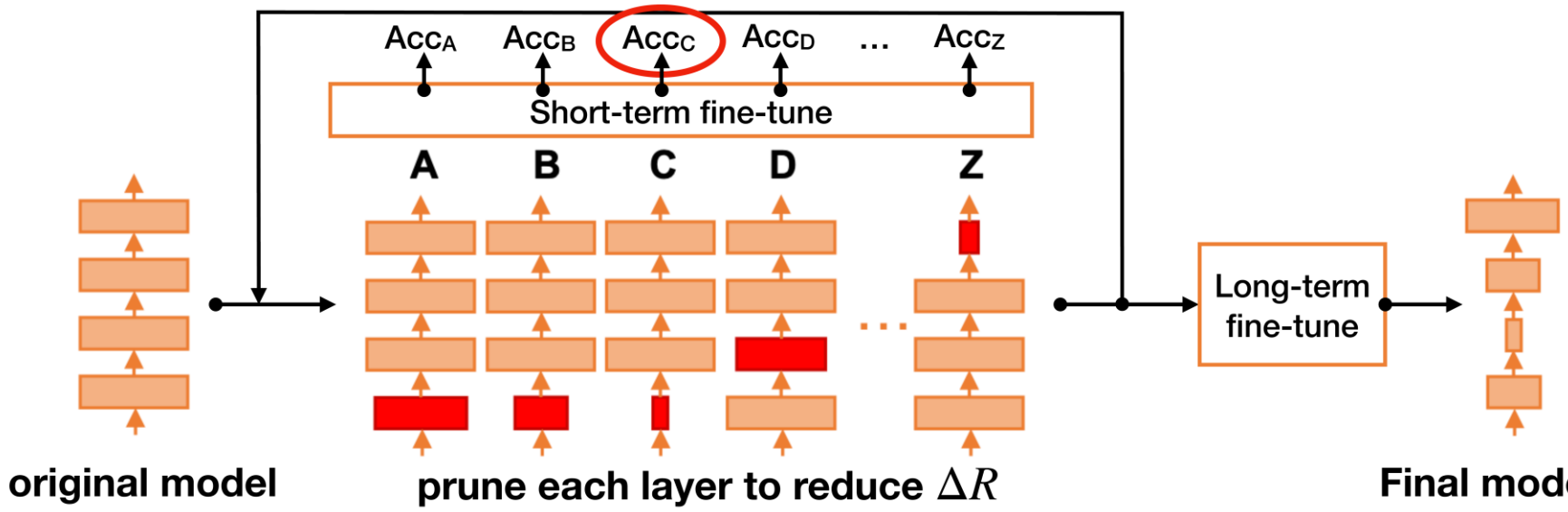
- A rule-based iterative/progressive method
- The goal of NetAdapt is to find a per-layer pruning ratio to meet a global resource constraint
  - e.g., latency, energy, …
- The process is done iteratively
- Take latency constraint as an example



Yang, T. J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., ... & Adam, H. (2018). Netadapt: Platform-aware neural network adaptation for mobile applications. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 285-300).

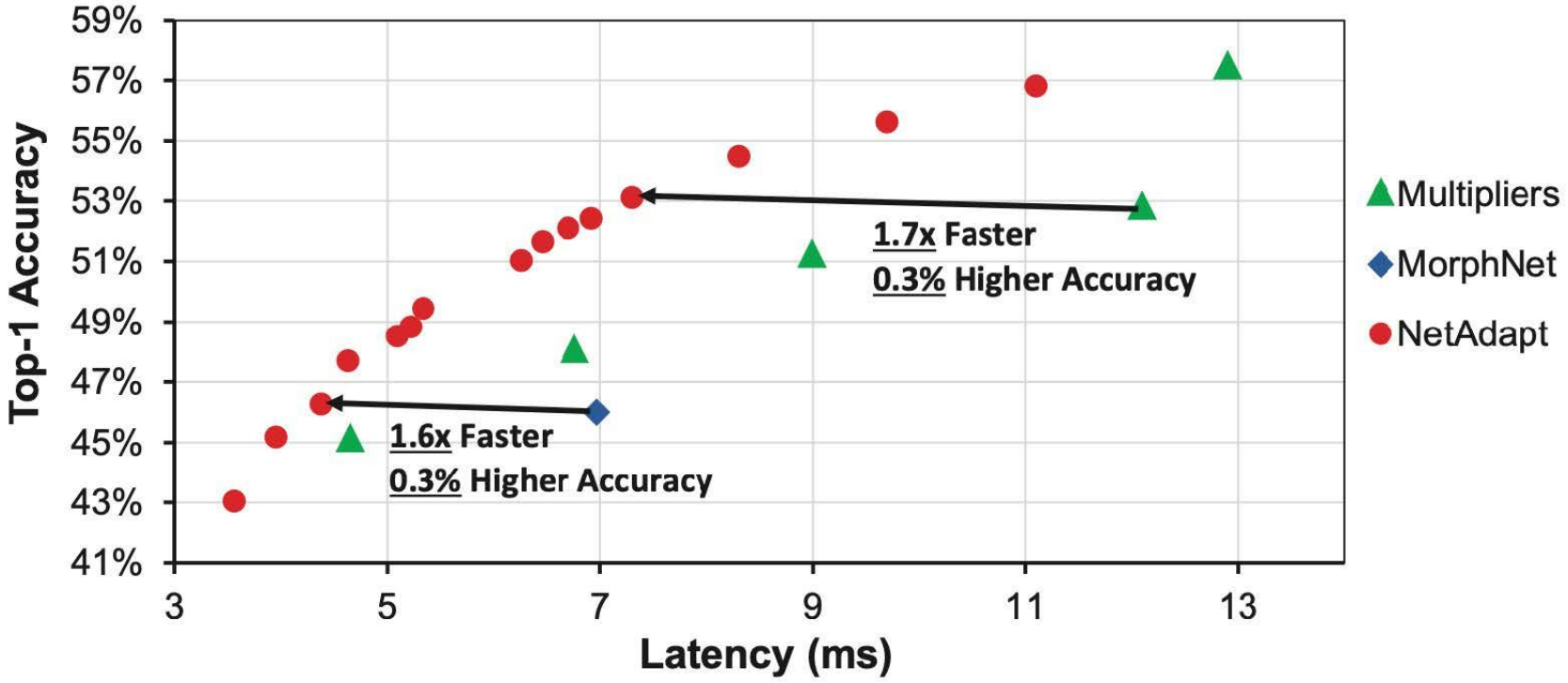
NetAdapt

- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)
  - For each layer L_k (in A-Z in the figure)
    - Prune the layer s.t. the latency reduction meets ΔR (based on a pre-built lookup table)
    - Short-term fine-tune model (10k iterations); measure accuracy after fine-tuning
  - Choose and prune the layer with the highest accuracy
- Repeat until the total latency reduction satisfies the constraint
- Long-term fine-tune to recover accuracy



NetAdapt

- The iterative nature allows us to obtain **a serial of models** with different costs
  - # of models = # of iterations

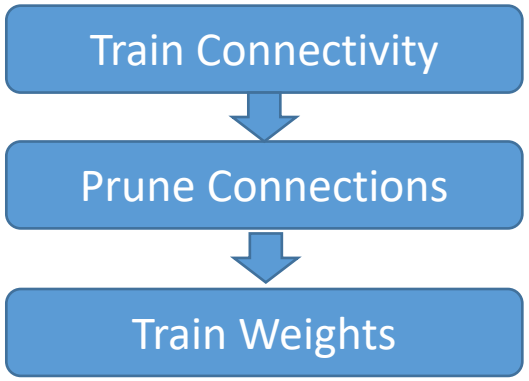


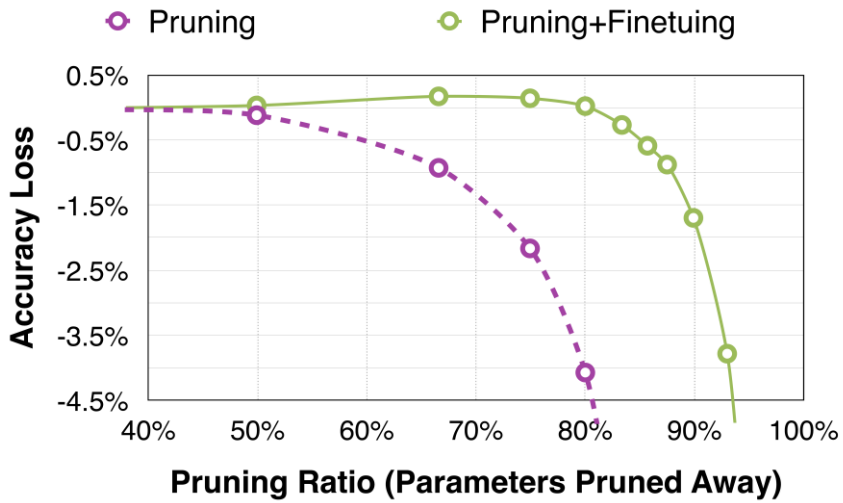
Outline

- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

Finetuning Pruned Neural Networks

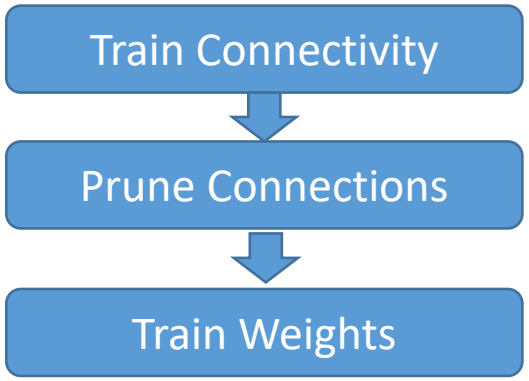
- After pruning, the model may decrease
  - Especially for larger pruning ratio
- Fine-tuning the pruned neural networks will help recover the accuracy and push the pruning ratio higher
  - Learning rate for fine-tuning is usually 1/100 or 1/10 of the original learning rate

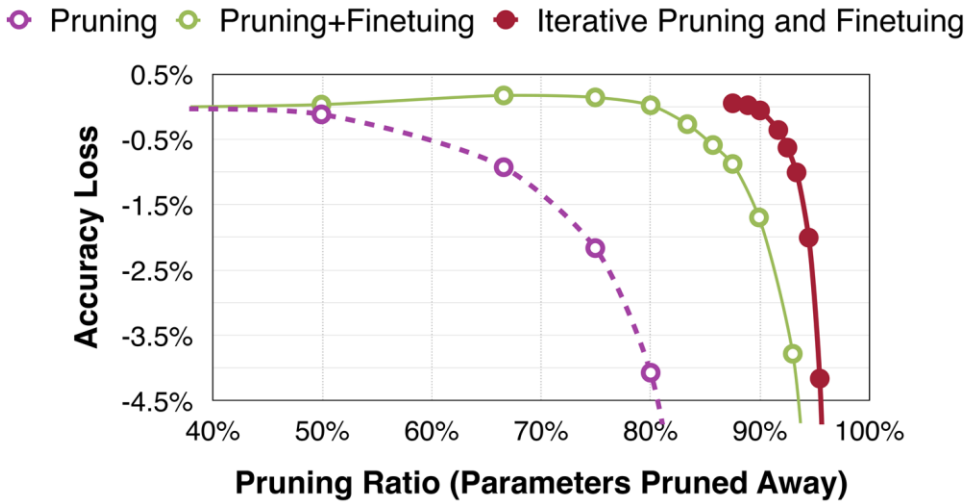




Finetuning Pruned Neural Networks

- Consider pruning followed by a fine-tuning is one iteration
- Iterative pruning gradually increases the target sparsity in each iteration
  - Boost pruning ratio from 5x to 9x on AlexNet compared to single-step aggressive pruning





Regularization

- When training neural networks or fine-tuning quantized neural networks, regularization is added to the loss term to
  - Penalize non-zero parameters
  - Encourage smaller parameters
- The most common regularization for improving performance of pruning is L1/L2 regularization
  - L1-Regularization

$$L' = L(x; W) + \lambda |W|$$

  - L2-Regularization

$$L' = L(x; W) + \lambda ||W||^2$$

  - Example
    - Magnitude-based Fine-grained Pruning applies L2 regularization on weights
    - Network Slimming applies smooth-L1 regularization on channel scaling factors

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision* (pp. 2736-2744).

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, *28*.

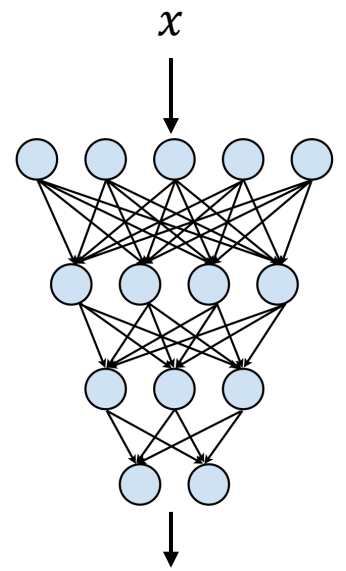
Neural Network Pruning

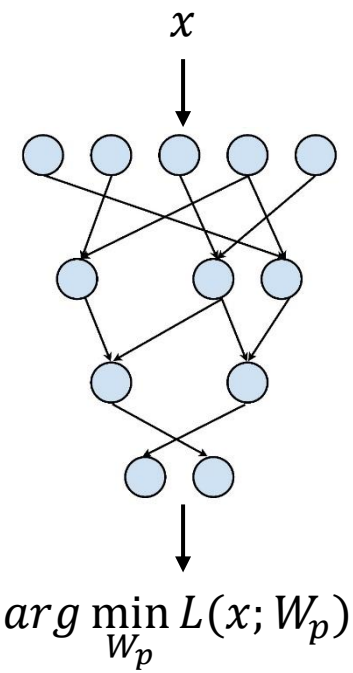
- In general, we could formulate the pruning as follows:

$$\arg\min_{w_p} L(x; W)$$

nonconvex problem
L is differentiable

  - Subject to $\|W_p\|_0 < N$ combinatorial constraint
  - L: the objective function for neural network training
  - X: input
  - W: original weights
  - W_p: pruned weights
  - $\|W_p\|_0$: calculates the # of nonzeros in, and is the target # of nonzeros





ADMM

- Rewrite the pruning problem in a form without constraint as

$$\arg\min_{W_p} L(x; W_p) + g(W_p)$$

  - Where $g(W_p) = \begin{cases} 0, & \|W_p\|_0 \le N \\ \infty, & otherwise \end{cases}$

- Split the variables and decompose the problem into simpler subproblems as

$$\arg\min_{W_p} L(x; W_p) + g(Z)$$

  - Subject to $W_p = Z$

- The **augmented Lagrangian** of the above problem is

$$\mathcal{L} = L(x; W_p) + g(Z) + tr\left[\Lambda^T (W_p - Z)\right] + \rho \|W_p - Z\|^2$$

$$\mathcal{L} = L(x; W_p) + g(Z) + \frac{\rho}{2} \|W_p - Z + U\|_2^2 - \frac{\rho}{2} \|U\|_2^2$$

ADMM

- The ADMM algorithm proceeds by repeating the following steps for $k = 0,1,\ldots,$ iterations
- Define

