EE-508: Hardware Foundations for Machine Learning

Sparsity and Pruning

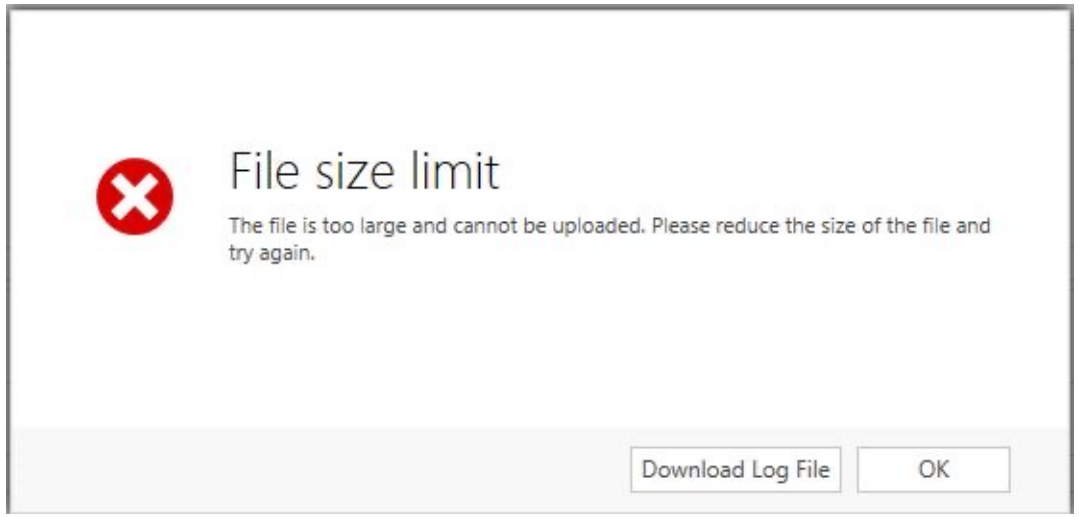
University of Southern California

Ming Hsieh Department of Electrical and Computer Engineering

Instructors:
Arash Saifhashemi

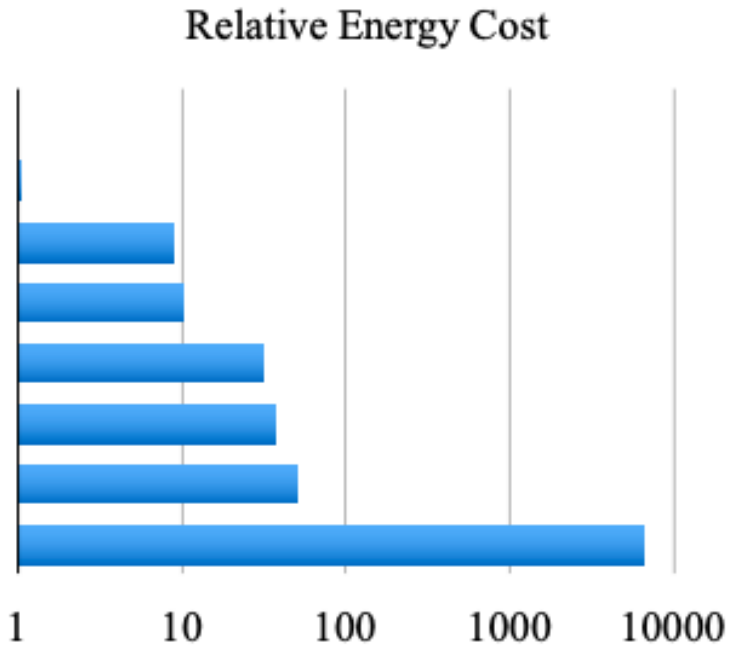
Sparsity in DNNs

- Data used in DNN computations often exhibits sparsity.
  - Many values are repeated, commonly zeros, leading to a high percentage of zero values.
- Benefits:
  - Reducing data footprint: lower storage and data movement.
  - Decreasing MAC (Multiply-Accumulate) operations.



Sparsity in DNNs

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

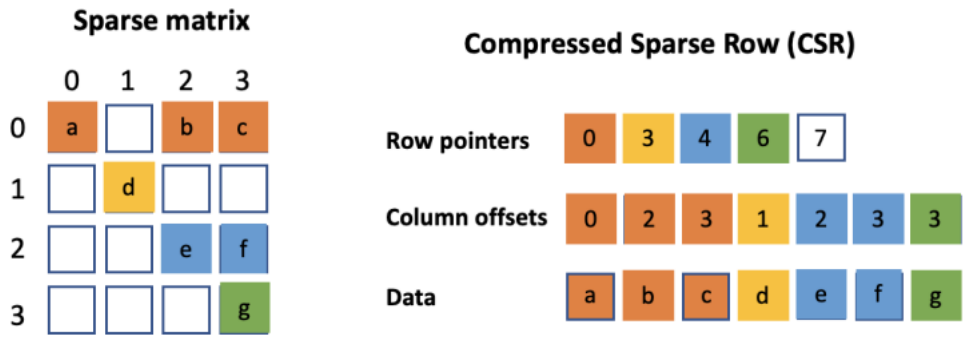


Energy table for 45nm CMOS process.
Memory access is 3 orders of magnitude more energy expensive than simple arithmetic

Source: Song Han et al. Learning both Weights and Connections for Efficient Neural Network

How Does Sparsity Help?

- Sparse data can be compressed.
  - Reduces storage.
  - Reduces movement of data.
- Beneficial if the compression/decompression algorithm is lightweight.



Source: Kevin Hunteret al. Two Sparsities Are Better Than One: Unlocking the Performance Benefits of Sparse-Sparse Networks

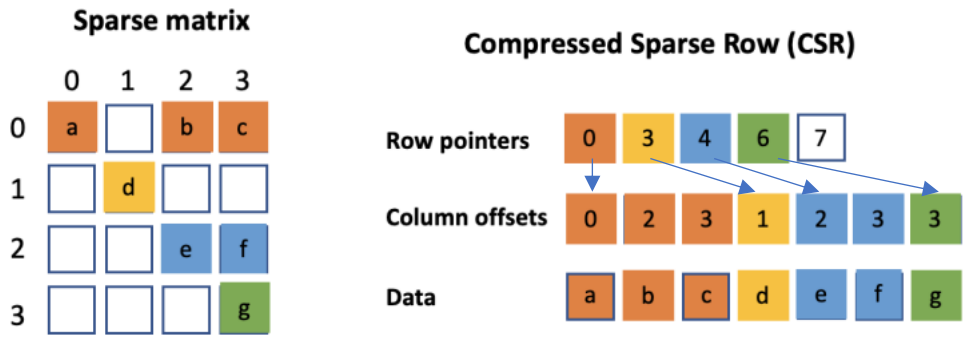
- **Compressed Sparse Row (CSR) Components**:
  - **Row Pointers**: Cumulative count of **non-zero** elements up to the end of each row.
  - **Column Offsets**: The column indices of each **non-zero** element in the matrix.
  - **Data**: Stores the values of the non-zero elements in the order they appear in the matrix from top to bottom and left to right.

$$Total\ CSR\ size = 2 \cdot NNZ + (n + 1)$$

NNZ: Number of non-zero values in a matrix of size nxn

How Does Sparsity Help?

- Sparse data can be compressed.
  - Reduces storage.
  - Reduces movement of data.
- Beneficial if the compression/decompression algorithm is lightweight.



Source: Kevin Hunteret al. Two Sparsities Are Better Than One: Unlocking the Performance Benefits of Sparse-Sparse Networks

- **Compressed Sparse Row (CSR) Components**:
  - **Row Pointers**: Cumulative count of non-zero elements up to the end of each row.
  - **Column Offsets**: The column indices of each non-zero element in the matrix.
  - **Data**: Stores the values of the non-zero elements in the order they appear in the matrix from top to bottom and left to right.

$$Total\ CSR\ size = 2 \cdot NNZ + (n + 1)$$

NNZ: Number of non-zero values in a matrix of size nxn

CSR Access Algorithm

- To retrieve A[i,j]:
  - Find the **row range** using row_pointer:

$$\text{start} = \text{row\_pointer}[i], \quad \text{end} = \text{row\_pointer}[i+1]$$

This identifies the **range of indices** in values[] corresponding to row i.

  - **Search** in `column_indices[start:end]` for column j (python notation):
    - If **found**, return values[k], where k is the matching index in column_indices[].
    - If **not found**, return **0** (since it's a sparse matrix).

Example Sparse Matrix

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 7 & 0 \\ 0 & 8 & 0 & 0 & 6 \end{bmatrix}$$

CSR Representation:

- `values[] = [3, 4, 5, 7, 8, 6]`
- `column_indices[] = [2, 4, 0, 3, 1, 4]`
- `row_pointer[] = [0, 2, 2, 4, 6]`

Example: Find A[2,3]

From row_pointer[]:

1. **start** = row_pointer[2] = 2
2. **end** = row_pointer[3] = 4
3. Look in column_indices[2:4] = [0, 3] (corresponding to values[2:4] = [5, 7]).

1.Find j=3 in column_indices[]:

1. column_indices[3] = 3 → **Match found**.
2. Corresponding value: values[3] = 7.

Result: A[2,3]=7

CSR Access Algorithm

- To retrieve A[i,j]:
  - Find the **row range** using row_pointer:

$$\text{start} = \text{row\_pointer}[i], \quad \text{end} = \text{row\_pointer}[i+1]$$

This identifies the **range of indices** in values[] corresponding to row i.

  - **Search** in `column_indices[start:end]` for column j (python notation):
    - If **found**, return values[k], where k is the matching index in column_indices[].
    - If **not found**, return **0** (since it's a sparse matrix).

Example Sparse Matrix

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 7 & 0 \\ 0 & 8 & 0 & 0 & 6 \end{bmatrix}$$

CSR Representation:

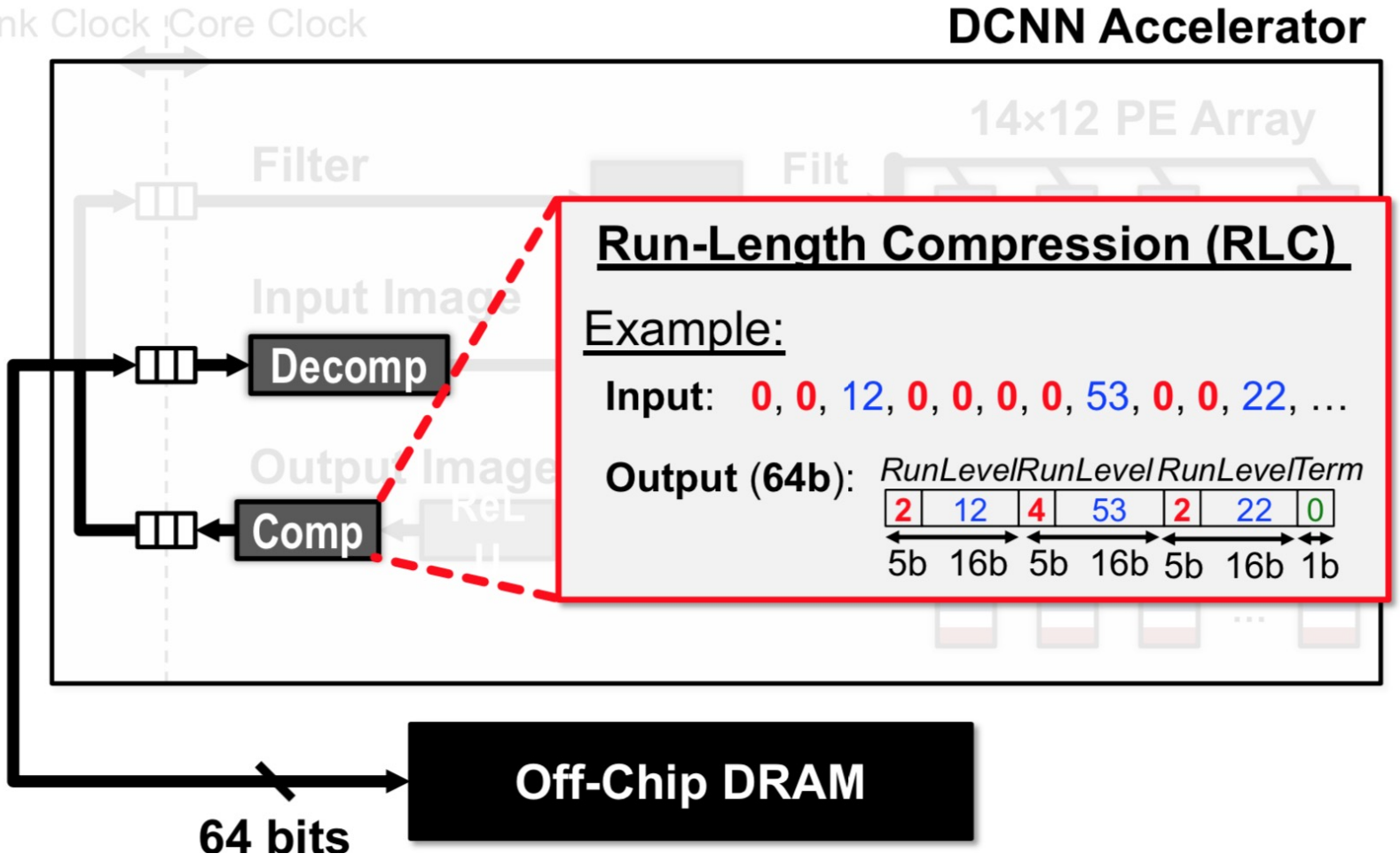
- `values[] = [3, 4, 5, 7, 8, 6]`
- `column_indices[] = [2, 4, 0, 3, 1, 4]`
- `row_pointer[] = [0, 2, 2, 4, 6]`

Example: Find A[1,2]

From row_pointer[]:

1. **start** = row_pointer[1] = 2
2. **end** = row_pointer[2] = 2
3. No elements exist in row 1.
4. No match found, return 0.