Differential gradient descent

L2 regularization

$$W_p^{(k+1)} := \arg\min_{W_p} \mathcal{L} = \arg\min_{W_p} L(x; W_p) + \frac{\rho}{2} \left\| W_p - Z^{(k)} + U^{(k)} \right\|_2^2$$

$$Z^{(k+1)} := \arg\min_{Z} \mathcal{L} = \arg\min_{Z} g(Z) + \frac{\rho}{2} \left\| W_p^{(k+1)} - Z + U^{(k)} \right\|_2^2$$

$$U^{(k+1)} := W_p^{(k+1)} - Z^{(k+1)} + U^{(k)}$$

- We can solve $Z^{(k+1)}$ analytically
  - Keep N elements of $W_p^{(k+1)} + U^{(k)}$ with the largest magnitudes and set the rest to zero

$$Z^{(k+1)} := \arg\min_{Z} \left\| W_p^{(k+1)} + U^{(k)} - Z \right\|_2^2$$

    - S.t. $\|Z\|_0 \leq N$

Outline

- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

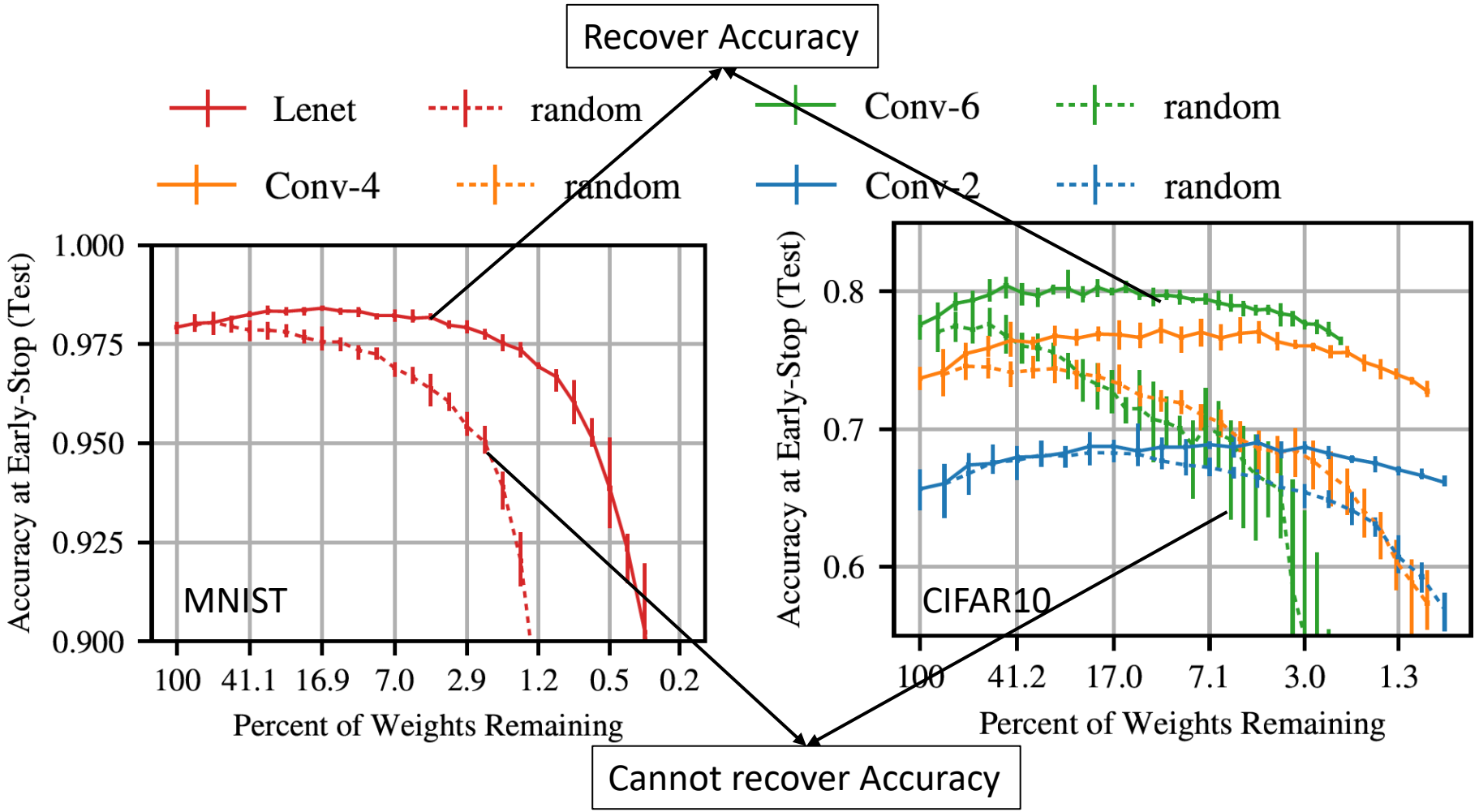
Train Sparse Neural Network From Scratch

- Neural Network Pruning shows that
  - A neural network can be reduced in size

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

Pruning from Scratch v.s Trained



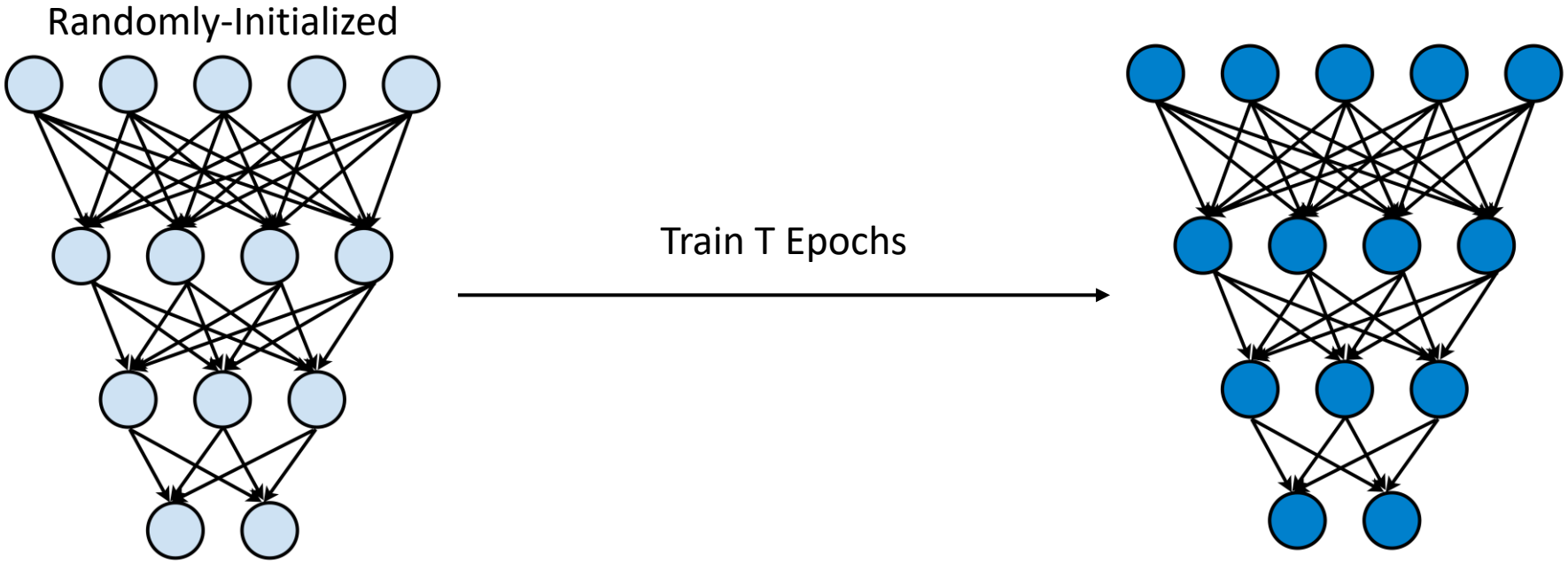
Dashed Lines: randomly sampled sparse networks
Solid Lines: correct sparsity mask (found by training) & same initialization.

J. Frankle and M. Carbin, 'The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks', in *International Conference on Learning Representations*, 2019.

The Lottery Ticket Hypothesis

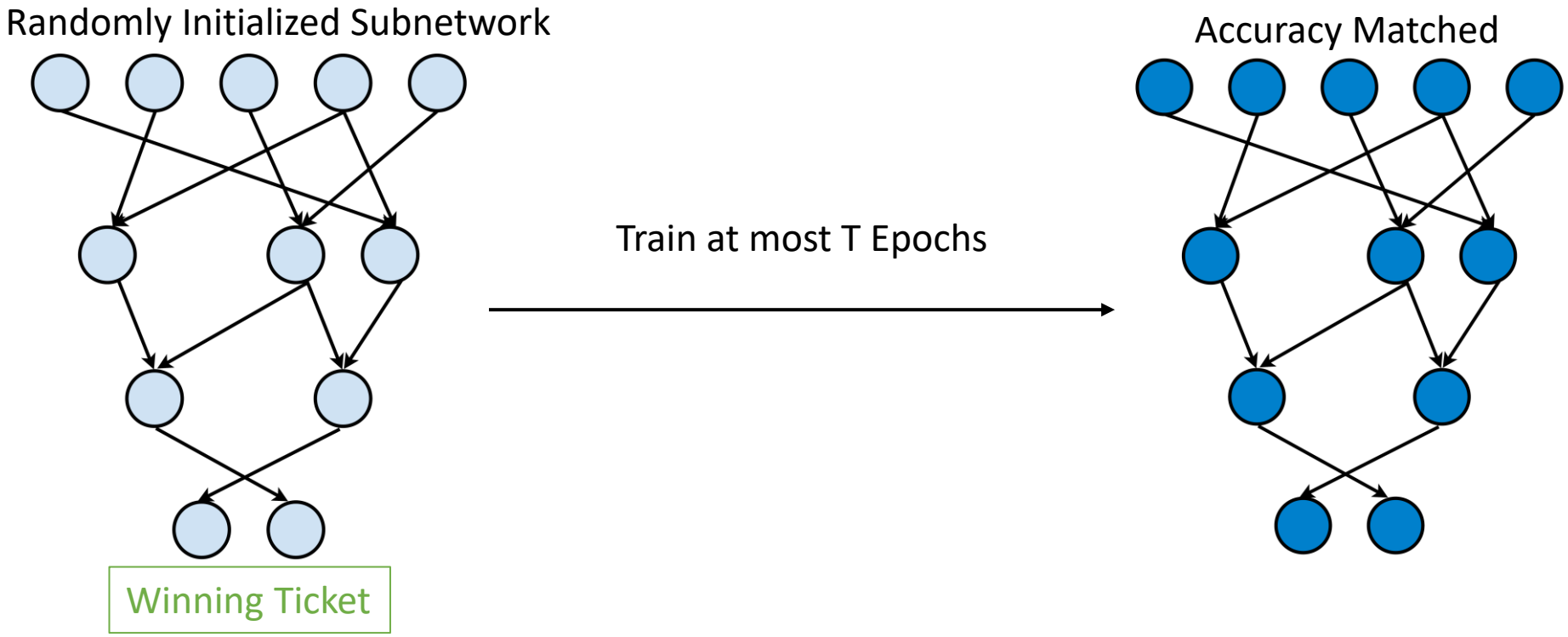
"A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations."