I/O Compression in Eyriss



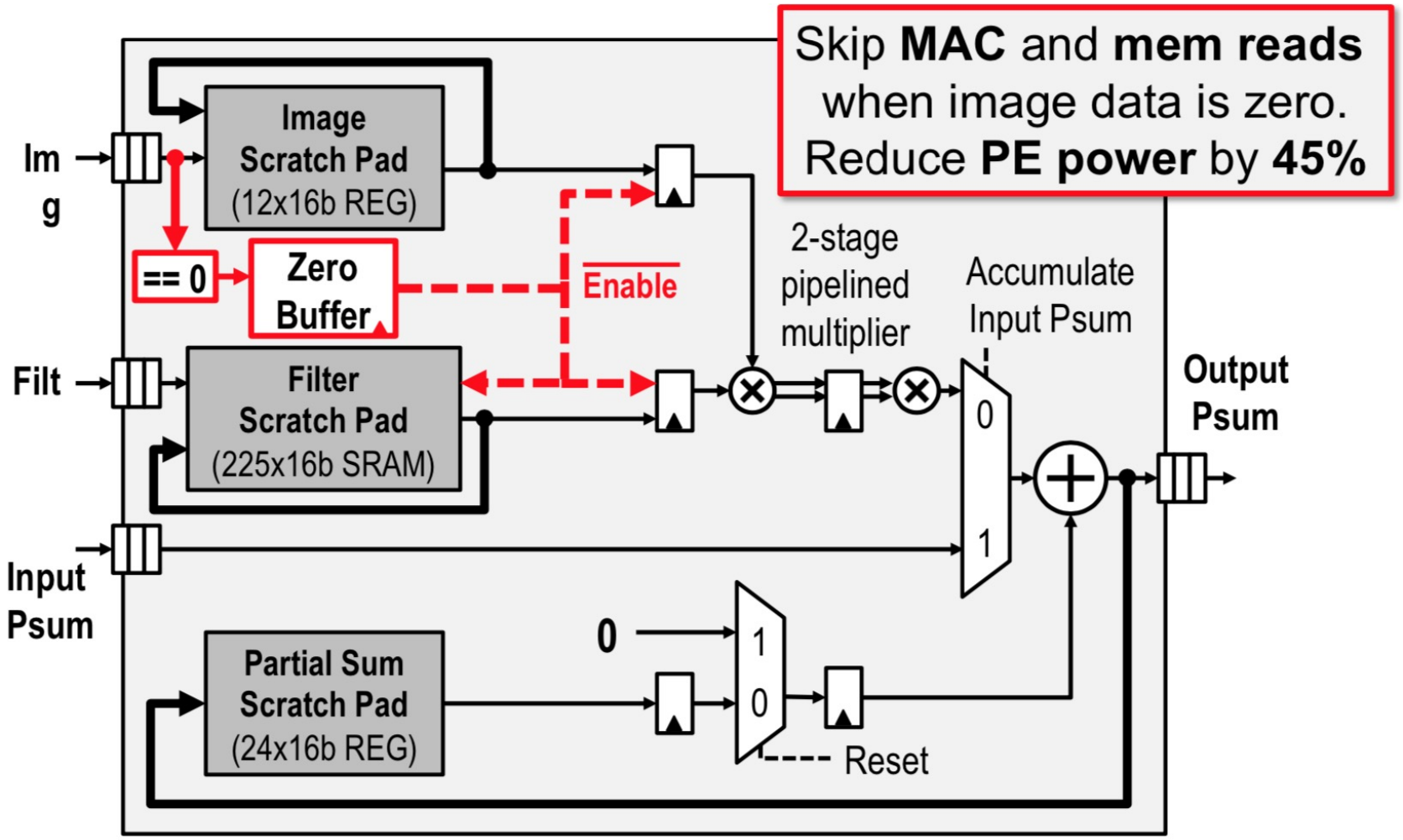
Consecutive zeros with a maximum run length of 31 are represented using a 5-b number as the Run. The next value is inserted directly as a 16-b Level, and the count for run starts again. Every three pairs of run and level are packed into a 64-b word, with the last bit indicating if the word is the last one in the code.

Source: Chen et al., ISSCC 2016

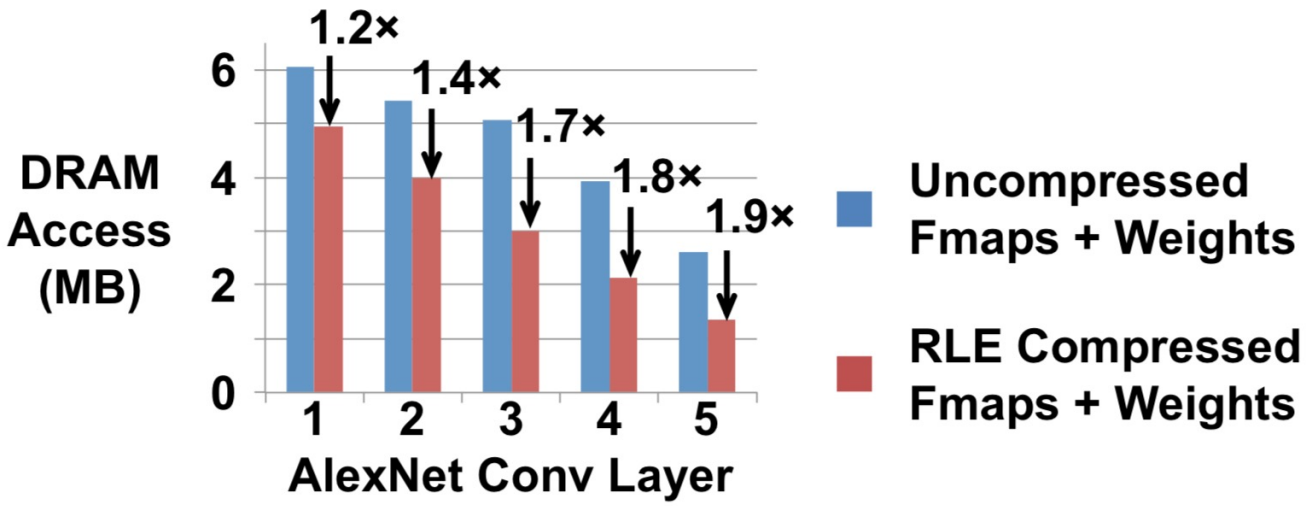
RLC Encoding

- Format: (Run-Length, Value) ...., (0, end)
- Example 1:
  - Input data: **0, 0**, 12**, 0, 0, 0, 0**, 53, **0, 0,** 22, …
  - Output: (2, 12), (4, 53), (2, 22), (0, end)
- Example 2:
  - Input data: 0, 0, **12, 5**, 0, 0, 53, 8, 0, 22, …
  - Output: (2, 12), **(0, 5)**, (2, 53), (0, 8), (1, 22), (0, Term)
  - **If two nonzero values are consecutive**, the second value is **stored with a run-length of 0**.
- **Runtime Complexity:** O(N), where is is the input size

Data Gating/Zero Skipping In Eyeriss



Compression Reduces Dram Bandwidth



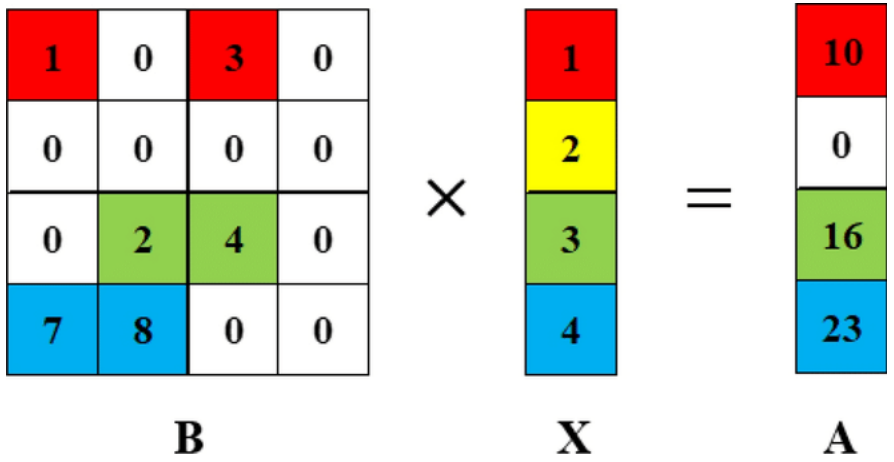
Simple RLC within 5% - 10% of theoretical entropy limit

Source: Chen et al., ISSCC 2016

- **Shannon entropy** of the data, which represents the **lowest possible average number of bits per symbol** needed to represent the data **optimally**.
- Since it's **within 5%-10%**, it means **RLE is already performing near-optimal compression**, making it an **efficient choice for AlexNet**.

How Does Sparsity Help?

- Sparse data potentially leads to less computation
  - Reduces storage.
  - Reduces movement of data.
- Requires detection of extra calculations



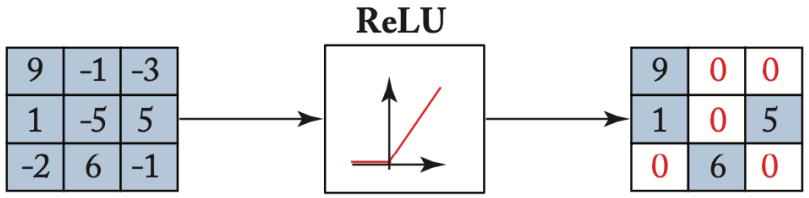
Source: Haodong Bian et al. A simple and efficient storage format for SIMD-accelerated SpMV

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 4 \\ 6 & 0 & 0 \end{bmatrix}$$

$$C = A + B = \begin{bmatrix} 1+0 & 0+2 & 0+0 \\ 0+0 & 3+0 & 0+4 \\ 0+6 & 0+0 & 5+0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 4 \\ 6 & 0 & 5 \end{bmatrix}$$

Sources of Sparsity

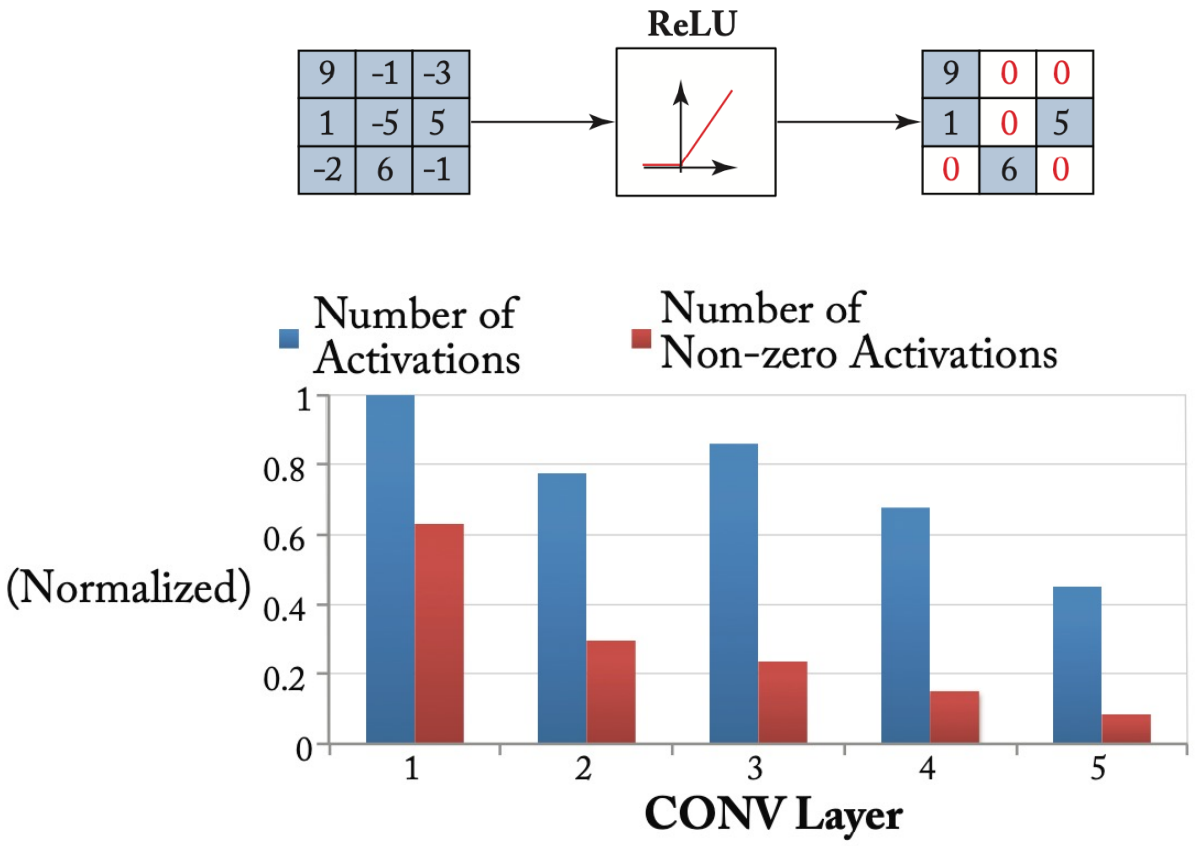
- Activations
  - Data dependent and not known in advance.
    - Optimization is generally performed during inference.
  - Sources:
    - ReLU (Another reason why ReLU is popular)
    - Correlation in data
- Weights
  - Can be known in advance
    - Optimization can be performed offline.