- The Lottery Ticket Hypothesis

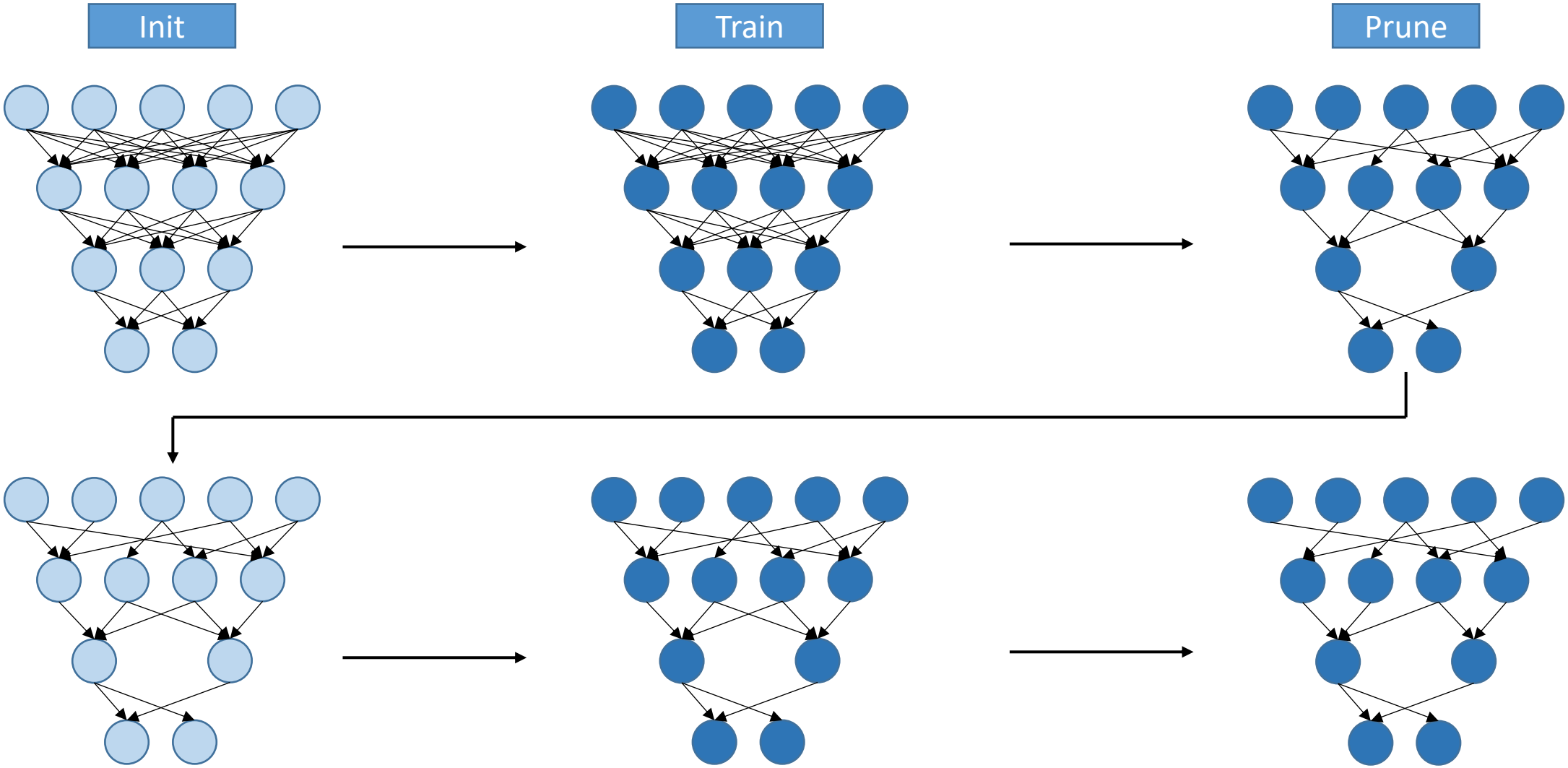


The Lottery Ticket Hypothesis

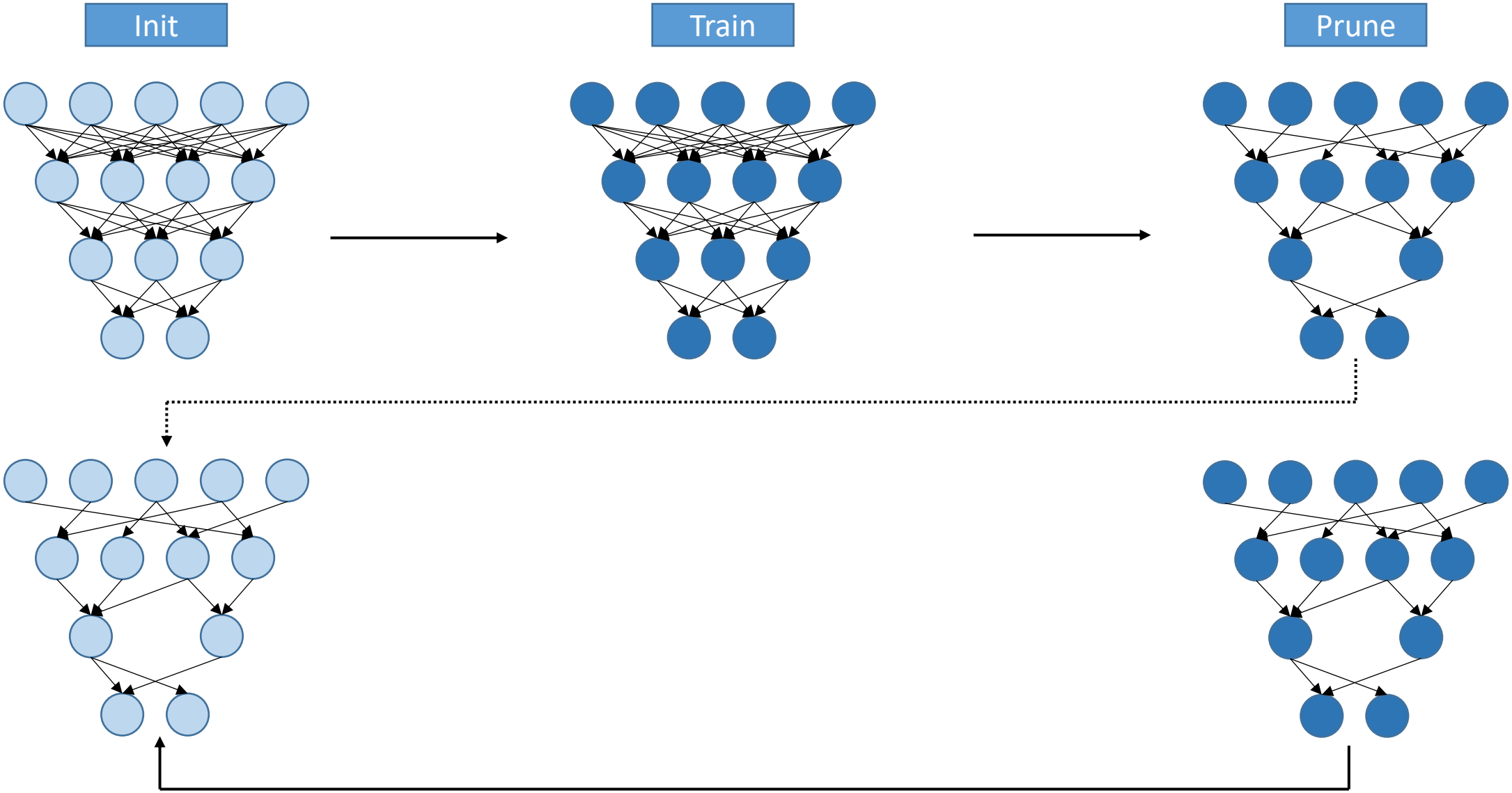
- “A randomly-initialized, dense neural network contains **a subnetwork** that is initialized such that—when **trained in isolation**—it can **match the test accuracy** of the original network after training for **at most the same number of iterations**.” - **The Lottery Ticket Hypothesis**