(a) ReLU nonlinearity

Source: Sze and Emer. Efficient Processing of Deep Neural Networks, 2020

Pruning Activations

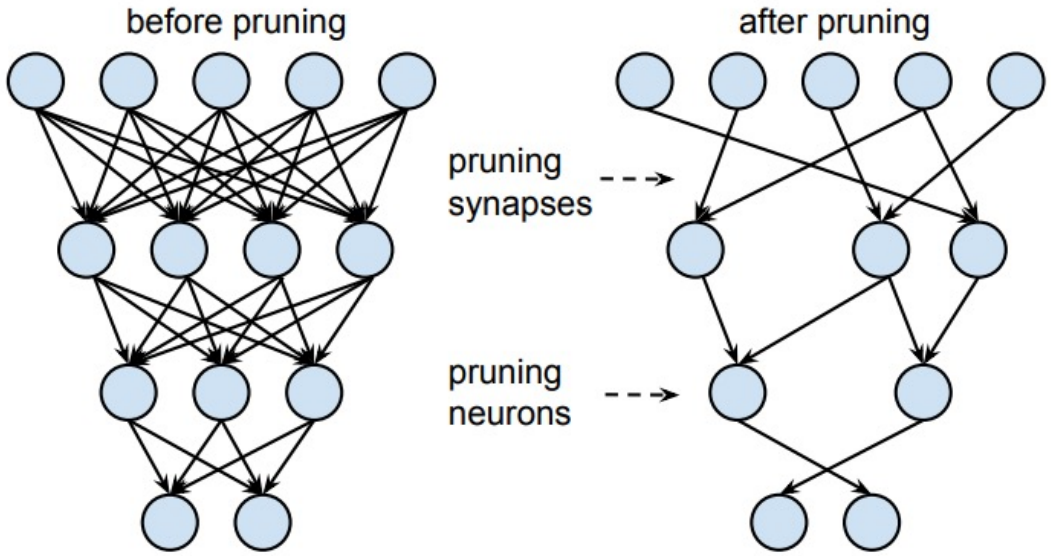


(b) Distribution of activation after ReLU of AlexNet

Source: Sze and Emer. Efficient Processing of Deep Neural Networks, 2020

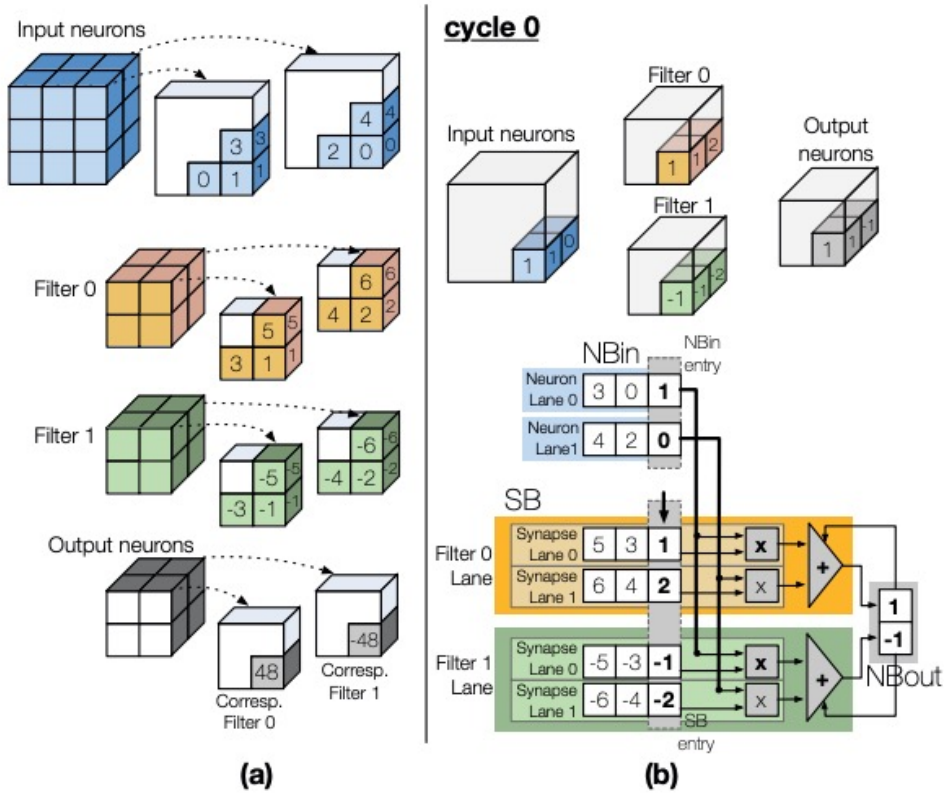
Sparsity in DNNs

- Types of sparsity:
  - Inherent in the data
  - Created using a process called Pruning



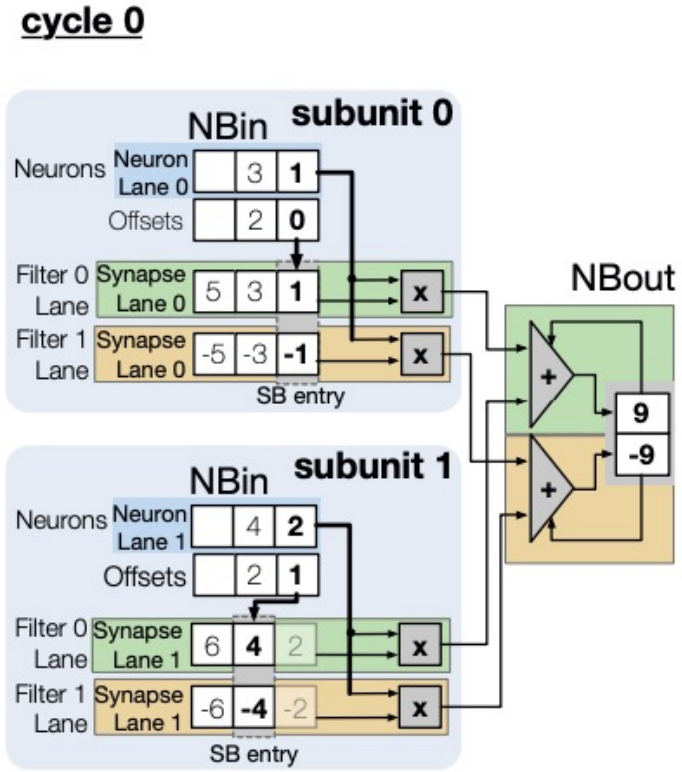
Source: Song Han et al. Learning both Weights and Connections for Efficient Neural Network

Sparsity in Activations



DaDianNao

Source: Albericio, ISCA 2016



DaDianNao enhanced by CNV

NBin (Neuron Buffer Input)

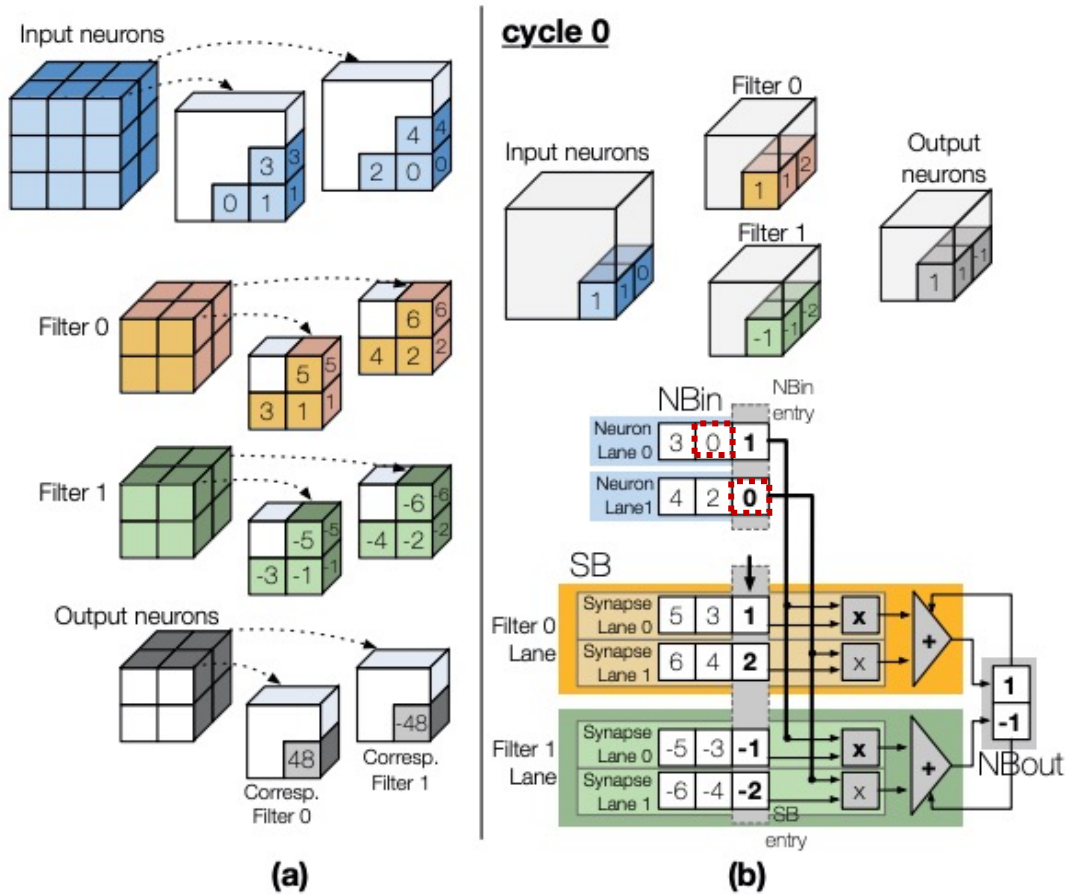
•Stores the input activations that will be processed in this cycle.

SIB (Synapse Input Buffer)

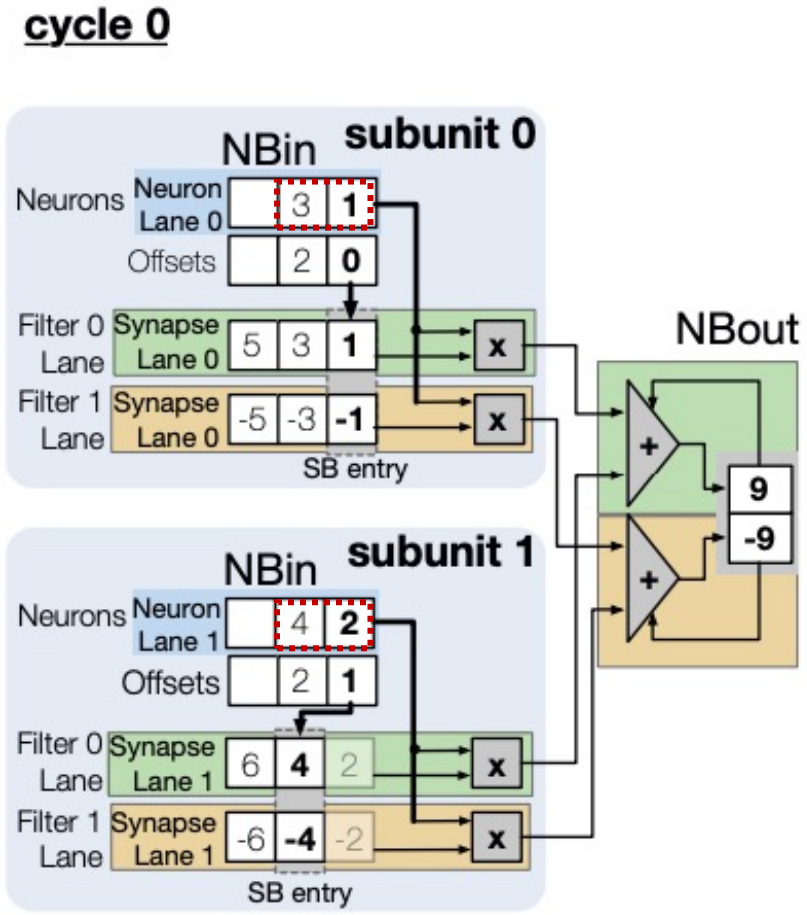
•Stores filter weights (Filter 0 and Filter 1).

•Data is arranged into **lanes** for parallel execution.

Sparsity in Activations



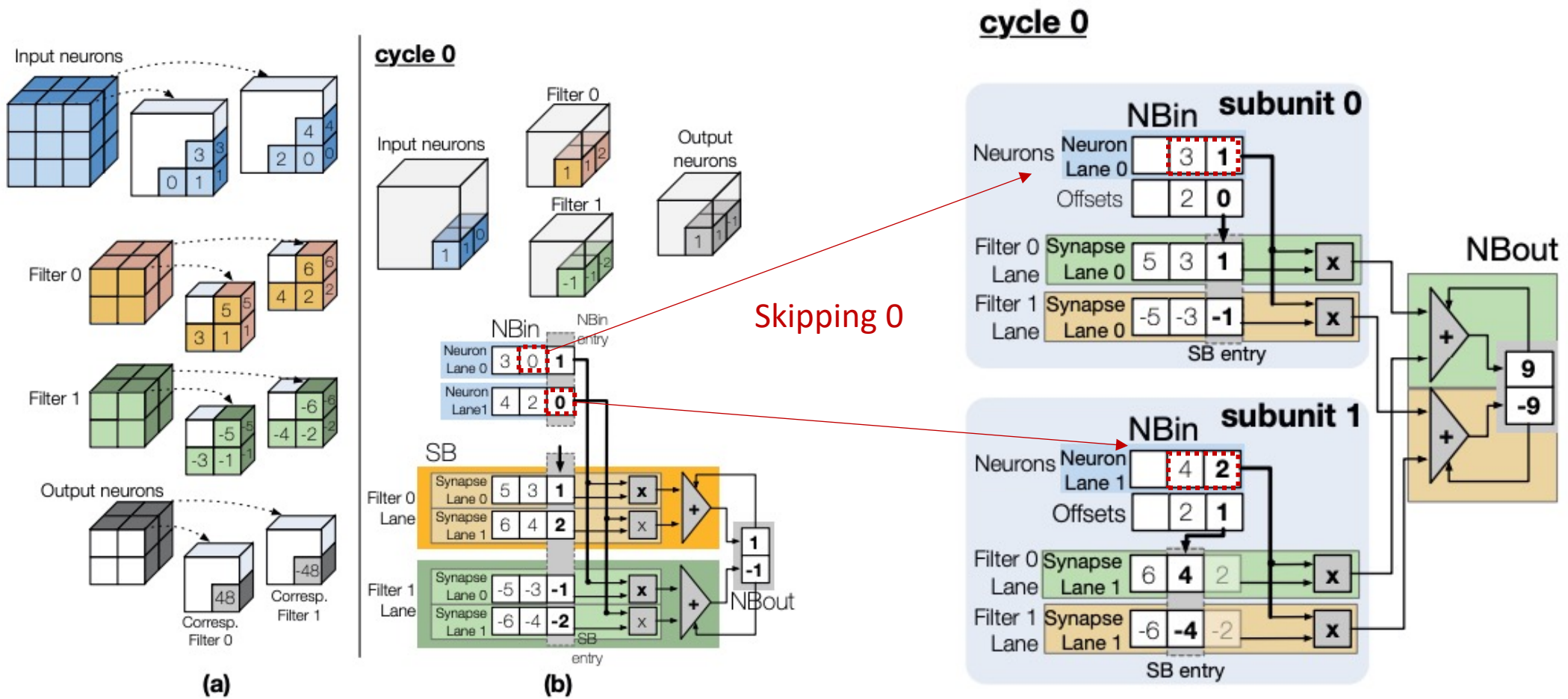
DaDianNao



DaDianNao enhanced by CNV

Source: Albericio, ISCA 2016

Cnvlutin: Skipping Zeros



DaDianNao

DaDianNao enhanced by CNV

4.49% overhead, 1.37x speed up on average. 1.52x with further pruning

Source: Albericio, ISCA 2016

Cnvlutin: Skipping Zeros

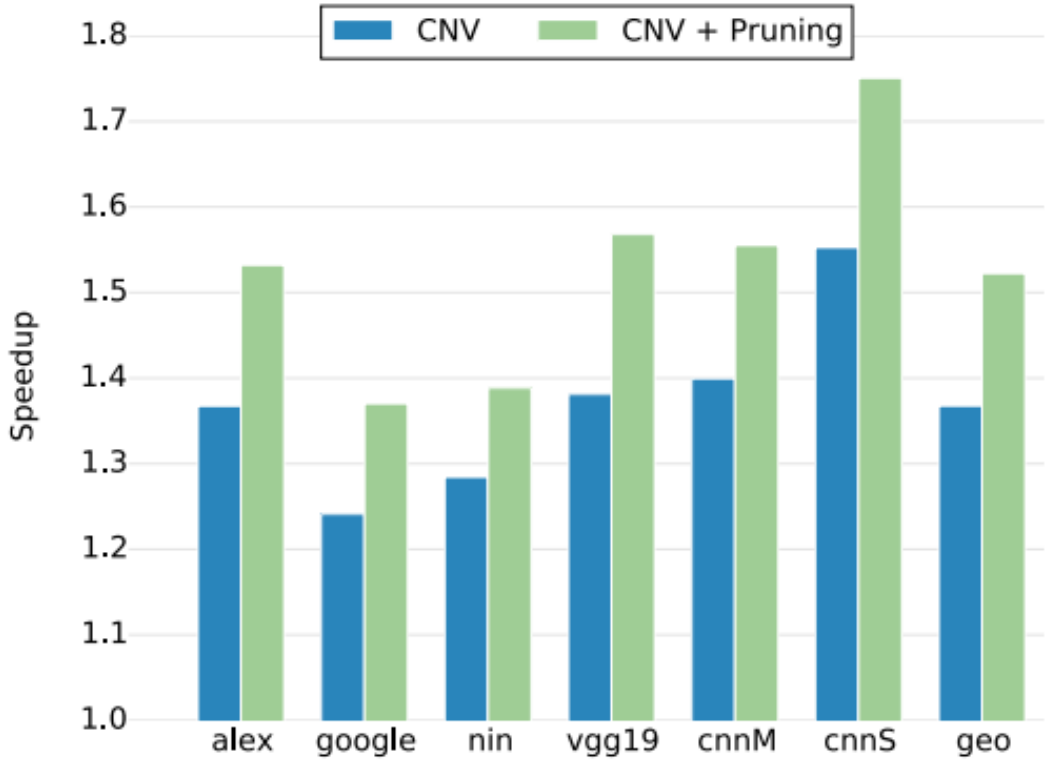


Fig. 9: Speedup of *CNV* over the baseline.

The first bar (CNV) shows the speedup when only zero neurons are considered, while the second bar (CNV + Pruning) shows the speedup when additional neurons are also skipped without affecting the network overall accuracy

Source: Albericio, ISCA 2016

Cnvlutin: Skipping Zeros

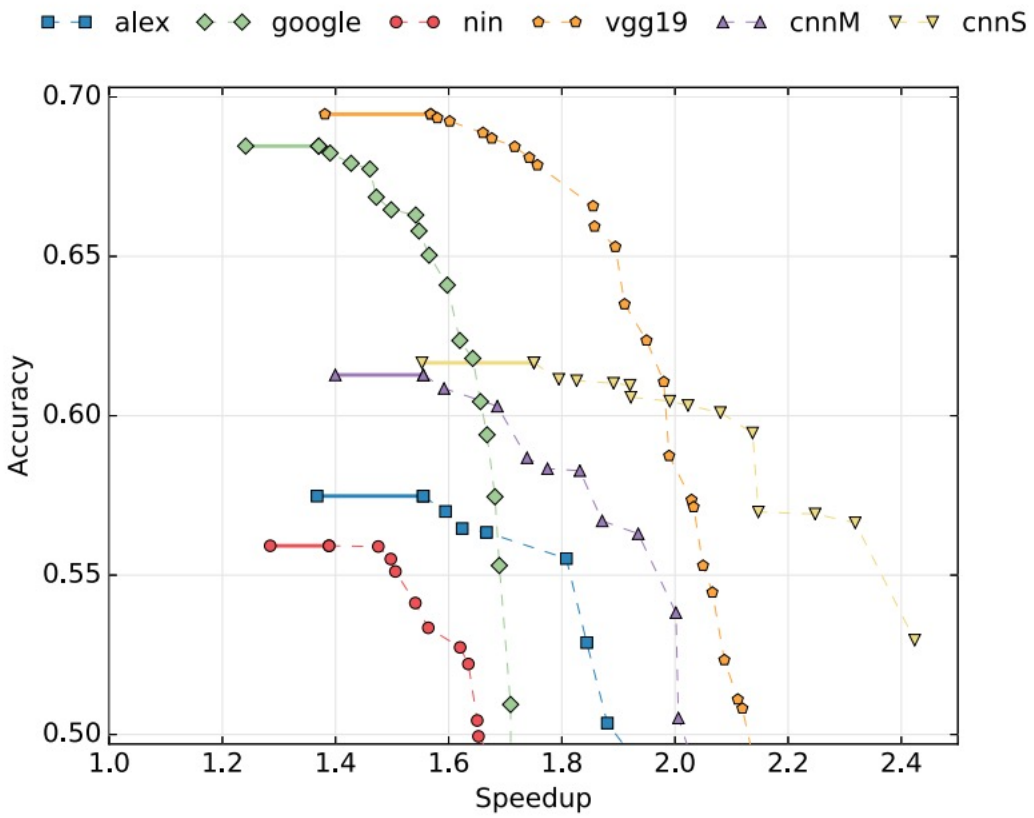


Fig. 14: Accuracy vs. Speedup trade-off from pruning neurons

The leftmost point for each network corresponds to CNV in Figure 9 where only zero-valued neurons were removed. Generally, all networks exhibit an initial region where neurons can be pruned without affecting accuracy. This region is shown with a solid line.

Source: Albericio, ISCA 2016

[UCNN, ISCA 2018]: Exploiting Redundant Weights

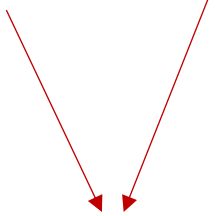
$$[w, x, y, z] \cdot [a, b, a, c] = aw + bx + ay + cz$$

4 Multiplications
4 weight reads
3 Additions

Change order

$$[w, y, x, z] \cdot [a, a, b, c] = a(w + y) + bx + cz$$

$[0,2,1,3]$ $[0,2,1,3]$



3 Multiplications
3 weight reads
3 Additions

Overhead: Indirection tables hold the shuffled indices

[UCNN, ISCA 2018]: Exploiting Redundant Weights

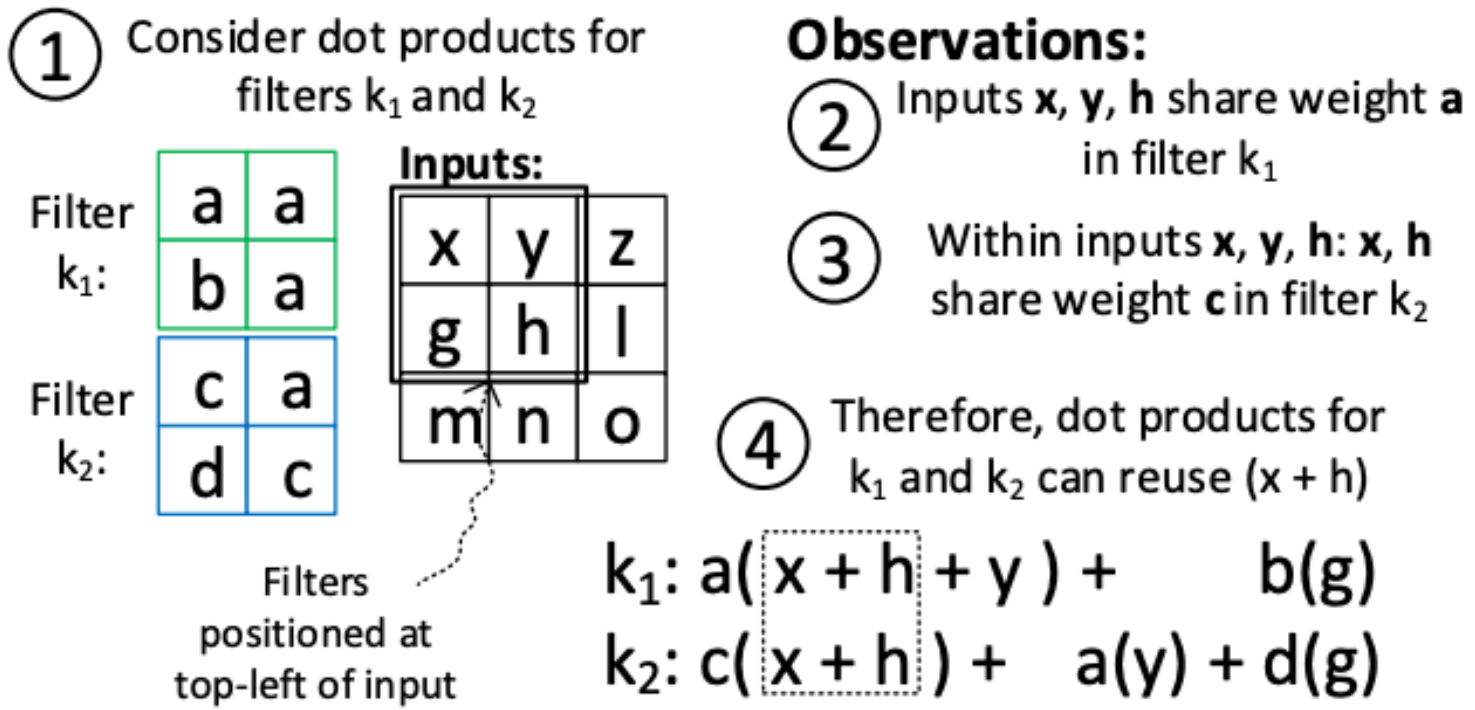
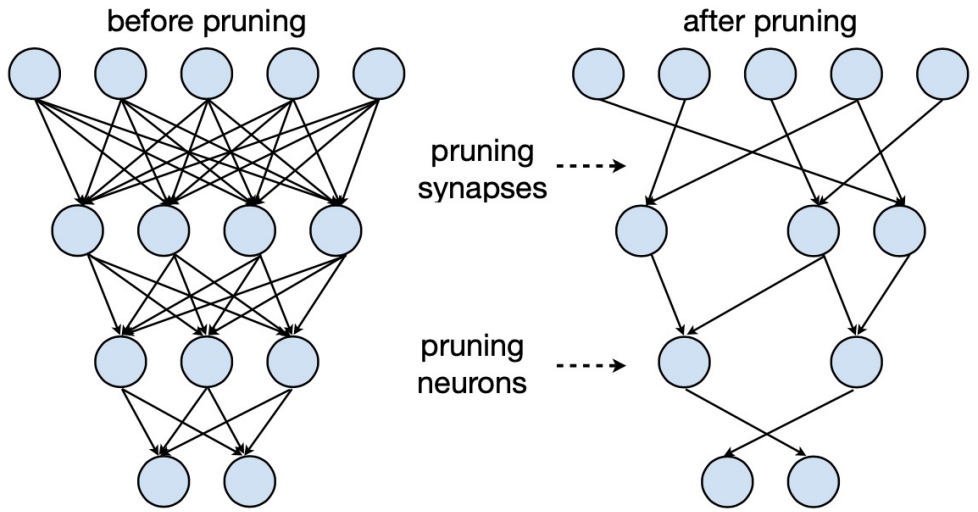
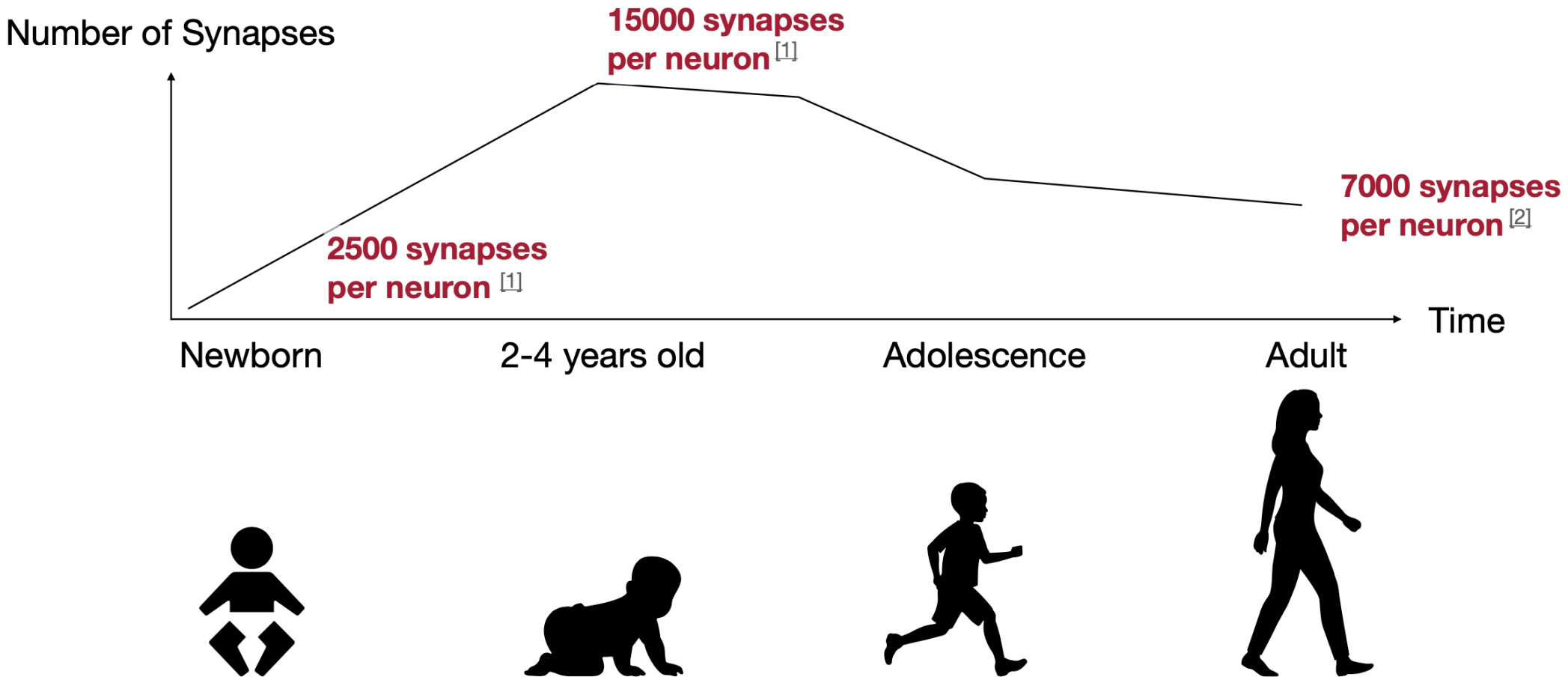


Fig. 4. Activation group reuse example ($G = 2$).