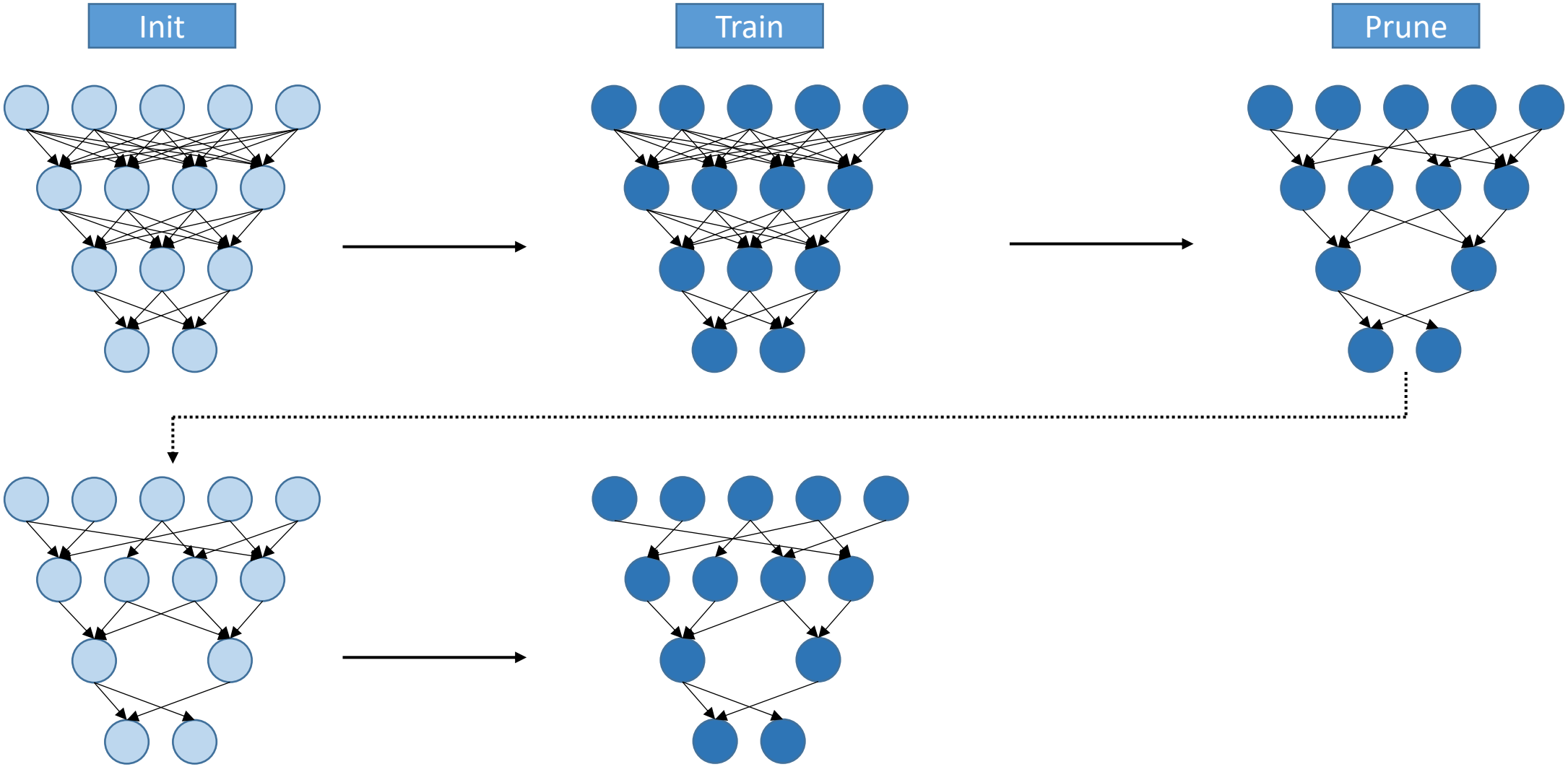
Iterative Magnitude Pruning



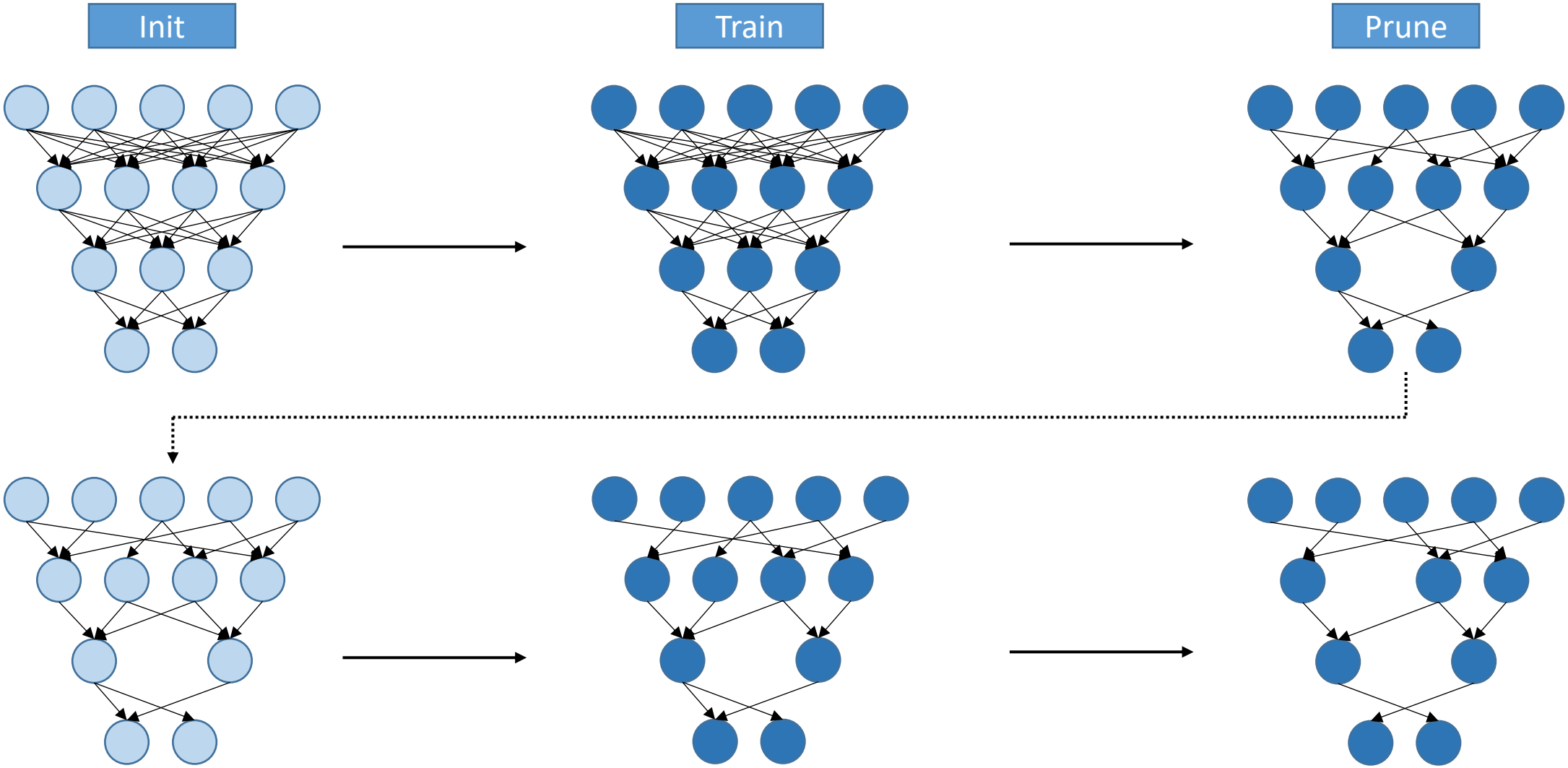
Iterative Magnitude Pruning



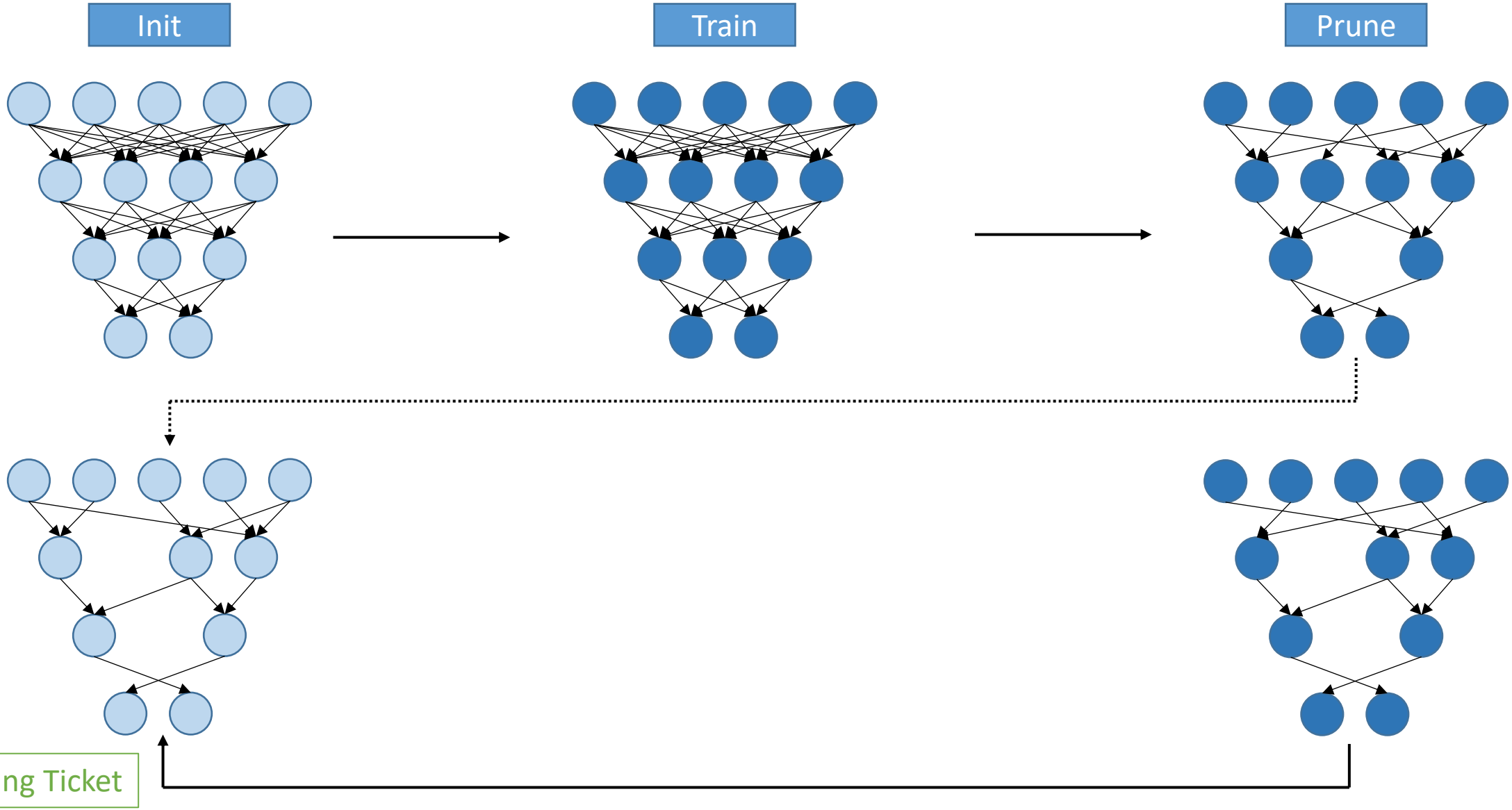
Iterative Magnitude Pruning



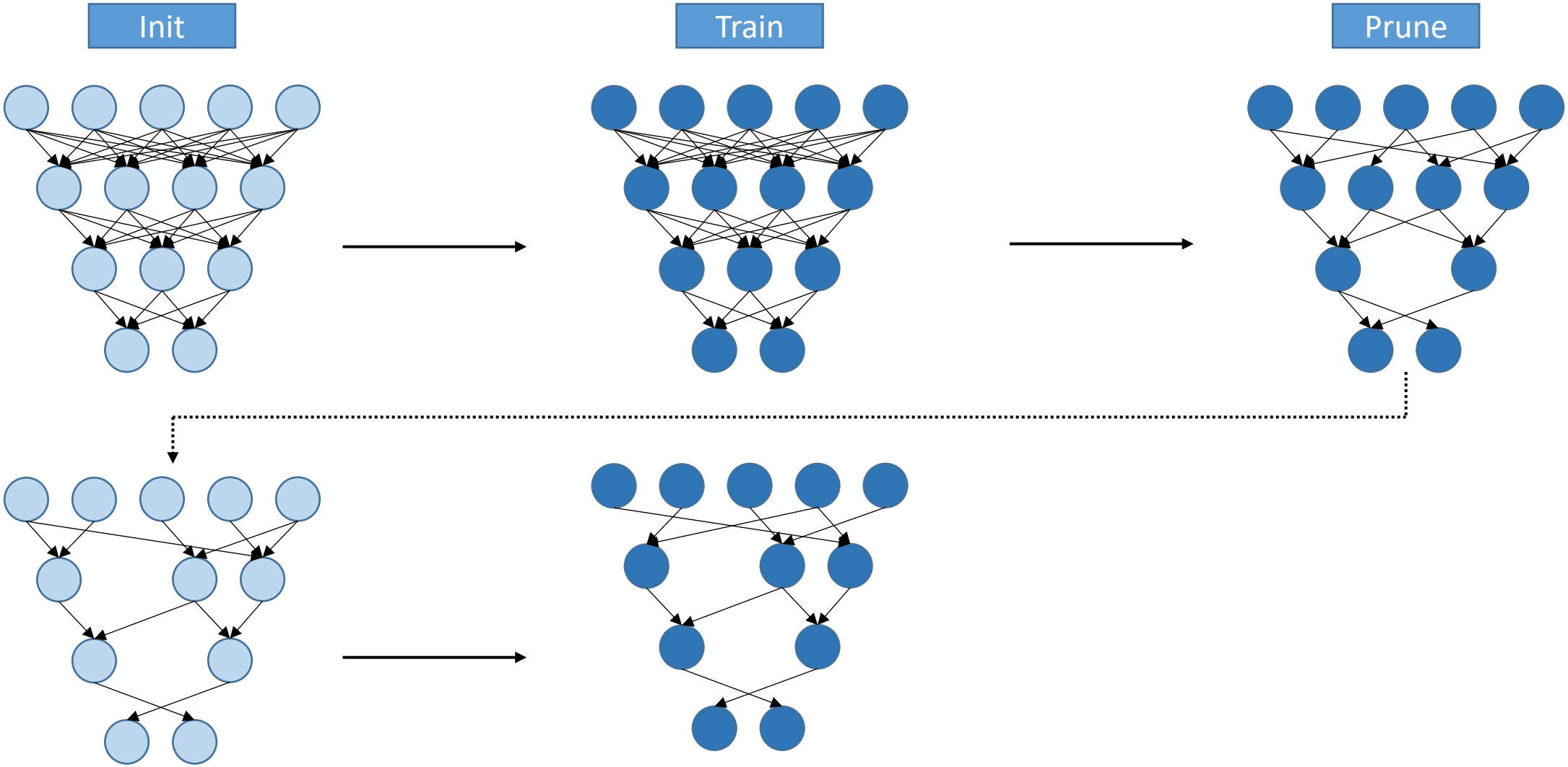
Iterative Magnitude Pruning



Iterative Magnitude Pruning

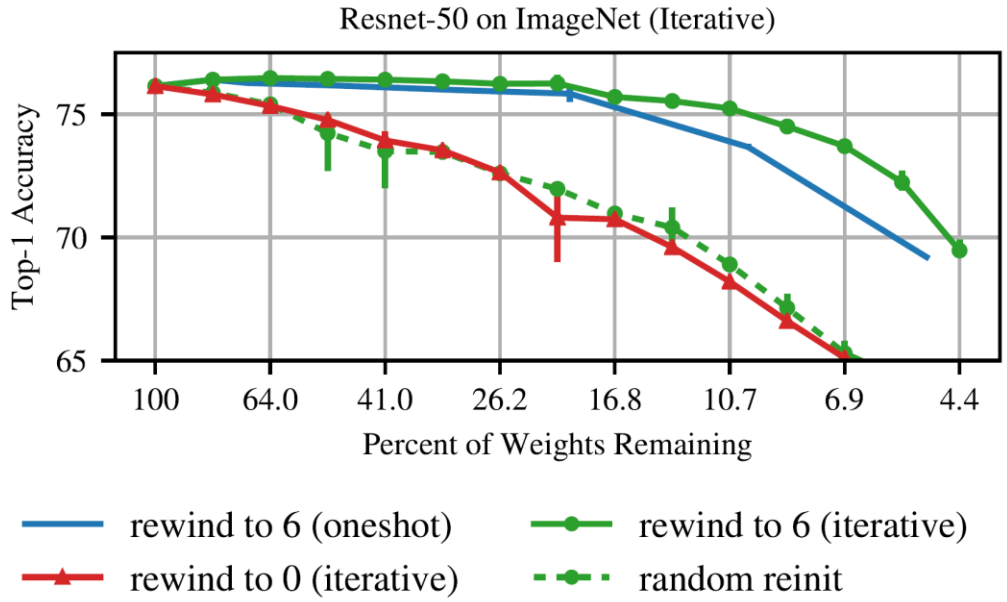


Iterative Magnitude Pruning



Scaling Limitation

- Resetting the weights to the very initial value $W_{t=0}$
  - Works for small-scale tasks such as MNIST and CIFAR-10
  - Fails on deep networks
- Instead, it is possible to robustly obtain pruned subnetworks by resetting the weights to the values after **a small number of k training iterations**, that is $W_{t=k}$



Total Training Epochs: T = 90
Rewinding to: k = 6

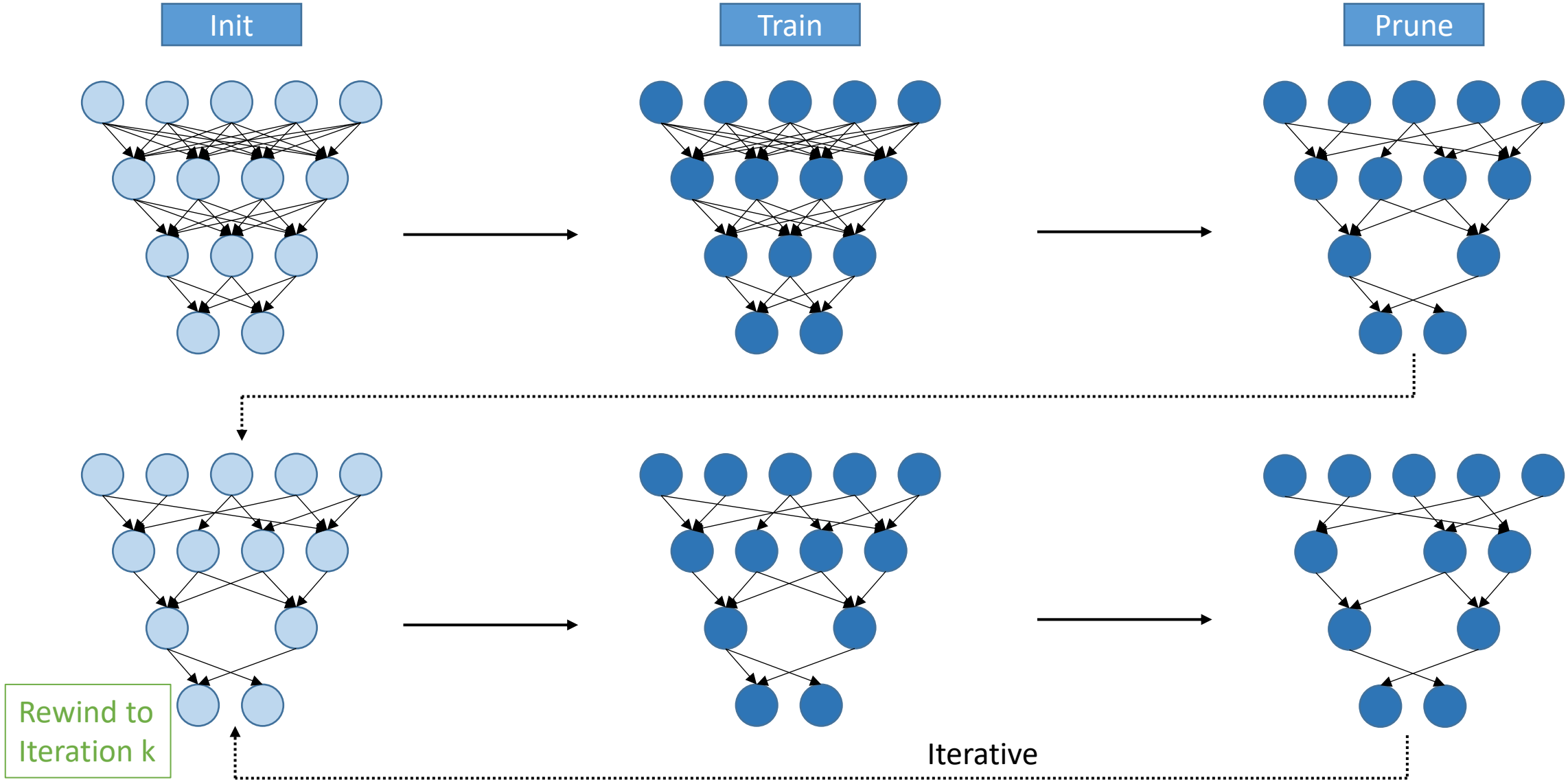
Frankle, J., Dziugaite, G. K., Roy, D. M., & Carbin, M. (2019). Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*.

Scaling Limitation

"Consider a dense, randomly-initialized neural network $f(x; W_0)$ that trains to accuracy a^* in T^* iterations. Let W_t be the weights at iteration t of training. There exist an iteration $k \ll T^*$ and fixed pruning mask $m \in \{0,1\}^{|W_0|}$ (where $\|m\|_1 \ll |W_0|$) such that subnetworks $m \odot W_k$ trains to accuracy $a \geq a^*$ in $T \leq T^* - \mathrm{k}$ iterations."

- *The Lottery Ticket Hypothesis with Rewinding*

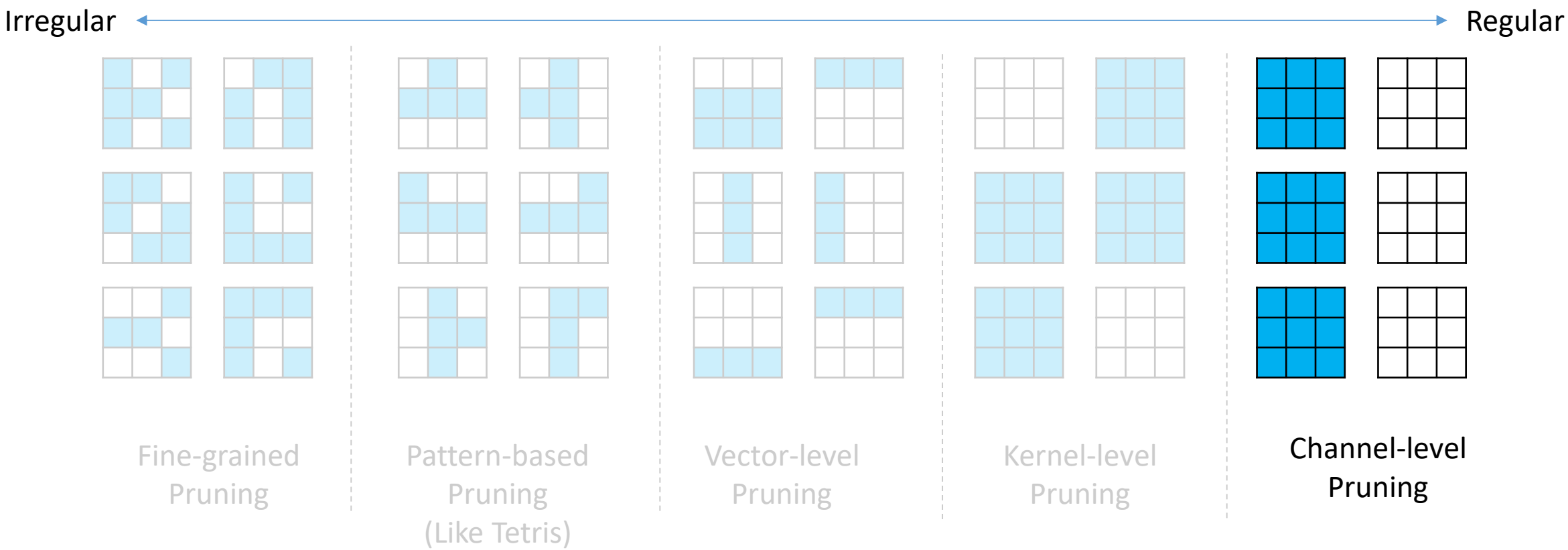
Iterative Magnitude Pruning with Rewinding



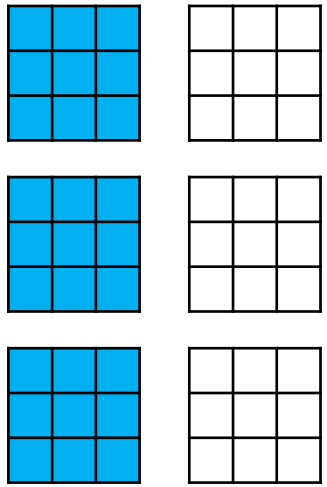
Outline

- Introduction
- Pruning Granularity
- Pruning Criterion
- Pruning Ratio
- Fine-tune/Training Pruned Network
- Lottery Ticket Hypothesis
- System Support for Sparsity

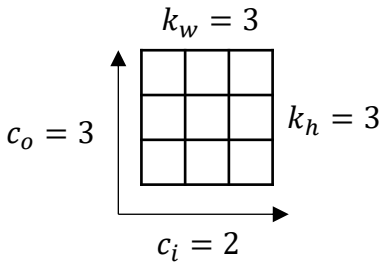
Different Pruning Granularity



Channel-level Pruning



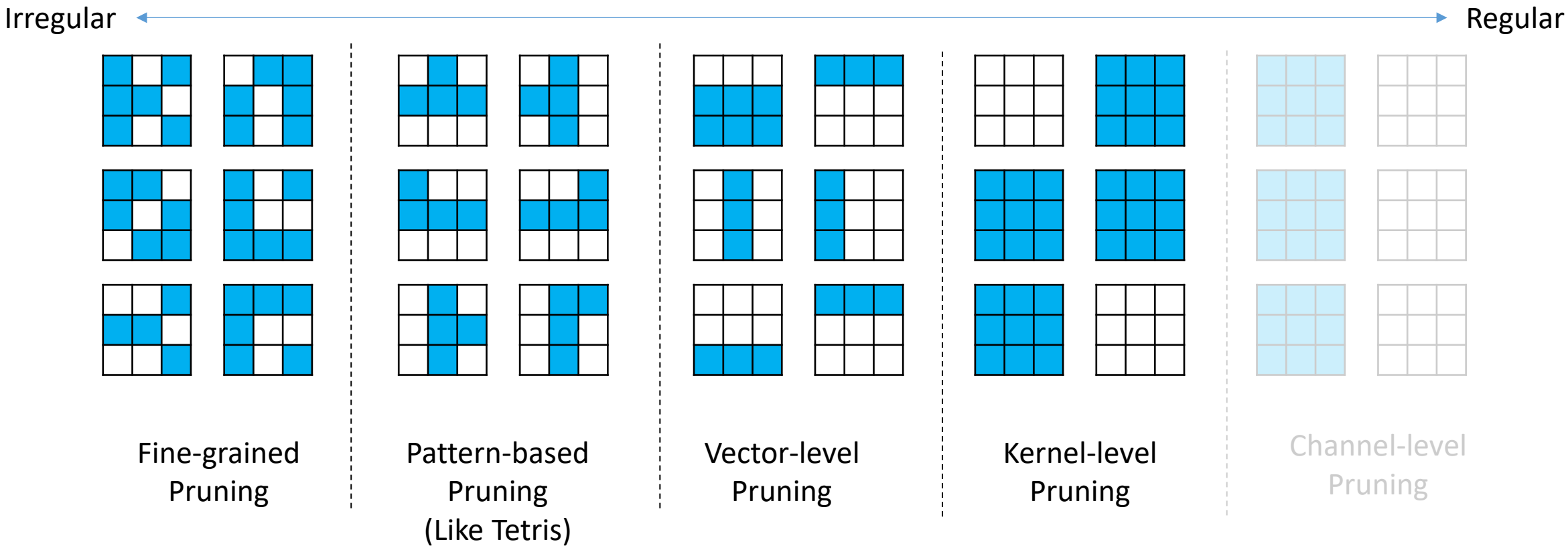
Channel-level Pruning



Original Implementation	Pruned
layer = nn.Conv2d(64, 64, kernel_size=3, padding=1, stride=1)	layer = nn.Conv2d(64, **32**, kernel_size=3, padding=1, stride=1)

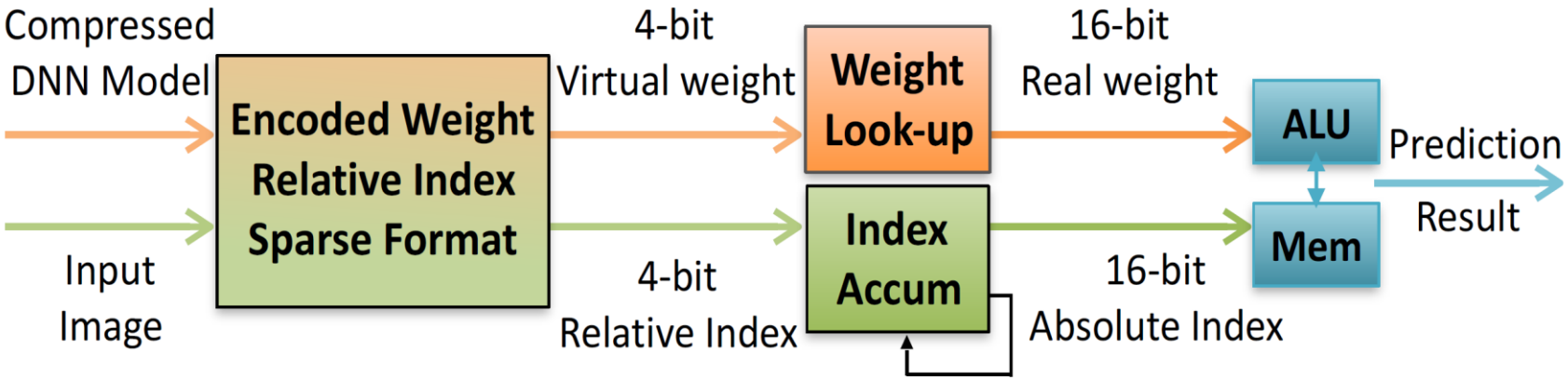
No need for specialized system support!

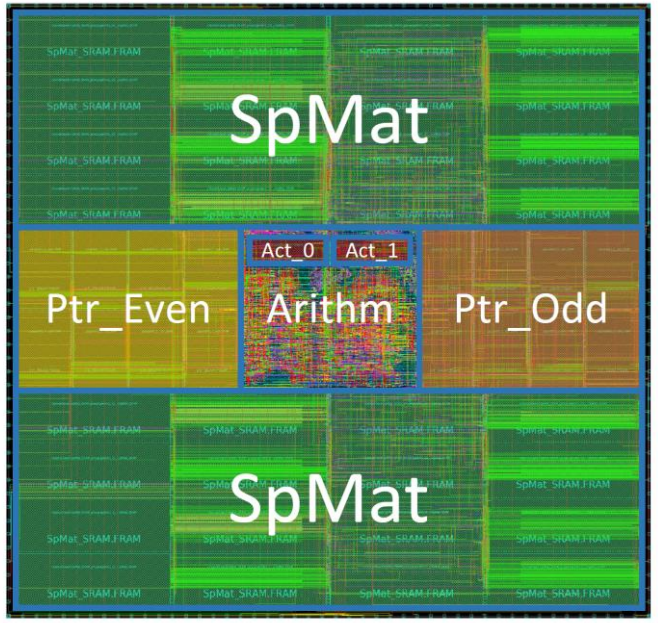
Different Pruning Granularity

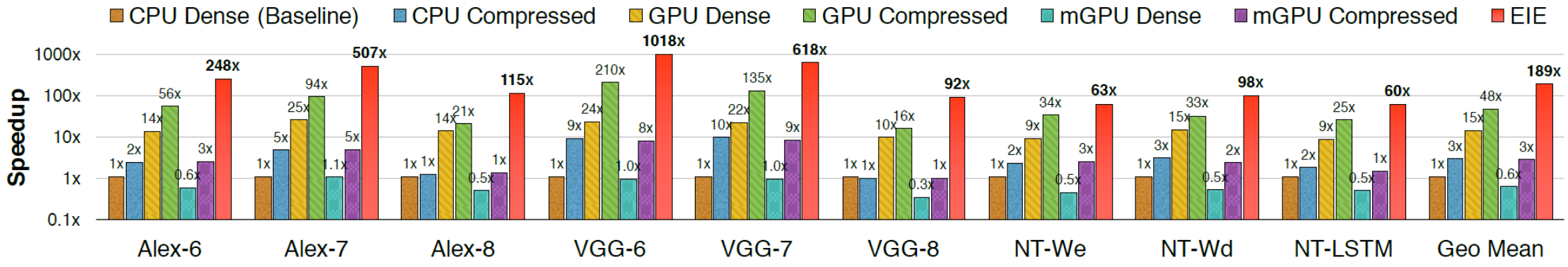


EIE: Efficient Inference Engine

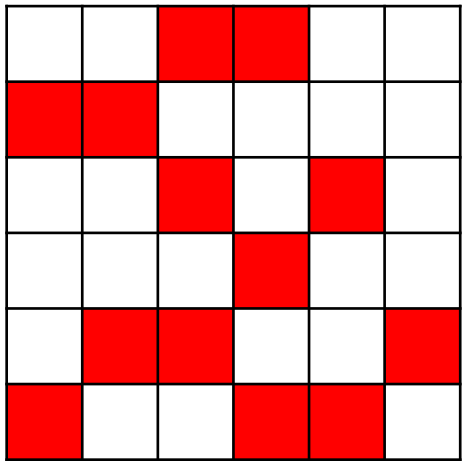
- DNN Accelerator for Sparse, Compressed Model





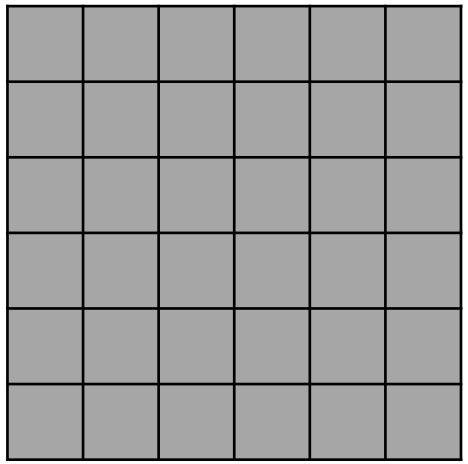


Fine-grained Pruning



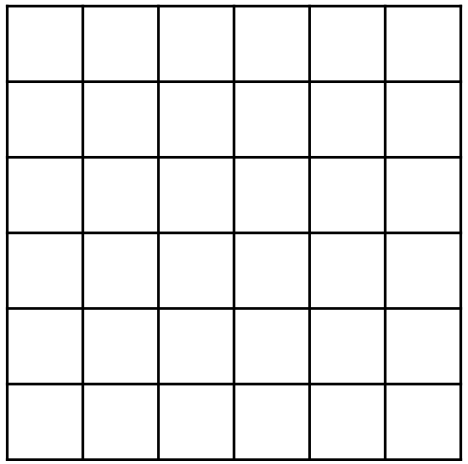
Weight

×



Input Activation

=



Output Activation

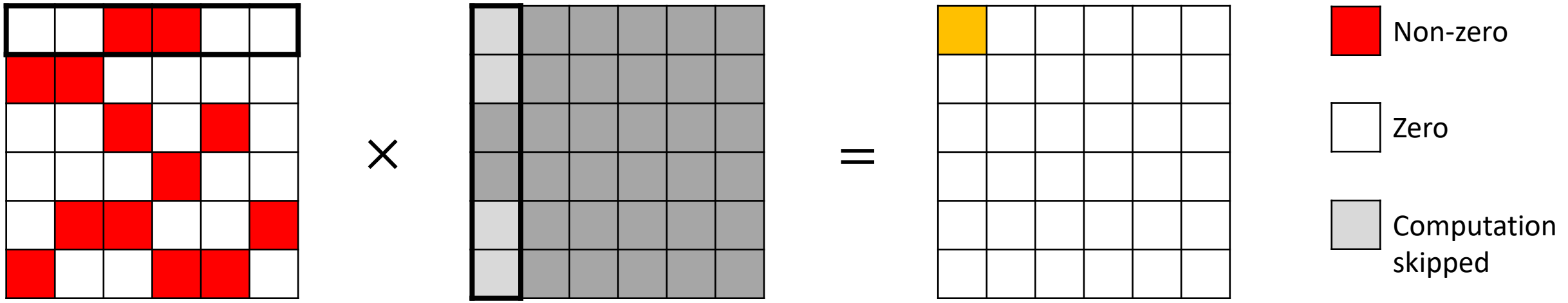
Non-zero

Zero

Computation skipped

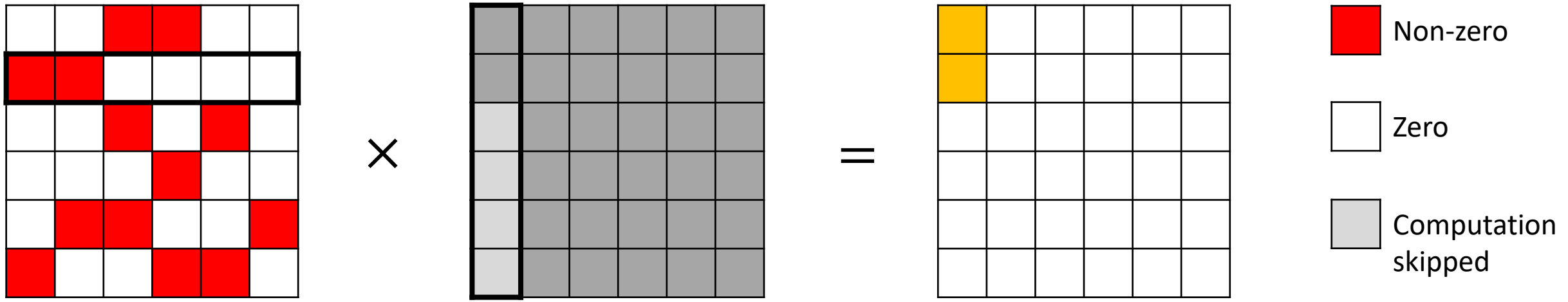
Zero x Any computation can be skipped!

Fine-grained Pruning



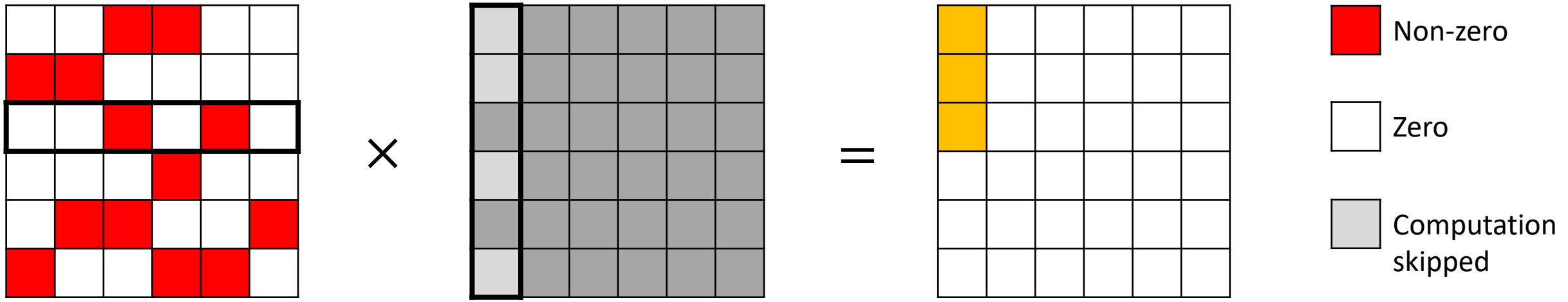
Zero x Any computation can be skipped!

Fine-grained Pruning



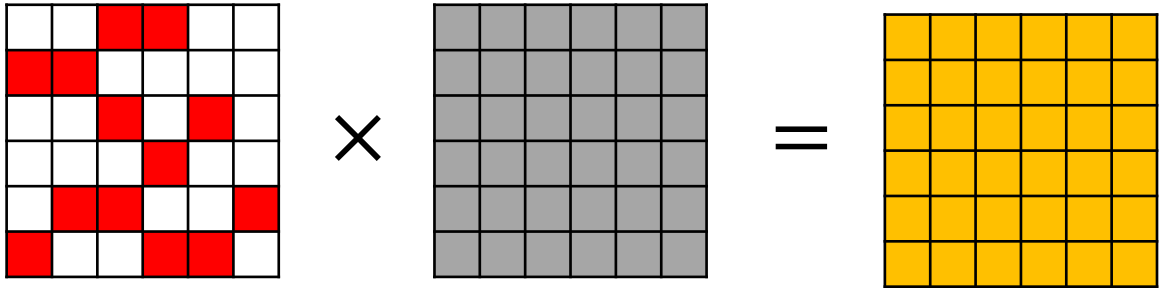
Zero x Any computation can be skipped!

Fine-grained Pruning



Zero x Any computation can be skipped!