Pruning


before pruning
after pruning
pruning
synapses
pruning
neurons

Pruning Happens in Human Brain



Do We Have Brain to Spare? [Drachman DA, Neurology 2004]
Peter Huttenlocher (1931–2013) [Walsh, C. A., Nature 2013]

Data Source: 1, 2
Slide Inspiration: Alila Medical Media

Network Pruning

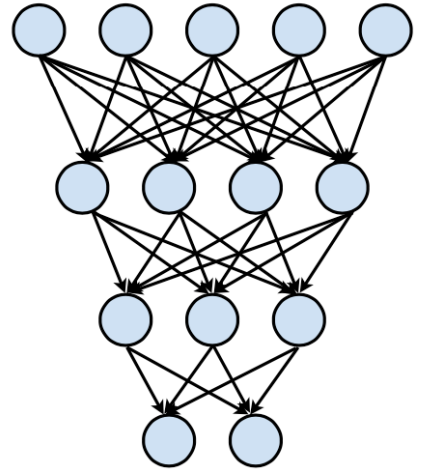
- Make neural network smaller by removing synapses and neurons
- DNN models are often designed with more parameters (weights) than necessary
  - Many weights in a DNN are redundant and can be pruned without significantly impacting accuracy.
  - Sparse DNN models, can achieve higher accuracy compared to dense models, given the same number of non-zero (effective) weights.
    - Why?
      - Noise Reduction: Eliminating small weights reduces noise, allowing the model to learn more effectively from the signal (Acts like regularization)
      - Better generalization and avoid overfitting
    - Whether sparse models outperform dense models depends on how the sparsity is introduced and how effectively the important connections are retained.

Source: D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, What is the state of neural network pruning? in Conference on Machine Learning and Systems (MLSys), 2020.

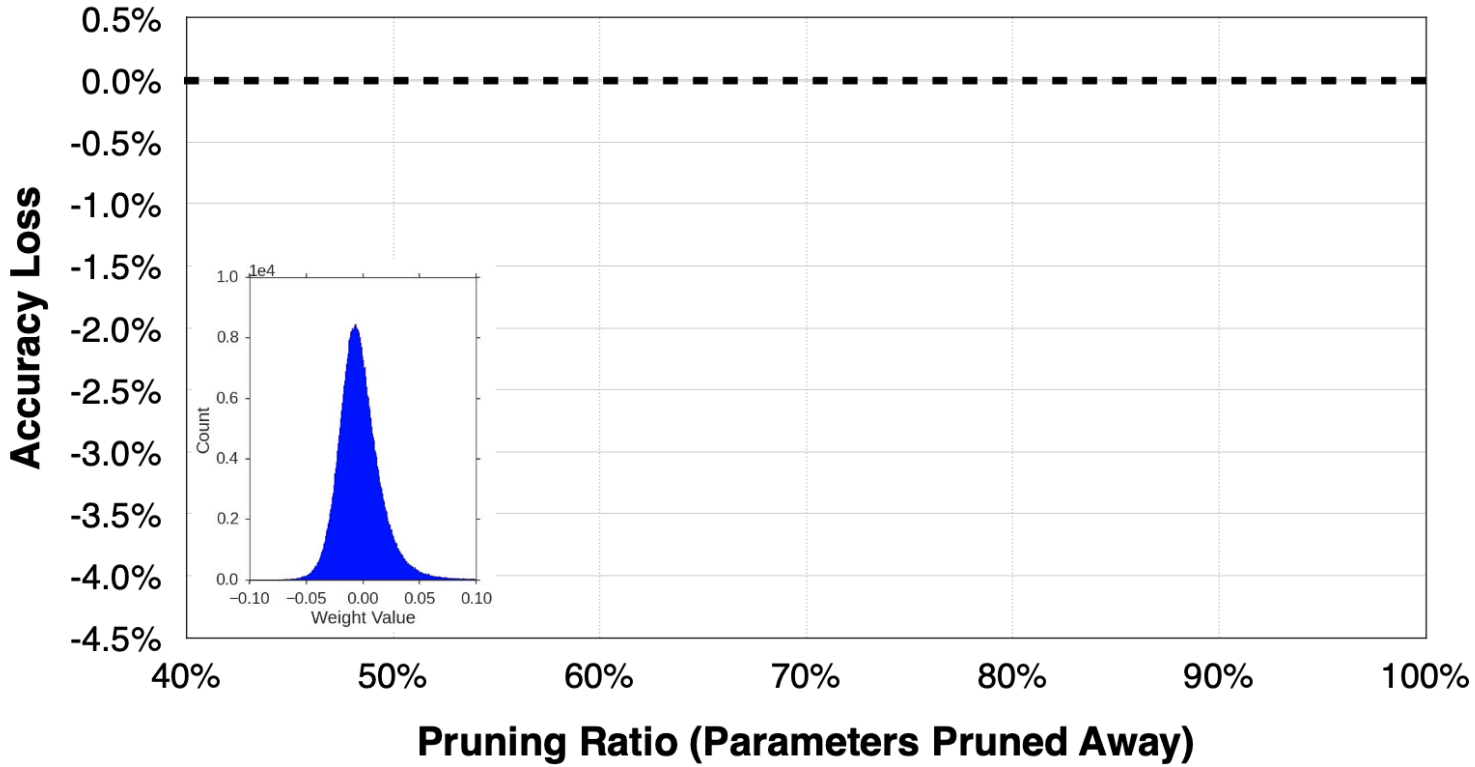
Network Pruning

- Network pruning typically involves two main stages applied iteratively:
  - **Weight/Neuron removal**: Identifying and removing (setting to zero) specific weights in the pre-trained dense DNN model.
  - **Fine-tuning**: Adjusting the remaining non-zero weights to recover or improve model performance post-pruning.

Neural Network Pruning



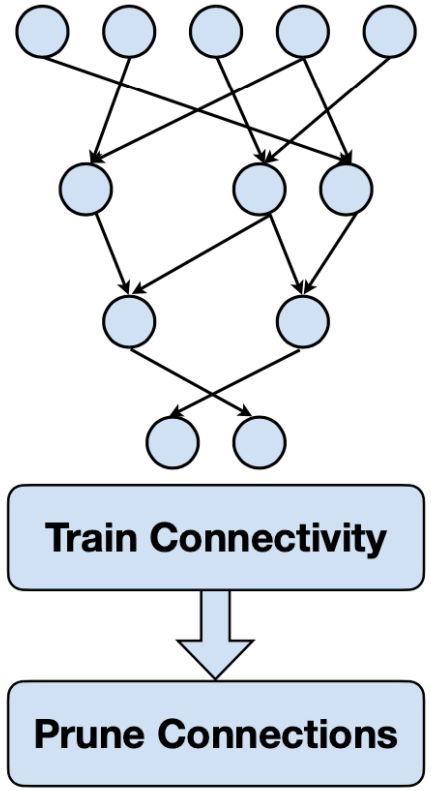
Train Connectivity

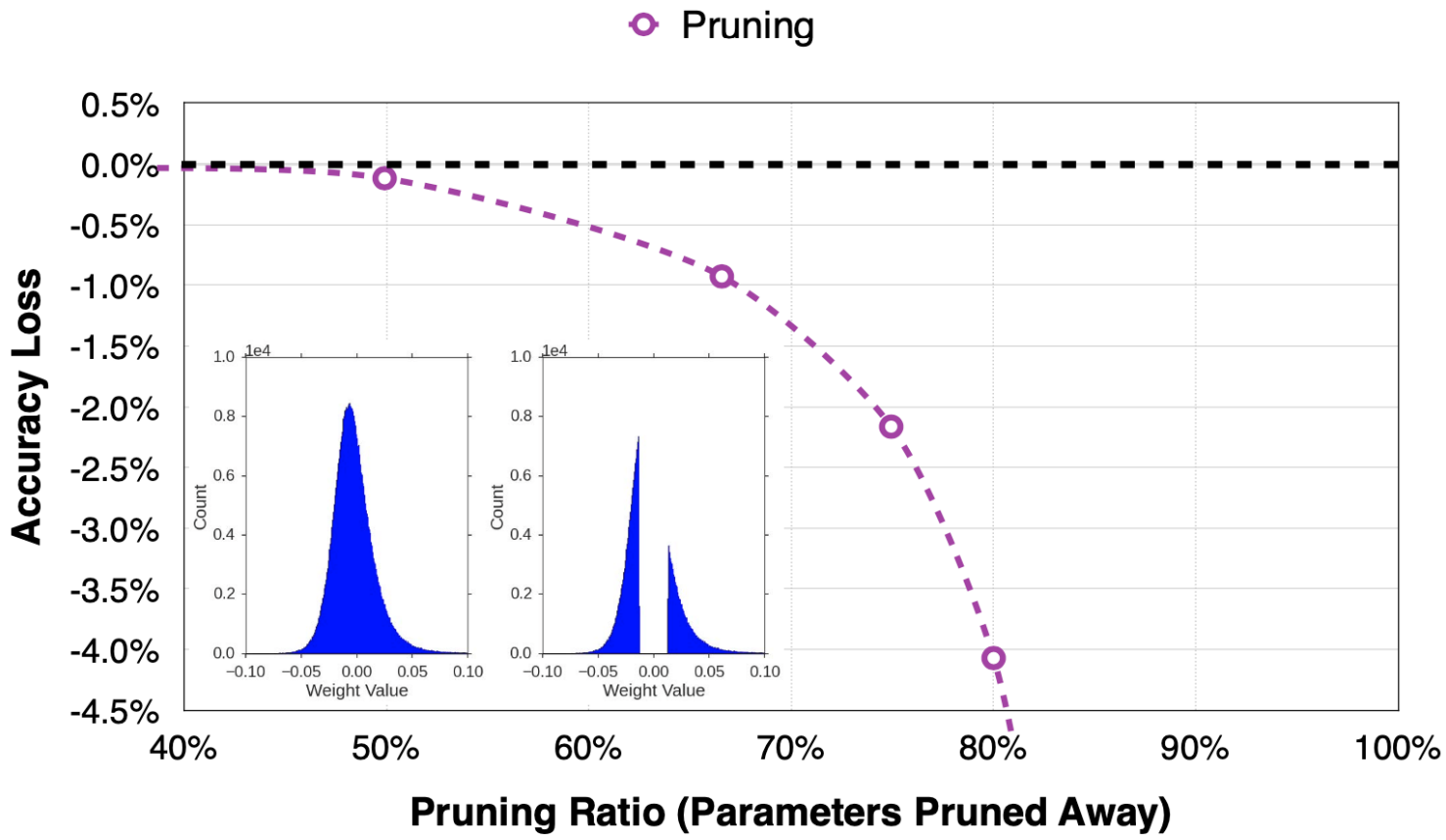


Learning Both Weights and Connections for Efficient Neural Network [Han *et al*., NeurIPS 2015]