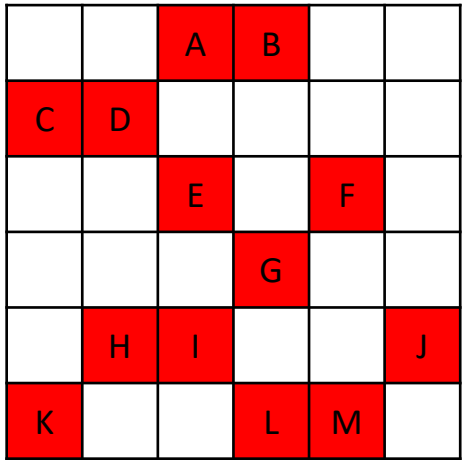
Sparse Matrix-Matrix Multiplication (SpMM)



Dense Matrix Multiplication	Sparse Matrix Multiplication

for m in range(M):
  for n in range(N):
  for k in range(A[m].size):
  C[m][n] = A[m][k] * B[k][n]
	for m in range(M):
	  for n in range(N):
	  for k in A[m].nonzeros:
	  C[m][n] = A[m][k] * B[k][n]
	
	  A[0].nonzeros = [2, 3]
	  A[1].nonzeros = [0, 1]
	  …
	  A[5].nonzeros = [0, 3, 4]

CSR Format for Sparse Matrices



0	2	4	6	7	10	13

Row Pointer

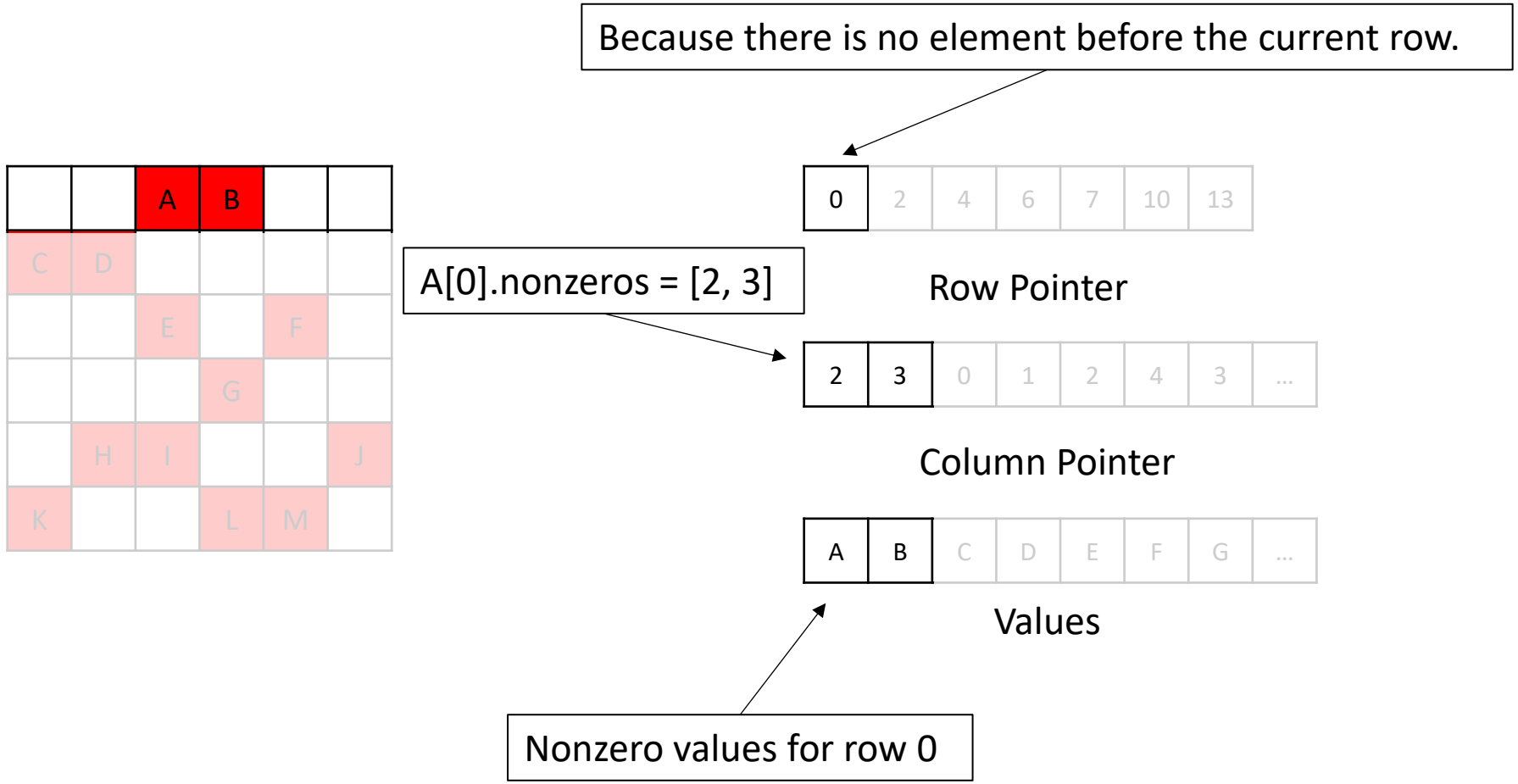
2	3	0	1	2	4	3	…

Column Pointer

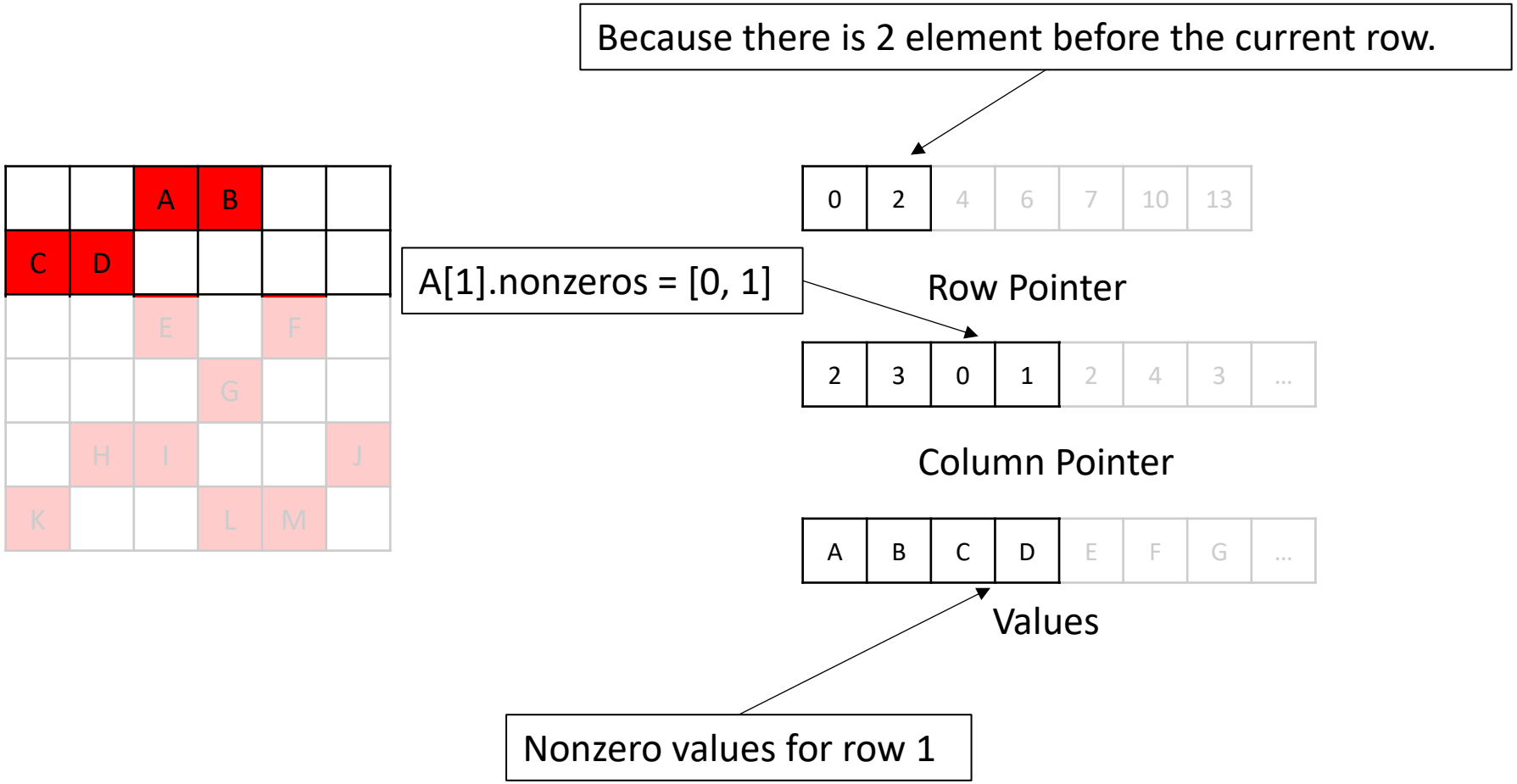
A	B	C	D	E	F	G	…

Values

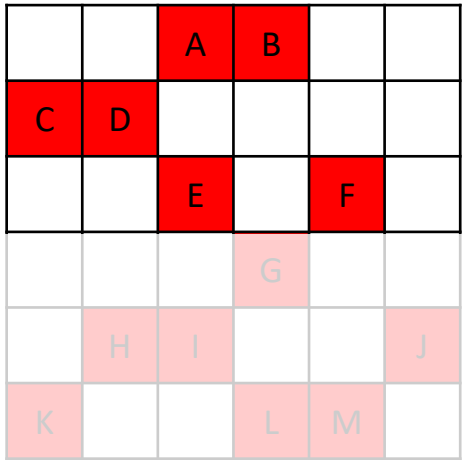
CSR Format for Sparse Matrices



CSR Format for Sparse Matrices



CSR Format for Sparse Matrices



Because there is 4 element before the current row.

0	2	4	6	7	10	13

Row Pointer

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

2	3	0	1	2	4	3	…

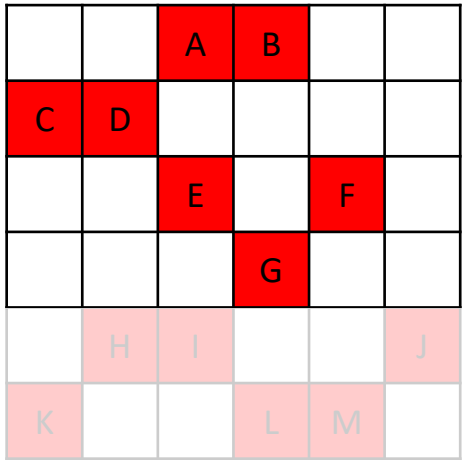
Column Pointer

A	B	C	D	E	F	G	…

Values

Nonzero values for row 2

CSR Format for Sparse Matrices



Because there is 6 element before the current row.

0	2	4	6	7	10	13

Row Pointer

A[3].nonzeros = [3]

2	3	0	1	2	4	3	…

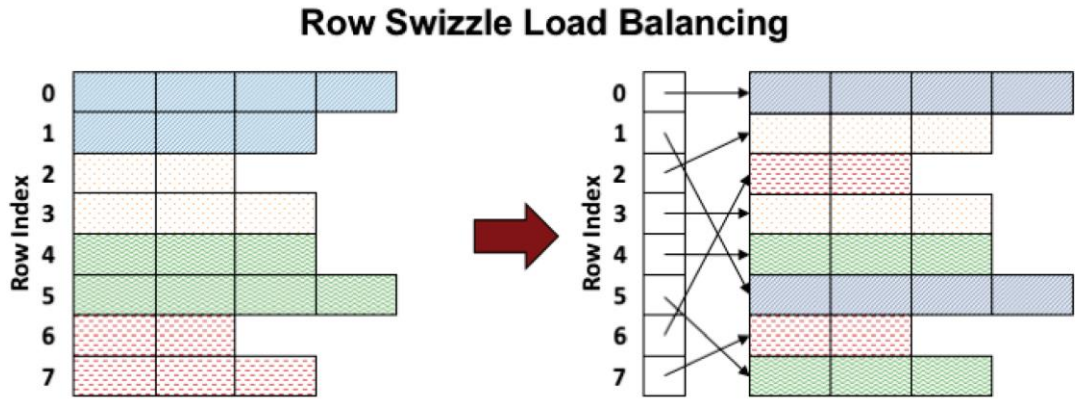
Column Pointer

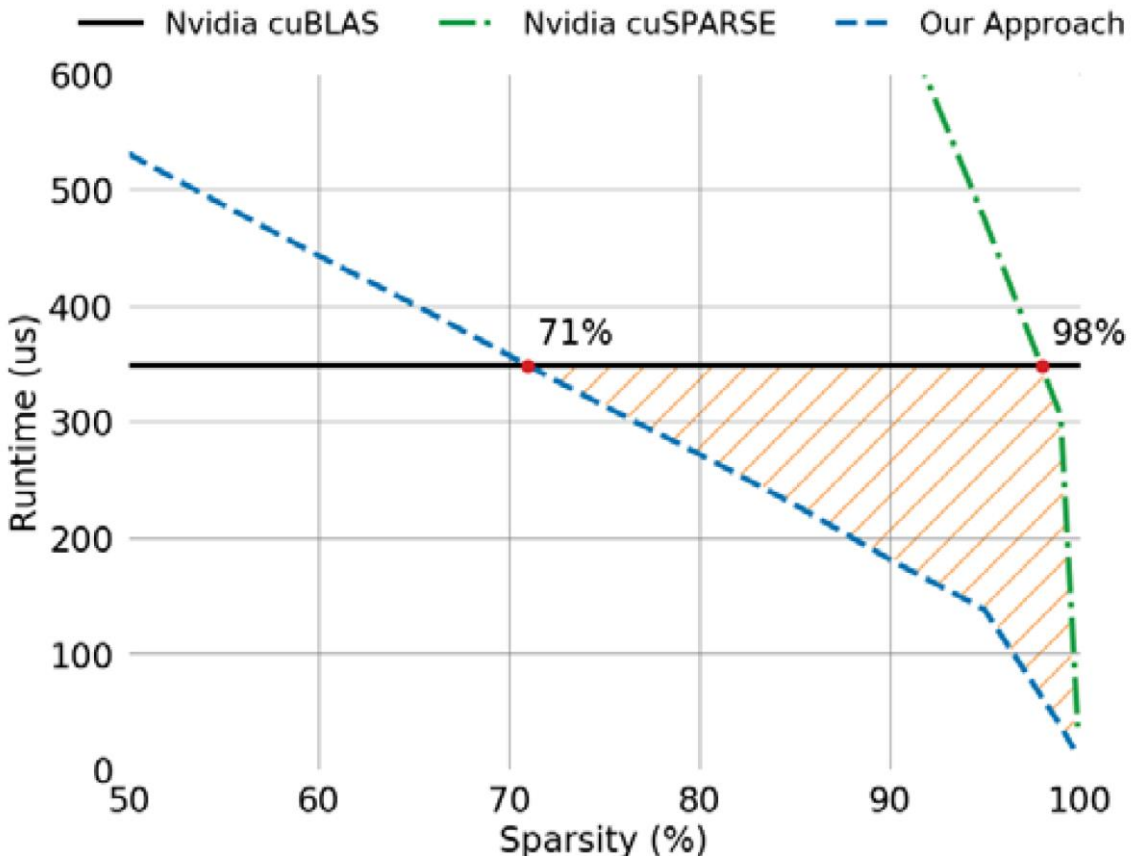
A	B	C	D	E	F	G	…

Values

Nonzero values for row 3

GPU Support for SpMM



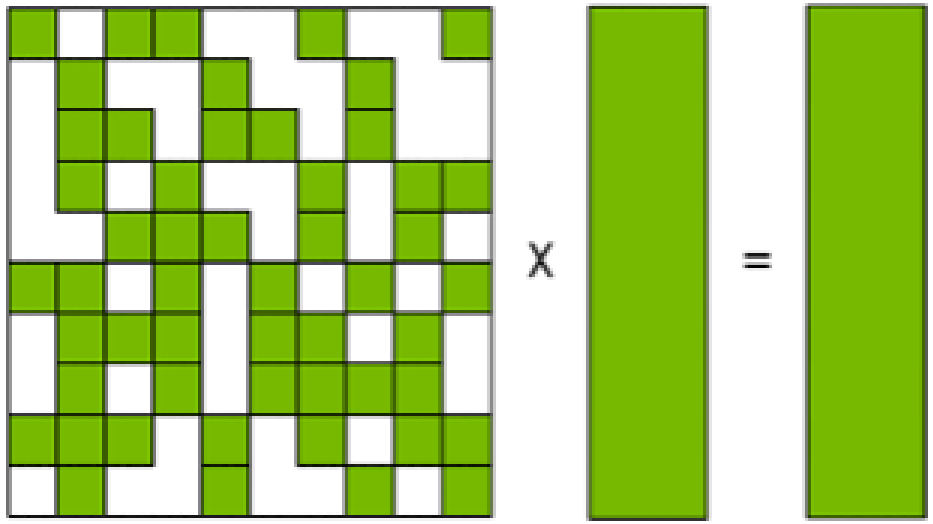


Load Balancing: each thread in the warp (the same color) has the same workload size

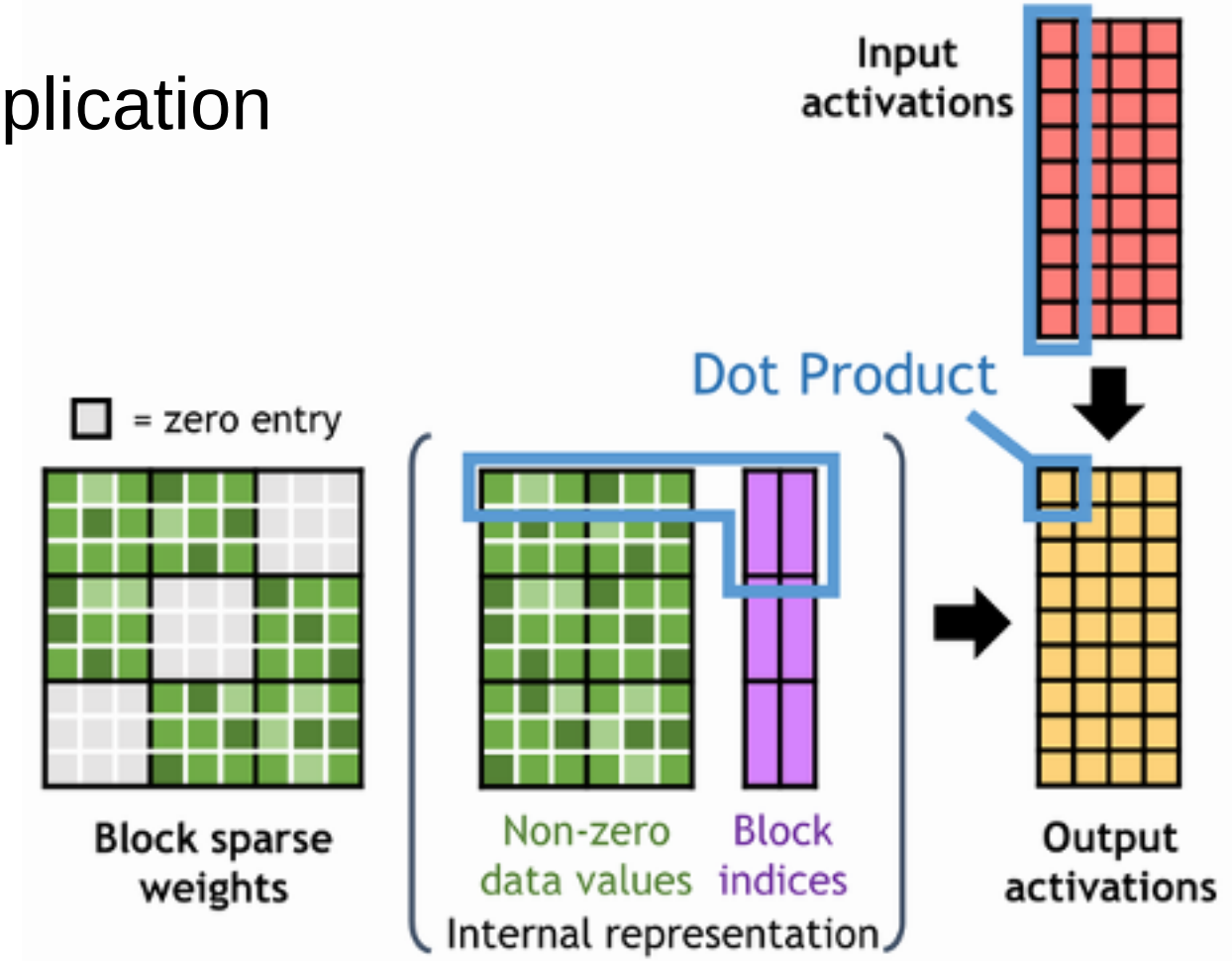
Gale, T., Zaharia, M., Young, C., & Elsen, E. (2020, November). Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1-14). IEEE.

Block SpMM

- Block Sparse Matrix-Matrix Multiplication



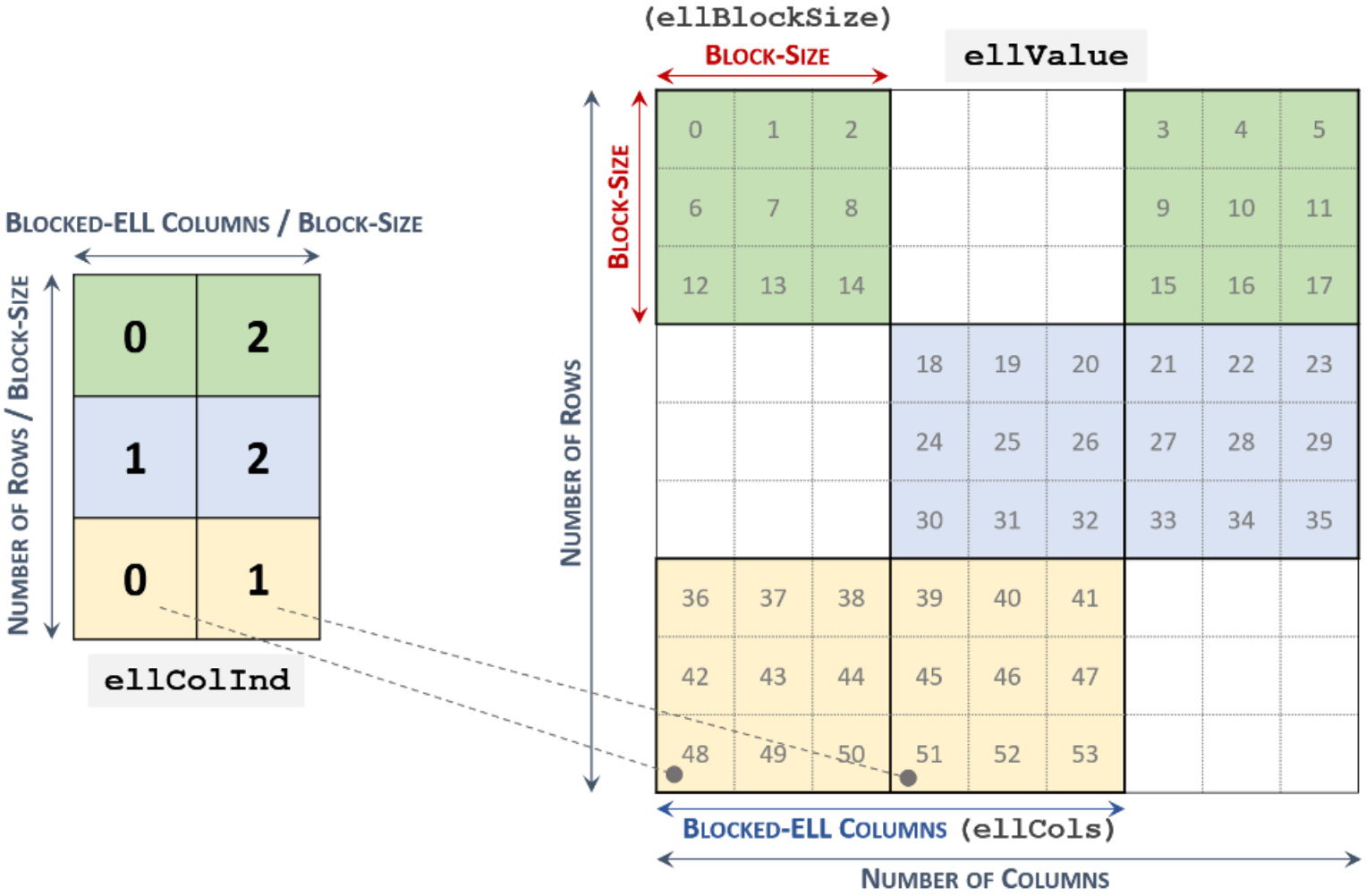
General SpMM: Irregular Sparsity



Block SpMM: **Structured** Sparsity

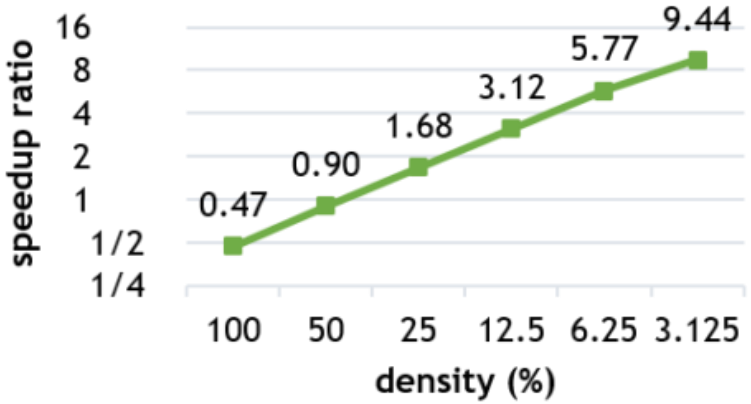
https://developer.nvidia.com/blog/accelerating-matrix-multiplication-with-block-sparse-format-and-nvidia-tensor-cores/