Slide Credit: Song Han

Neural Network Pruning

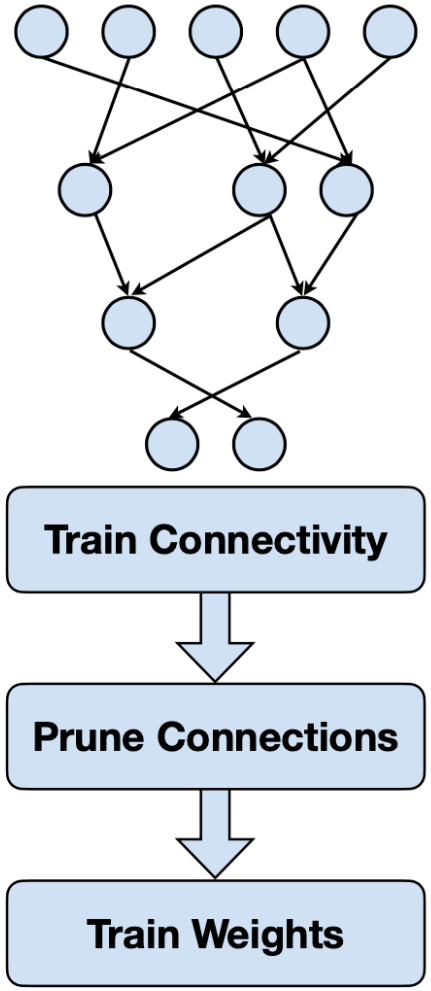


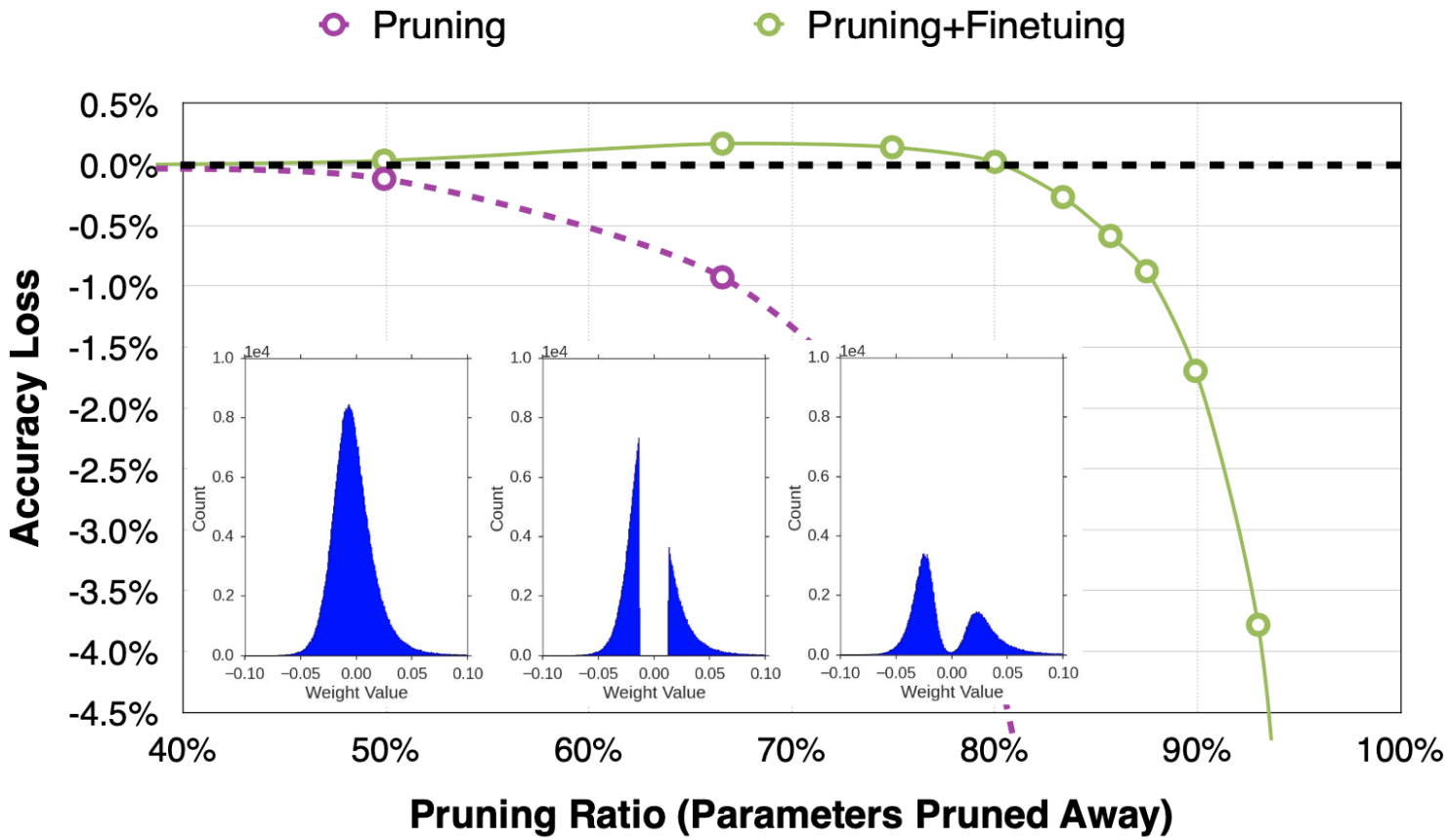


Learning Both Weights and Connections for Efficient Neural Network [Han *et al*., NeurIPS 2015]

Slide Credit: Song Han

Neural Network Pruning

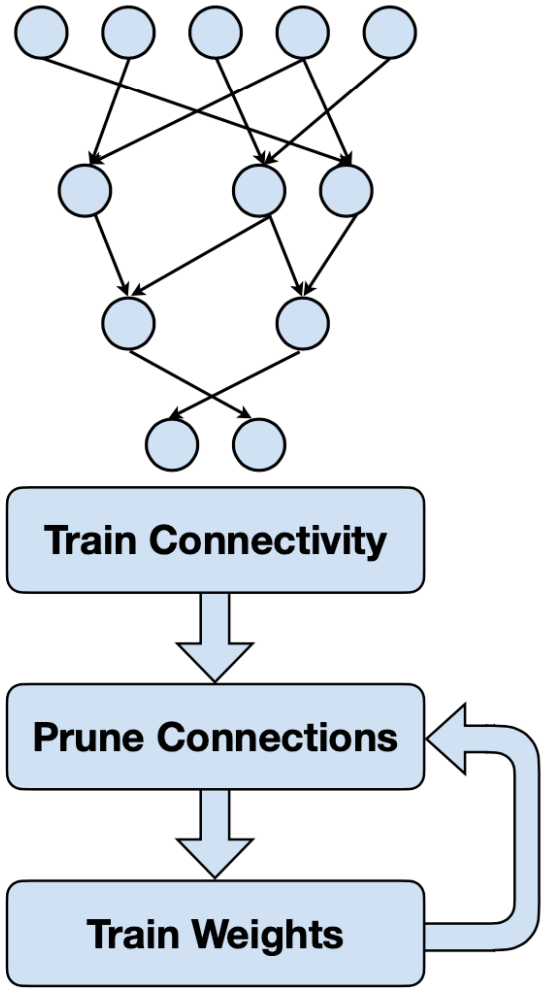


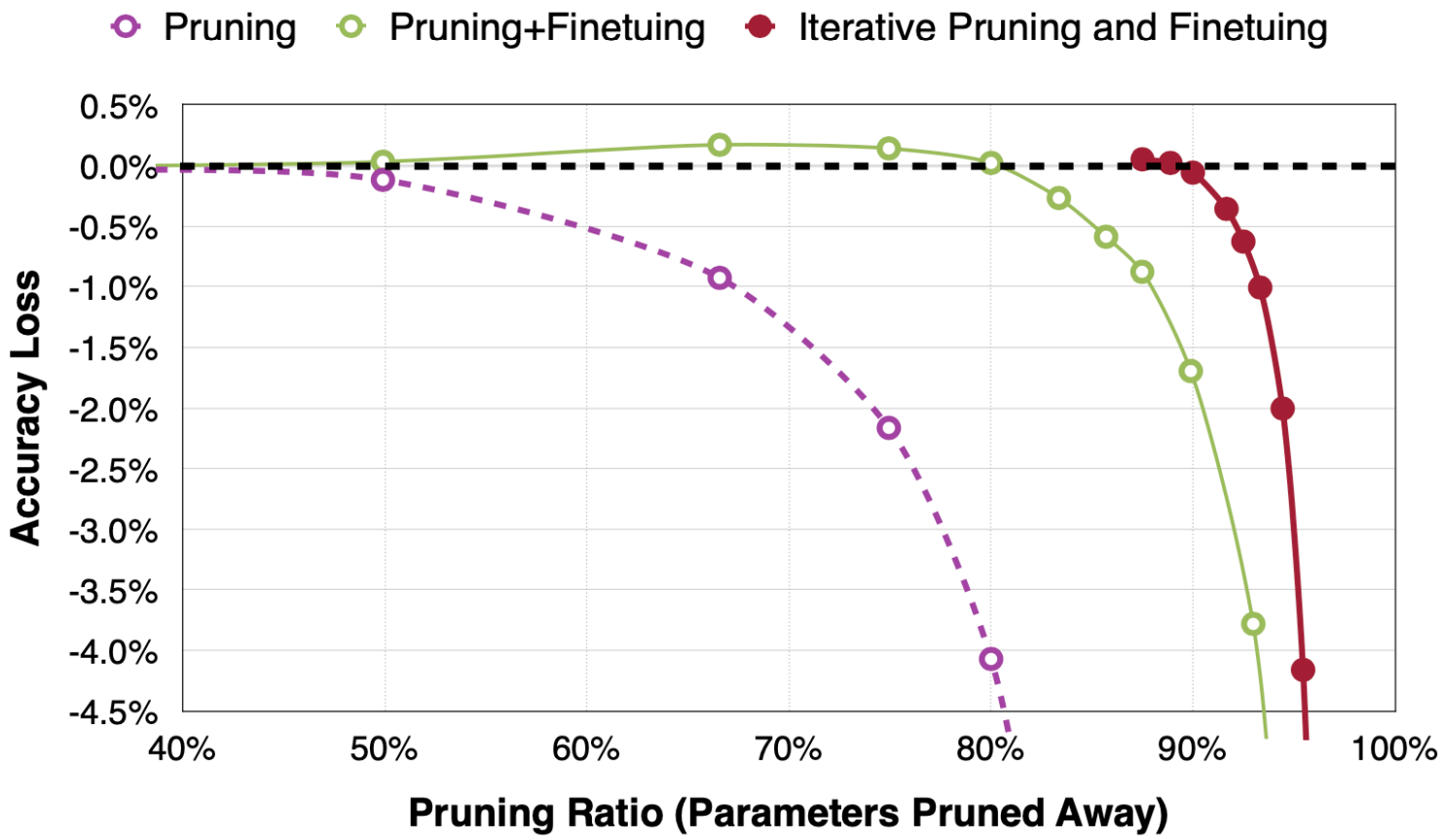


Learning Both Weights and Connections for Efficient Neural Network [Han *et al*., NeurIPS 2015]

Slide Credit: Song Han

Neural Network Pruning





Learning Both Weights and Connections for Efficient Neural Network [Han *et al*., NeurIPS 2015]

Slide Credit: Song Han

Network Pruning

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 ×

Efficient Methods and Hardware for Deep Learning [Han S., Stanford University]

Slide Credit: Song Han

Network Pruning

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.

Efficient Methods and Hardware for Deep Learning [Han S., Stanford University]

Slide Credit: Song Han

Pruning Formulation

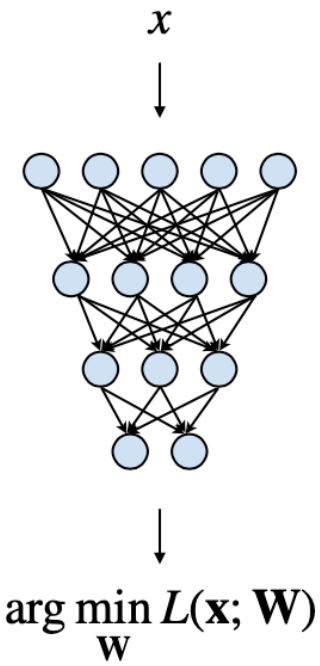
- In general, we could formulate the pruning as follows:

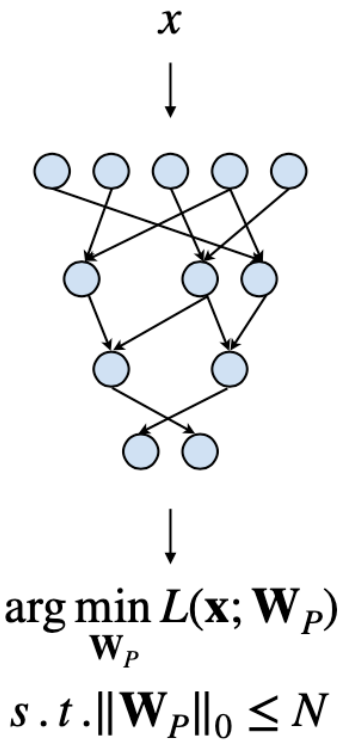
$$\arg\min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P)$$

subject to

$$\|\mathbf{W}_p\|_0 < N$$

  - L represents the objective function for neural network training;
  - $\mathbf{x}$ is input, $\mathbf{W}$ is original weights, $\mathbf{W}_P$ is pruned weights;
  - $\|\mathbf{W}_p\|_0$ calculates the #nonzeros in W_P, and N is the target #nonzeros.





Slide Credit: Song Han

Pruning Formulation

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

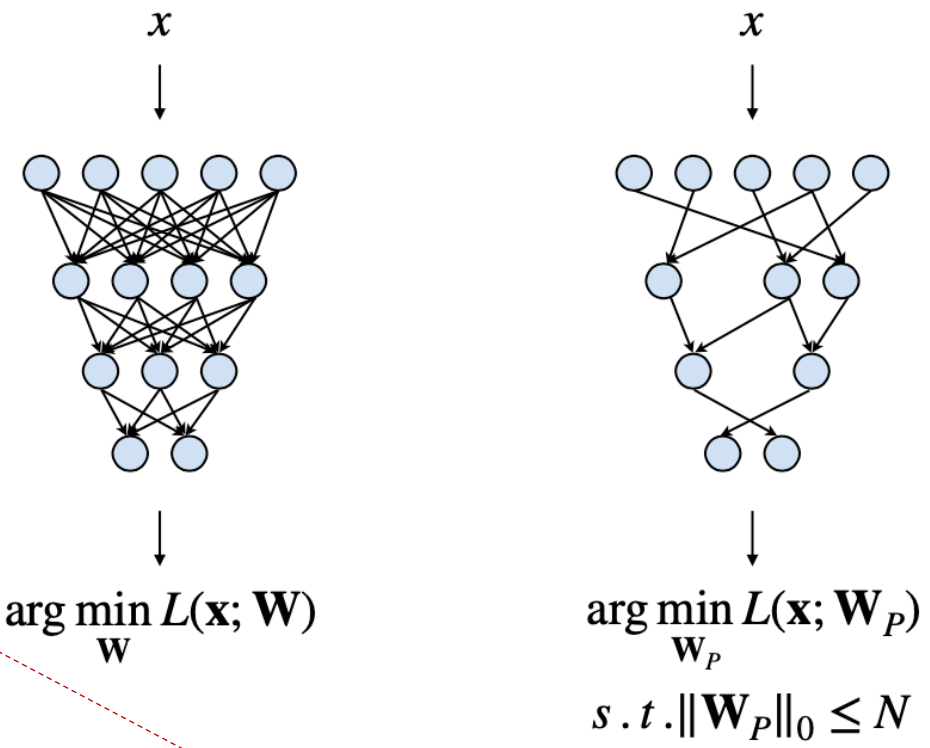
$$\arg\min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P)$$

subject to

$$\|\mathbf{W}_p\|_0 < N$$

- L represents the objective function for neural network training;
- $\mathbf{x}$ is input, $\mathbf{W}$ is original weights, $\mathbf{W}_P$ is pruned weights;
- $\|\mathbf{W}_p\|_0$ calculates the #nonzeros in W_P, and N is the target #nonzeros.

Find the set of pruned weights (Wp) that will minimize the loss function (L) of the network given the inputs (x) and the target #nonzero.

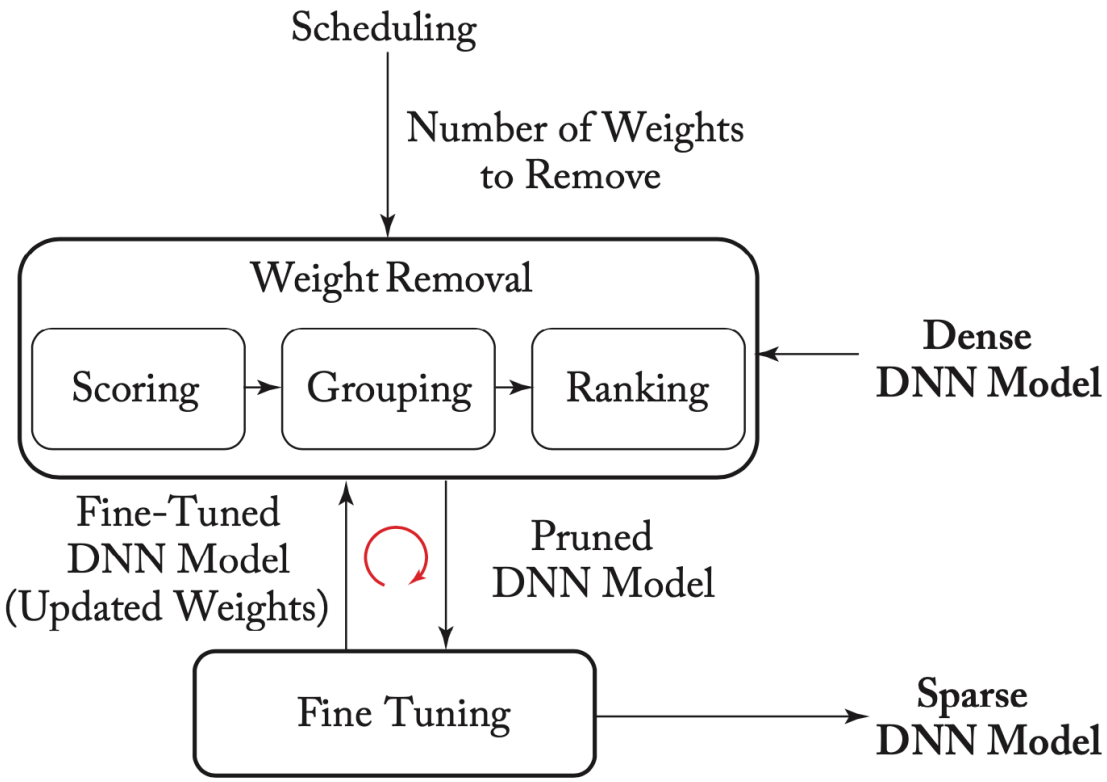


The number of non-zero weights in the pruned weight set (Wp) must be less than some number N. The $||\cdot||_0$ denotes the L0 norm, which counts the number of non-zero elements in Wp.

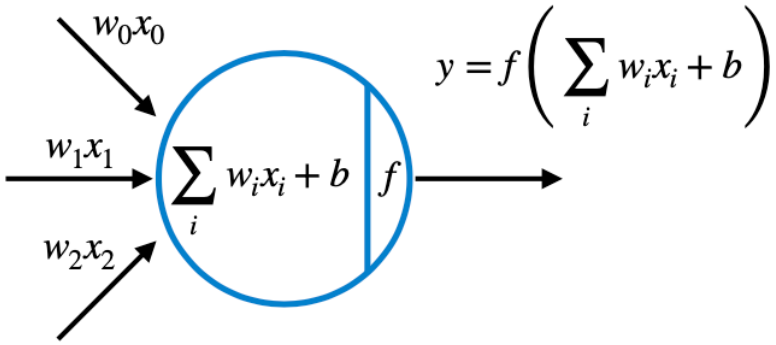
Slide Credit: Song Han

Network Pruning

- Scoring:
  - Assigns a score to each weight or a group of weights based on a given criterion.
  - The most common criterion is the impact of the weight(s) on accuracy.
- Grouping:
  - Weights can be grouped based on a pre-defined structure to allow groups of weights to be removed rather than a single weight.
- Ranking:
  - The weights are ranked based on their scores.
  - Removed weights based on rank.



Source: Sze and Emer. Efficient Processing of Deep Neural Networks, 2020



Example

$$f(\cdot) = \text{ReLU}(\cdot), \quad W = [10, -8, 0.1]$$

➡ $y = \text{ReLU}(10x_0 - 8x_1 + 0.1x_2)$

- If one weight will be removed, which one?

Pruning Criterion

Weight saliency

- The saliency of a weight can be determined in several ways:

Weight saliency

- The saliency of a weight can be determined in several ways:
  - **Magnitude-Based Pruning**: Weights with smaller absolute values are considered less salient.
    - The assumption is that weights closer to zero have a minimal impact on the activation of neurons.

Weight saliency

- The saliency of a weight can be determined in several ways:
  - **Magnitude-Based Pruning**: Weights with smaller absolute values are considered less salient.
    - The assumption is that weights closer to zero have a minimal impact on the activation of neurons.
  - **Gradient-Based Methods**: Weights that contribute less to the gradient are considered less important.

Weight saliency

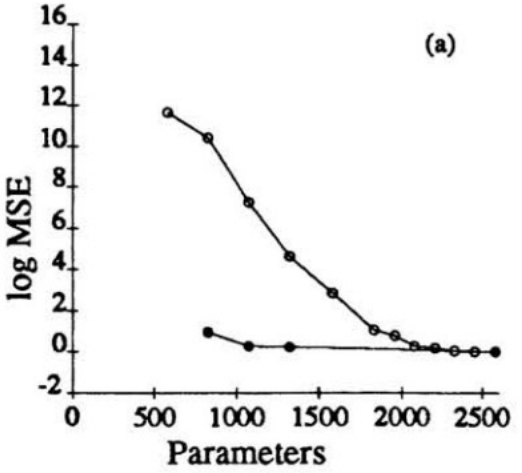
- The saliency of a weight can be determined in several ways:
  - **Magnitude-Based Pruning**: Weights with smaller absolute values are considered less salient.
    - The assumption is that weights closer to zero have a minimal impact on the activation of neurons.
  - **Gradient-Based Methods**: Weights that contribute less to the gradient are considered less important.
  - **Contribution to Output Variation**: how much a weight contributes to the variance in the model's output.
    - Weights that have minimal impact on output variation are deemed less important.

Weight saliency

- The saliency of a weight can be determined in several ways:
  - **Magnitude-Based Pruning**: Weights with smaller absolute values are considered less salient.
    - The assumption is that weights closer to zero have a minimal impact on the activation of neurons.
  - **Gradient-Based Methods**: Weights that contribute less to the gradient are considered less important.
  - **Contribution to Output Variation**: how much a weight contributes to the variance in the model's output.
    - Weights that have minimal impact on output variation are deemed less important.
  - **Second-Order Methods**: Computing the Hessian (the second-order partial derivatives of the loss function) to evaluate how changes in weights affect the loss.
    - Weights that, when perturbed, have minimal impact on the loss according to second-order information might be considered less salient.
    - Second-order methods take into account not just the immediate impact of changing a weight (which would be the first-order derivative or gradient) but also how the rate of change itself changes (which is the second-order derivative).

Prune Based On Hessian

- Optimal Brain Damage
  - Initial Training: Train the network to achieve a satisfactory performance level.
  - Compute the second derivatives of the loss with respect to each weight.
  - Determine the saliency, the importance of weights based on their impact on the loss.
  - Remove weights with the lowest saliency values.
  - Fine-Tune: Re-train the pruned network to restore performance.
  - Iterate: Repeat the pruning and fine-tuning steps as needed.



Objective function (in dB) versus number of parameters, without retraining (upper curve), and after retraining (lower curve).

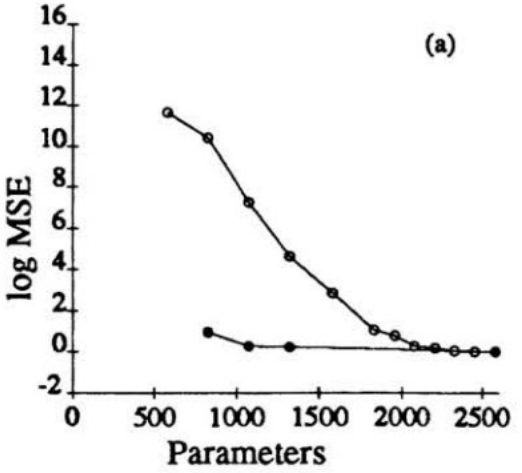
[Lecun, NeurIPS 1989]

Prune Based On Accuracy Impact

- Optimal Brain Damage
  - Initial Training: Train the network to achieve a satisfactory performance level.
  - Compute the second derivatives of the loss with respect to each weight.
  - Determine the saliency, the importance of weights based on their impact on the loss.
  - Remove weights with the lowest saliency values.
  - Fine-Tune: Re-train the pruned network to restore performance.
  - Iterate: Repeat the pruning and fine-tuning steps as needed.

$$S(w) = w^2 \times \frac{\partial^2 E}{\partial w^2}$$

Saliency of weight

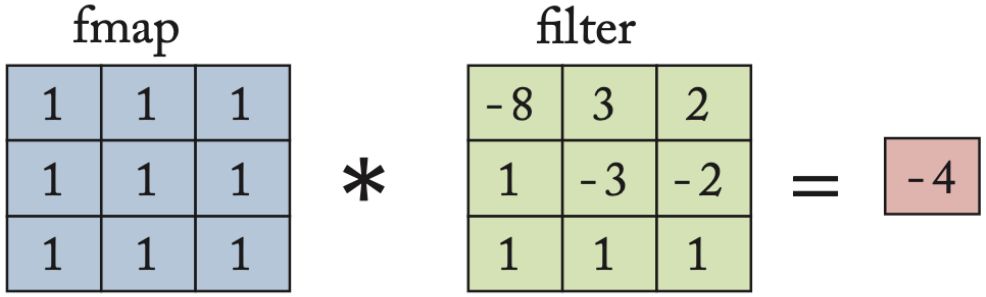


Objective function (in dB) versus number of parameters, without retraining (upper curve), and after retraining (lower curve).

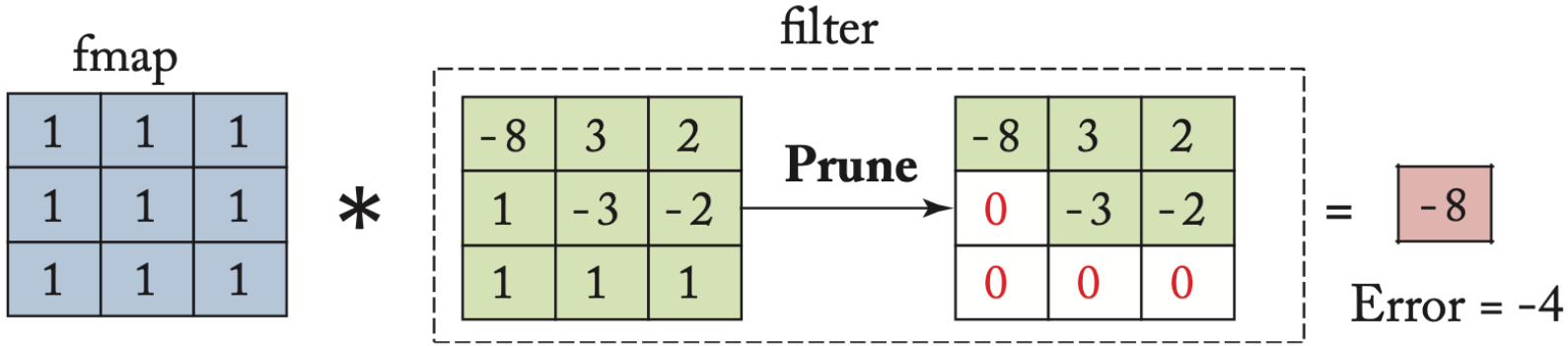
[Lecun, NeurIPS 1989]

Calculating the second derivative to be determined for each weight is expensive.
We actually consider magnitude multiplied by the second derivative of the loss function.