Blocked-Ellpack Format

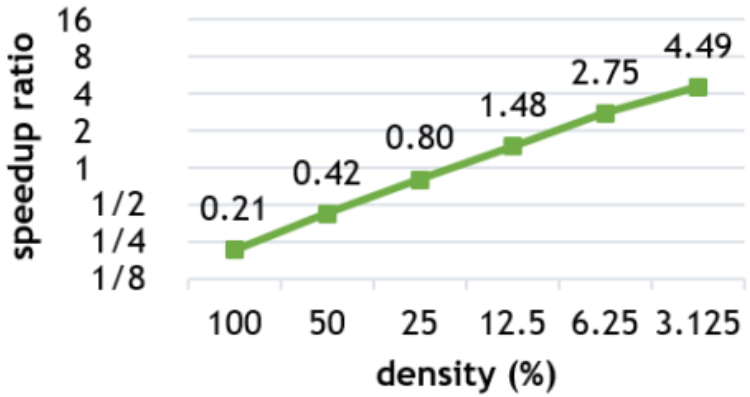


Recall the idea of CSR. There is an **index** array and a **value** array

Block SpMM - Results on NVIDA GPU

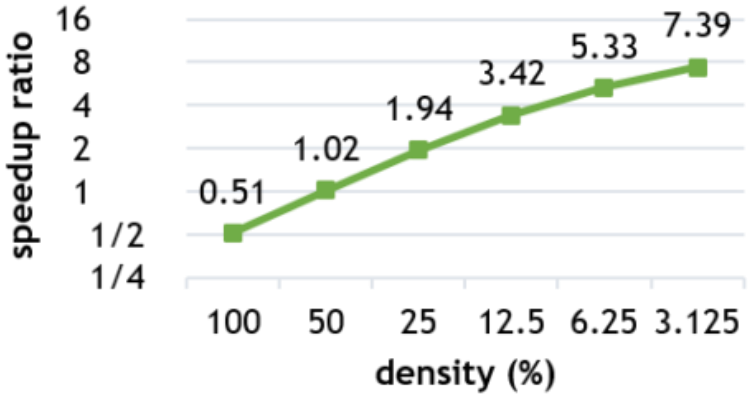


(a) block size = 32

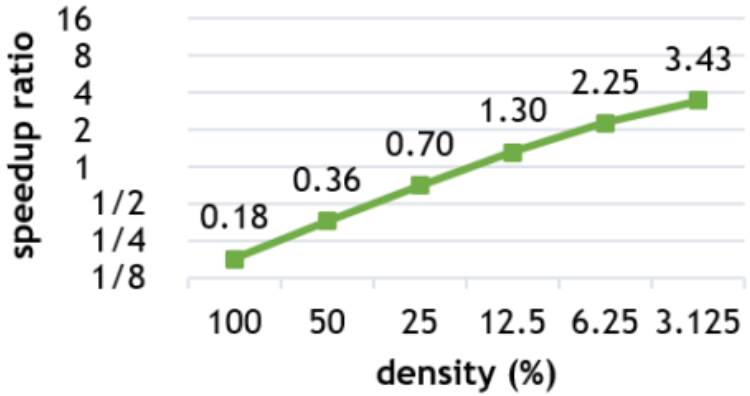


(b) block size = 16

Results on Tesla V100



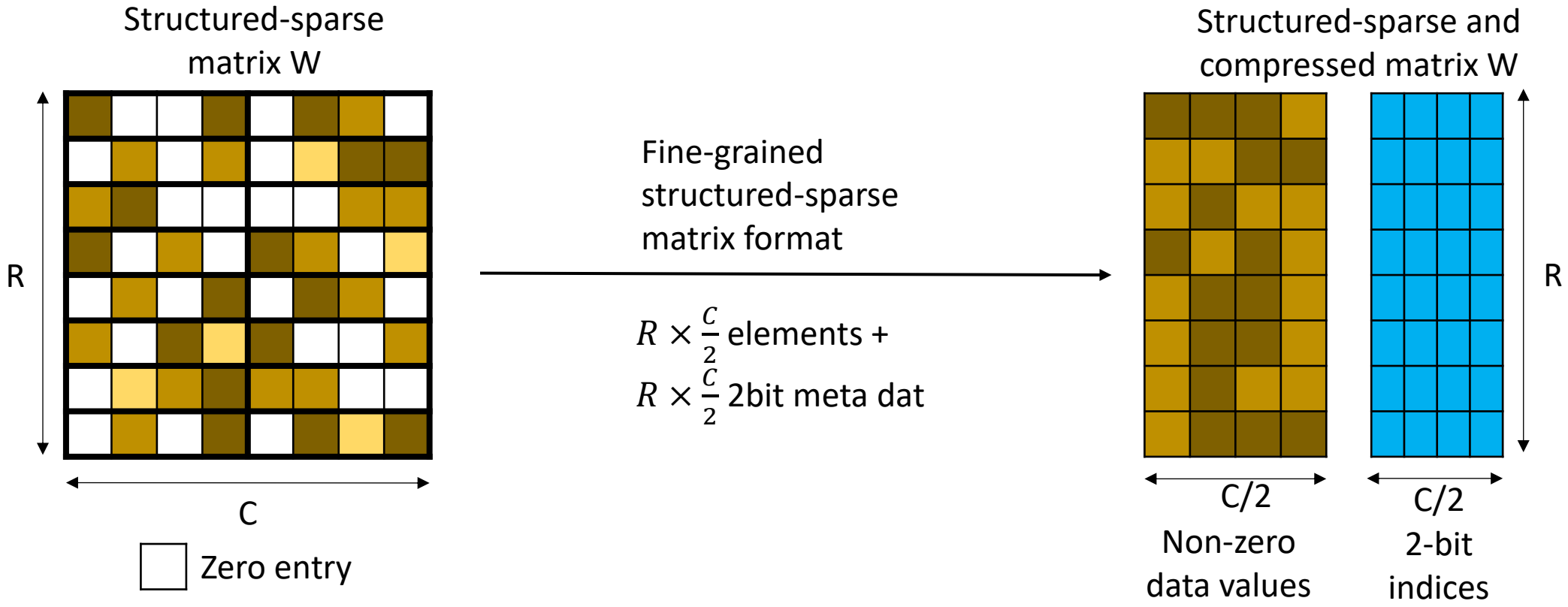
(a) block size = 32



(b) block size = 16

Results on Tesla A100

M:N Sparsity

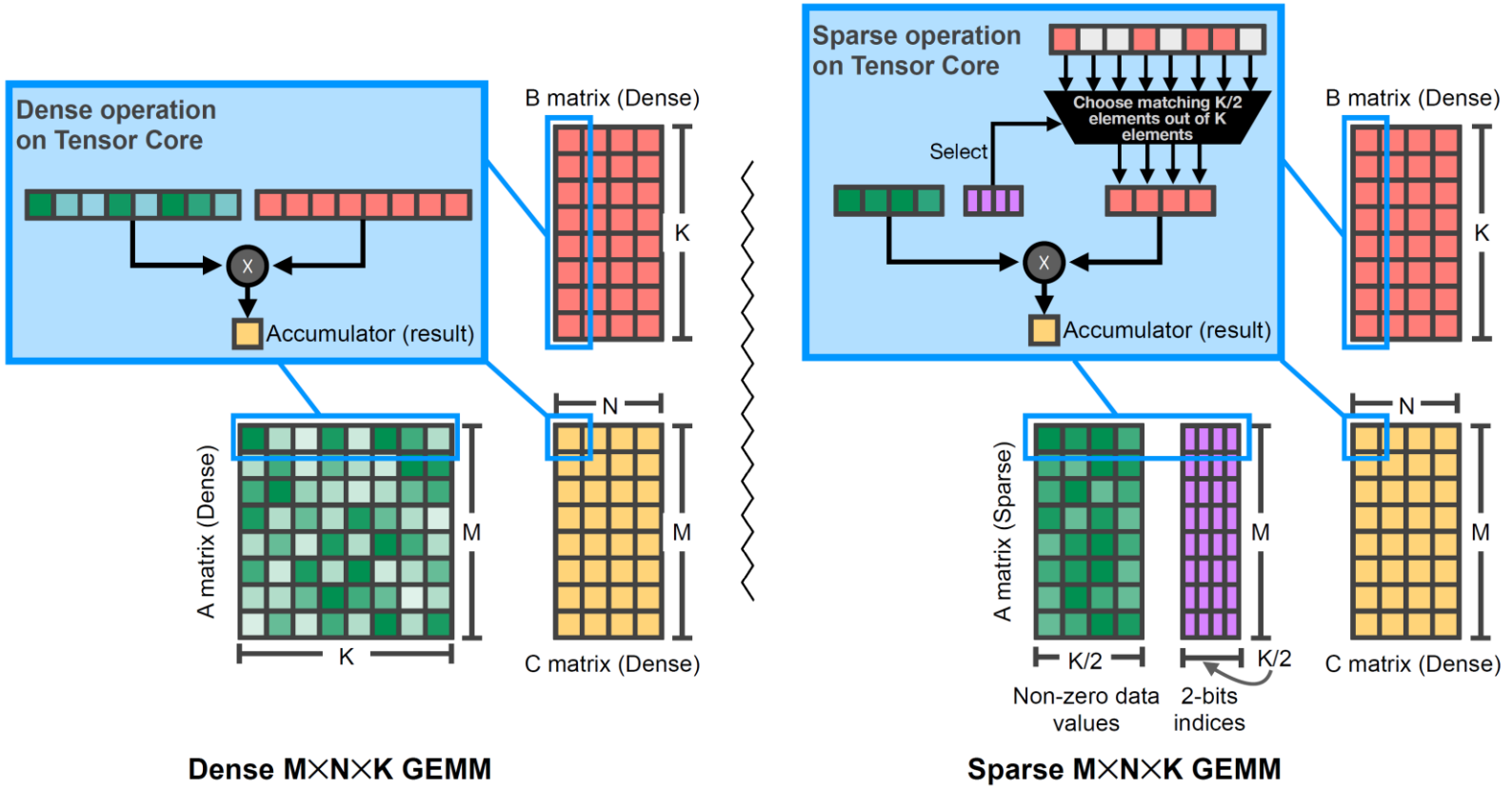


Two weights are non-zero out of four consecutive weights (2:4 sparsity

Push all the nonzero elements to the left in memory: save storage and computation.)

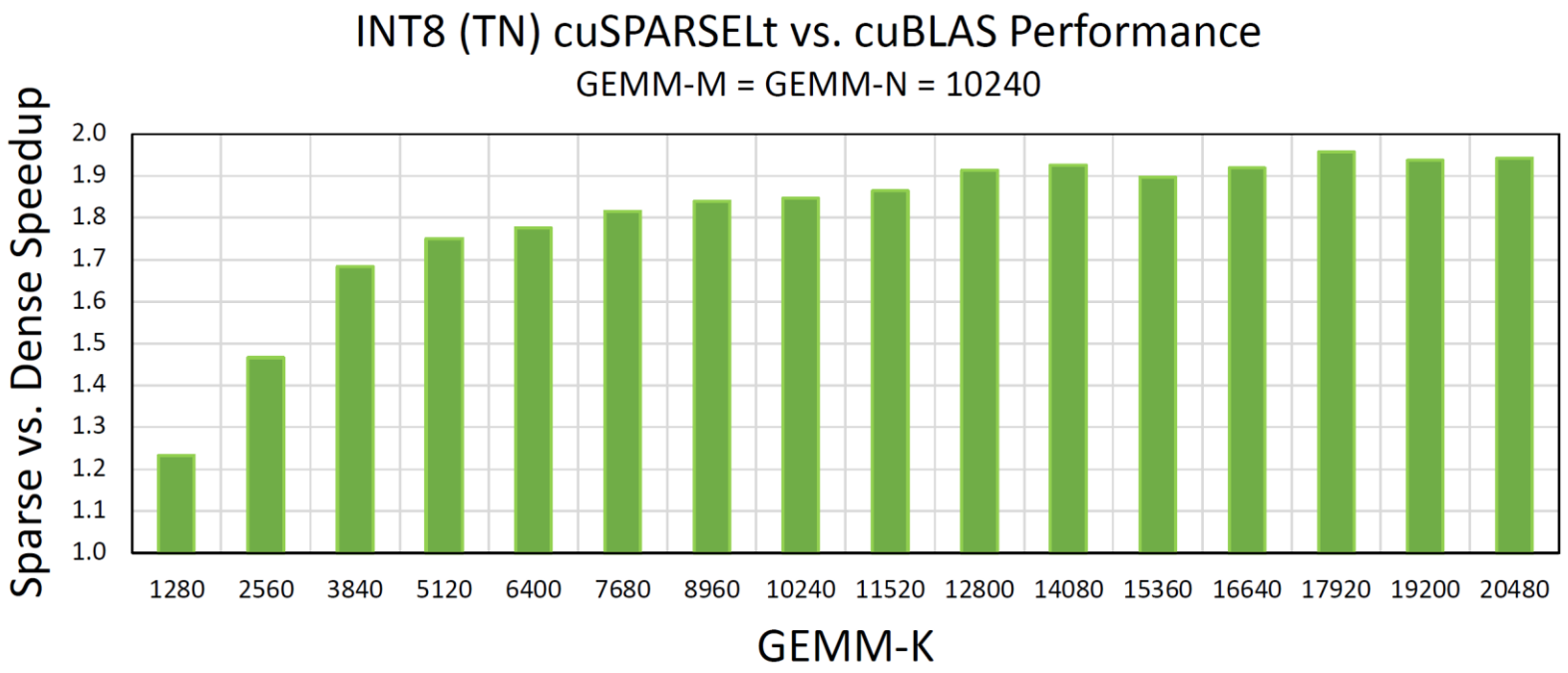
System Support for M:N Sparsity

- Mapping M:N sparse matrices onto NVIDIA tensor cores



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

Comparison of Sparse and Dense INT8 GEMMs on NVIDIA A100 Tensor Cores

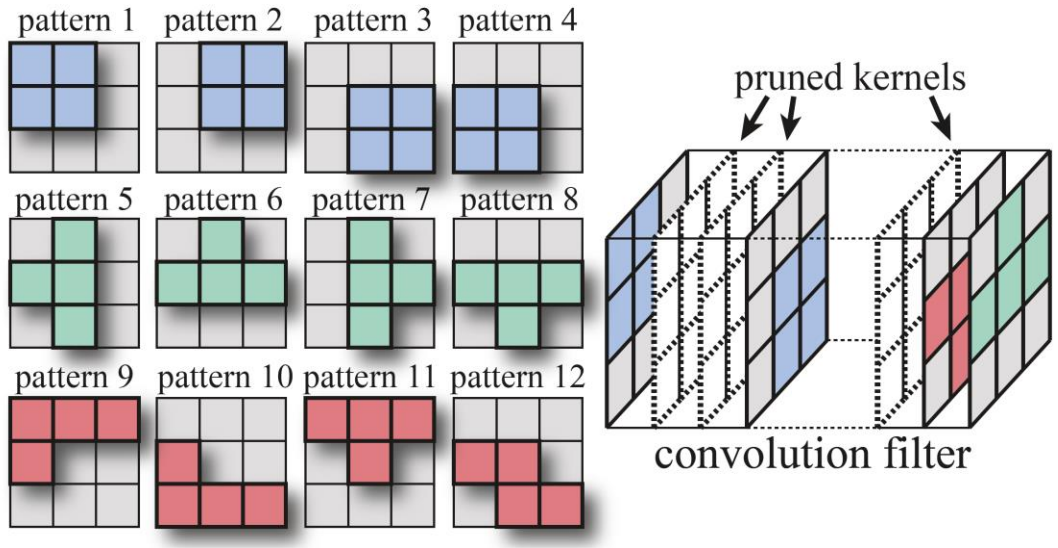


Network	Accuracy		
	Dense FP16	Sparse FP16	Sparse INT8
ResNet-34	73.7	73.9	73.7
ResNet-50	76.1	76.2	76.2
ResNet-50 (SWSL)	81.1	80.9	80.9
ResNet-101	77.7	78.0	77.9
ResNeXt-50-32x4	77.6	77.7	77.7
ResNeXt-101-32x16	79.7	79.9	79.9
ResNeXt-101-32x16 (WSL)	84.2	84.0	84.2
DenseNet-121	75.5	75.3	75.3
DenseNet-161	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 will bring about large speedup for GEMM workloads, and it will not incur performance drop for DNN models.

Pattern-Based Sparsity

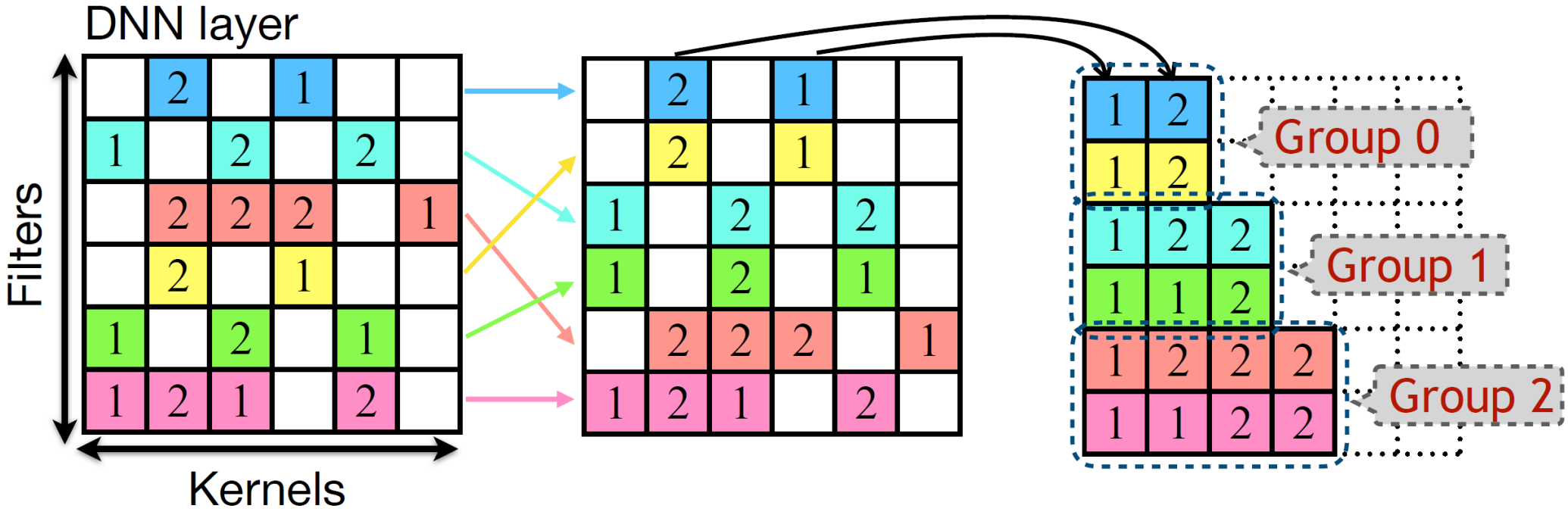
- For each output channel in the convolution filter
  - Either prune it away
  - Or select its sparsity pattern from a predefined set
- Problem
  - Load imbalance between different channel groups (different number of pruned kernels)



Zhang, C., Yuan, G., Niu, W., Tian, J., Jin, S., Zhuang, D., ... & Tao, D. (2021, June). Clicktrain: Efficient and accurate end-to-end deep learning training via fine-grained architecture-preserving pruning. In *Proceedings of the ACM International Conference on Supercomputing* (pp. 266-278).

Pattern-Based Sparsity Example: PatDNN

- Filter Kernel Reorder for pruned CNNs
  - Within each group (warp), every thread has the same amount of work

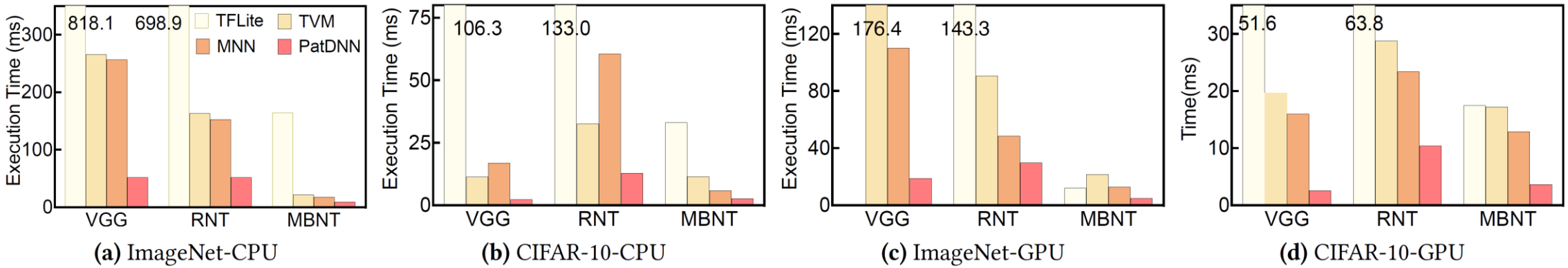


The number in the grid denote different kernel **patterns**

Niu, W., Ma, X., Lin, S., Wang, S., Qian, X., Lin, X., ... & Ren, B. (2020, March). Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 907-922).

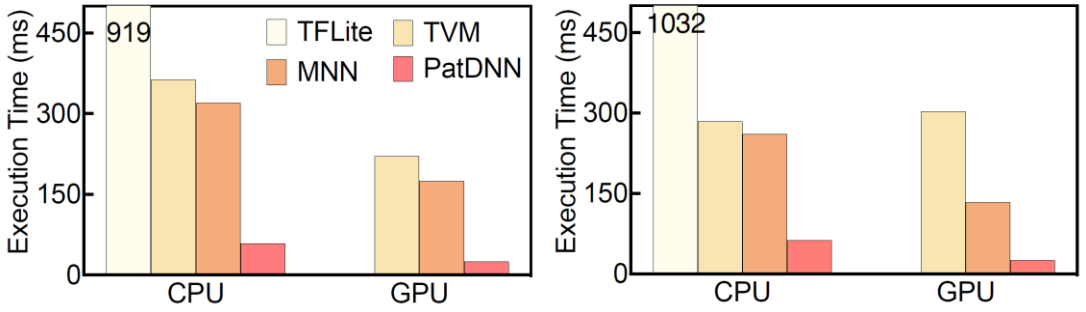
Pattern-Based Sparsity Example: PatDNN

- Overall performance on Samsung Galaxy S10



(a) ImageNet-CPU (b) CIFAR-10-CPU (c) ImageNet-GPU (d) CIFAR-10-GPU

- On other two platform



(a) Kirin 980 (b) Snapdragon 845