Magnitude Based Pruning



(a) Original without pruning



(b) Pruning based on the magnitude of weights (i.e., magnitude-based pruning)

Small weights are removed

Source: Sze and Emer. Efficient Processing of Deep Neural Networks, 2020 and Yang, CVPR 2017

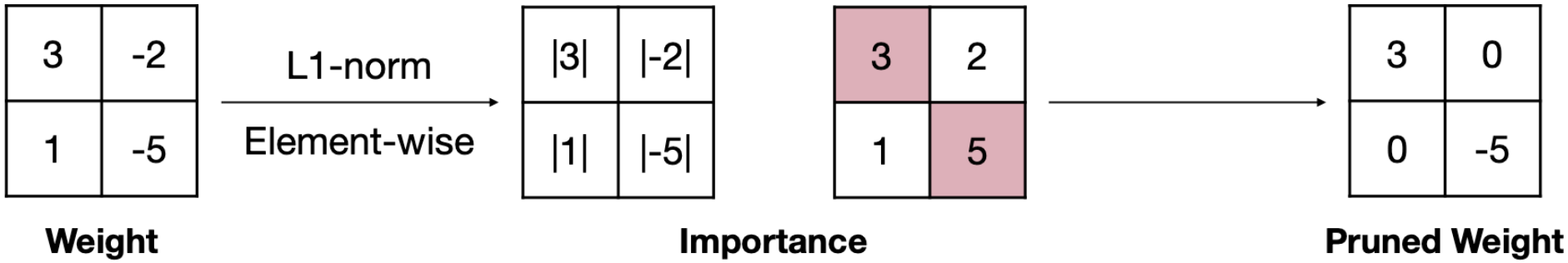
Magnitude Based Pruning

A heuristic pruning criterion

- Magnitude-based pruning considers weights with ***larger absolute values*** are more important than other weights.
  - For element-wise pruning,

$$Importance = |W|$$

- **Example**



Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Slide Credit: Song Han

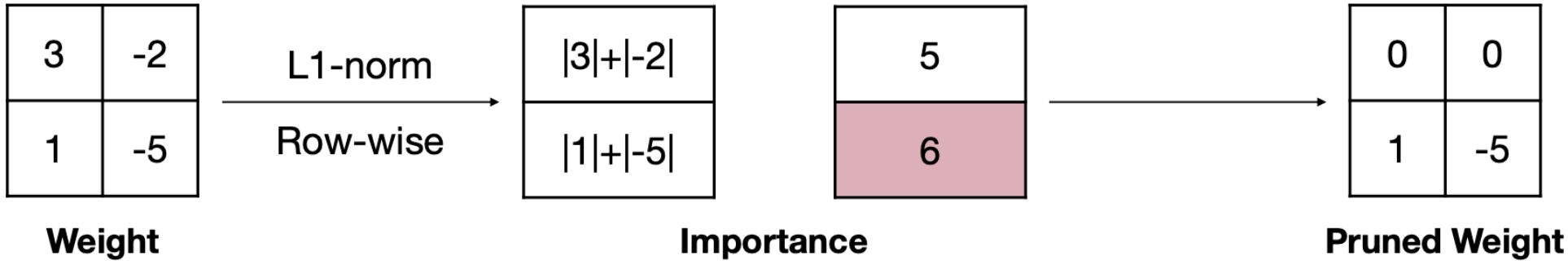
Magnitude Based Pruning

A heuristic pruning criterion

- Magnitude-based pruning considers weights with ***larger absolute values*** are more important than other weights.
  - For row-wise pruning, the L1-norm magnitude can be defined as,

$$Importance = \sum_{i \in S} |w_i|, \text{ where } \mathbf{W}^{(S)} \text{ is the structural set } S \text{ of parameters } \mathbf{W}$$

- **Example**



Learning Both Weights and Connections for Efficient Neural Network [Han *et al*., NeurIPS 2015]

Structural Set: Specific grouping or arrangement of parameters, e.g., weights in a row

Slide Credit: Song Han

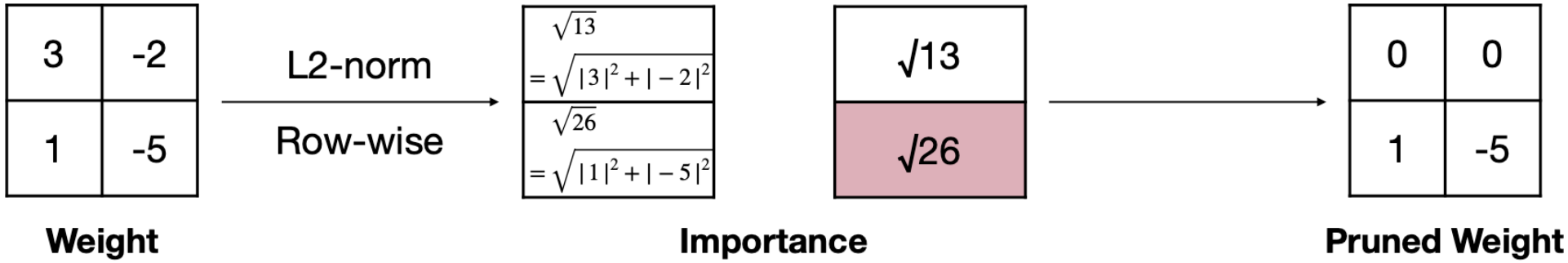
Magnitude Based Pruning

A heuristic pruning criterion

- Magnitude-based pruning considers weights with ***larger absolute values*** are more important than other weights.
  - For row-wise pruning, the L2-norm magnitude can be defined as,

$$Importance = \sqrt{\sum_{i \in S} |w_i|^2}, \text{ where } \mathbf{W}^{(S)} \text{ is the structural set } S \text{ of parameters } \mathbf{W}$$

- **Example**



Learning Both Weights and Connections for Efficient Neural Network [Han *et al*., NeurIPS 2015]

Structural Set: Specific grouping or arrangement of parameters, e.g., weights in a row

Slide Credit: Song Han

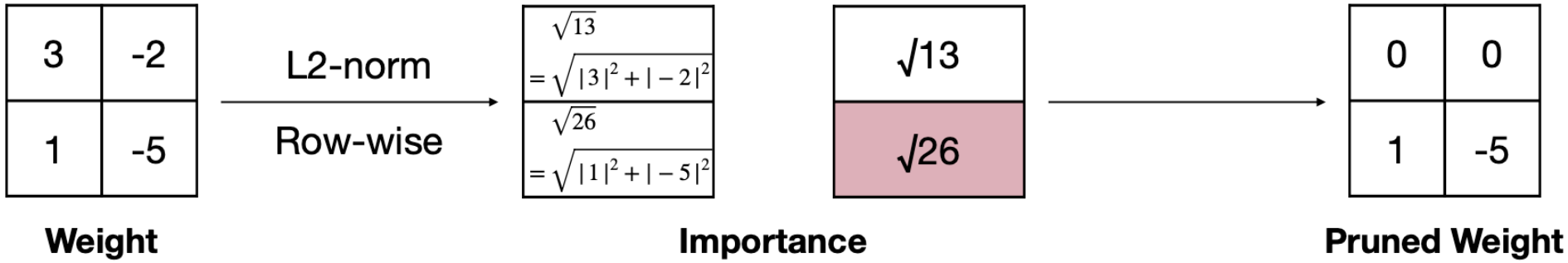
Magnitude Based Pruning

A heuristic pruning criterion

- Magnitude-based pruning considers weights with ***larger absolute values*** are more important than other weights.
- Magnitude is also known as L_p-norm defined as,

$$\|\mathbf{W}^{(S)}\|_p = \left(\sum_{i \in S} |w_i|^p \right)^{\frac{1}{p}}, \text{ where } \mathbf{W}^{(S)} \text{ is a structural set of parameters}$$

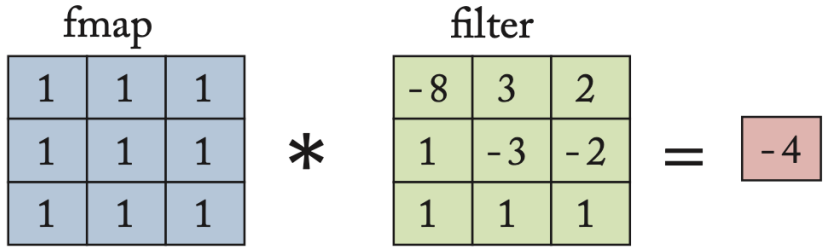
- **Example**



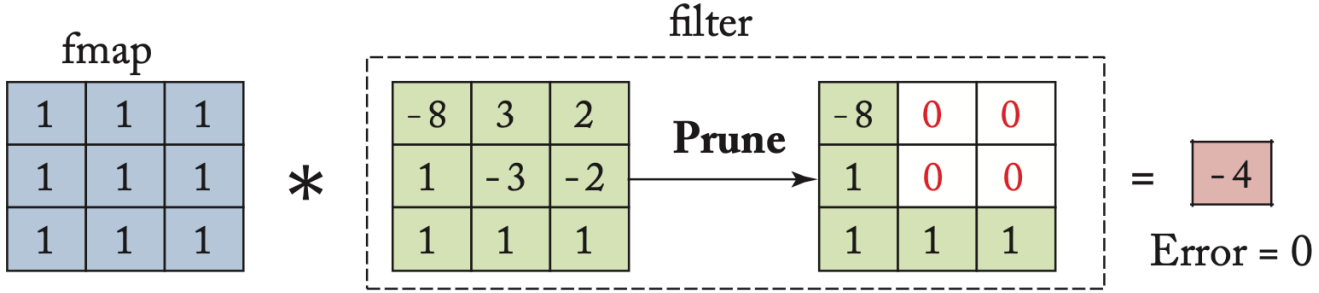
Learning Structured Sparsity in Deep Neural Networks [Wen *et al.*, NeurIPS 2016]

Structural Set: Specific grouping or arrangement of parameters, e.g., weights in a row

Feature-map-based Pruning



(a) Original without pruning



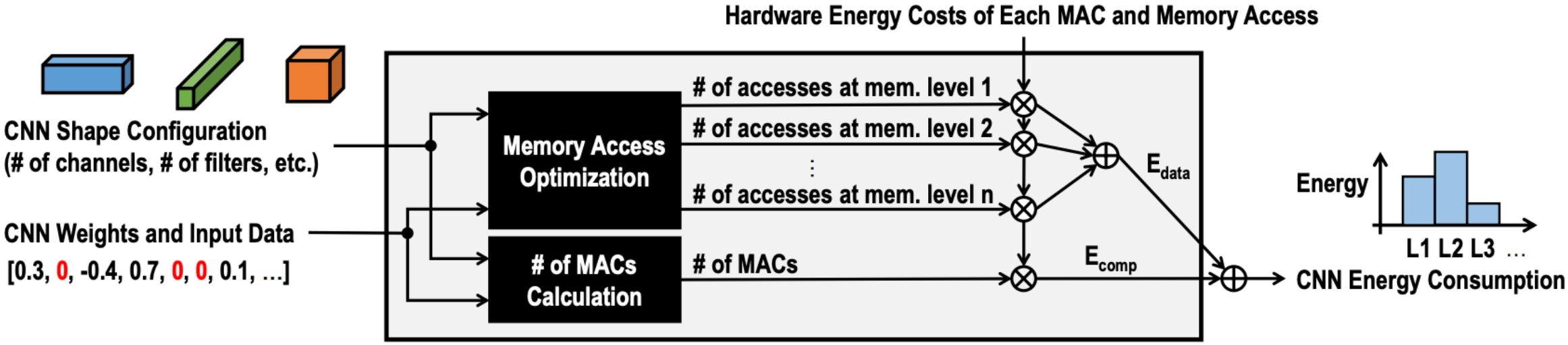
(c) Pruning based on the impact on the output feature map
(i.e., feature-map-based pruning)

Weights with large magnitudes but opposing signs are removed if their effect on the output feature map cancels each other out

- Finding such canceling weights requires additional computation, making this method expensive.
- In large models, identifying these "redundant" weight patterns can be time-consuming.

Source: Sze and Emer. Efficient Processing of Deep Neural Networks, 2020 and Yang, CVPR 2017

Energy Estimation

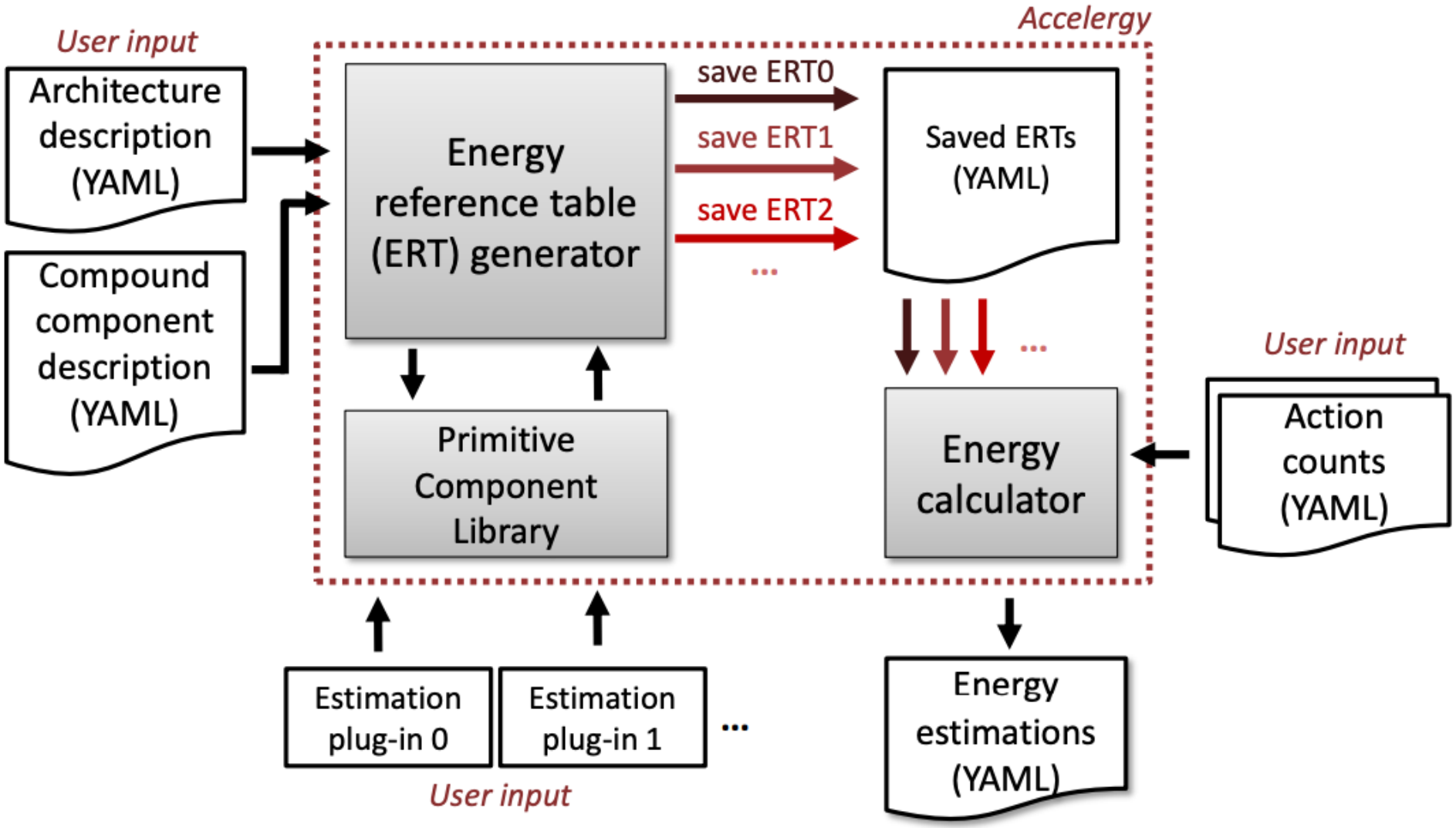


Estimates energy by analyzing the model's structure and operations, incorporating hardware energy consumption metrics for computation and memory access. It accounts for data sparsity and compression, adjusting for zero inputs and reduced bitwidths.

https://energyestimation.mit.edu/

Source: Sze and Emer. Efficient Processing of Deep Neural Networks, 2020 and Yang, CVPR 2017

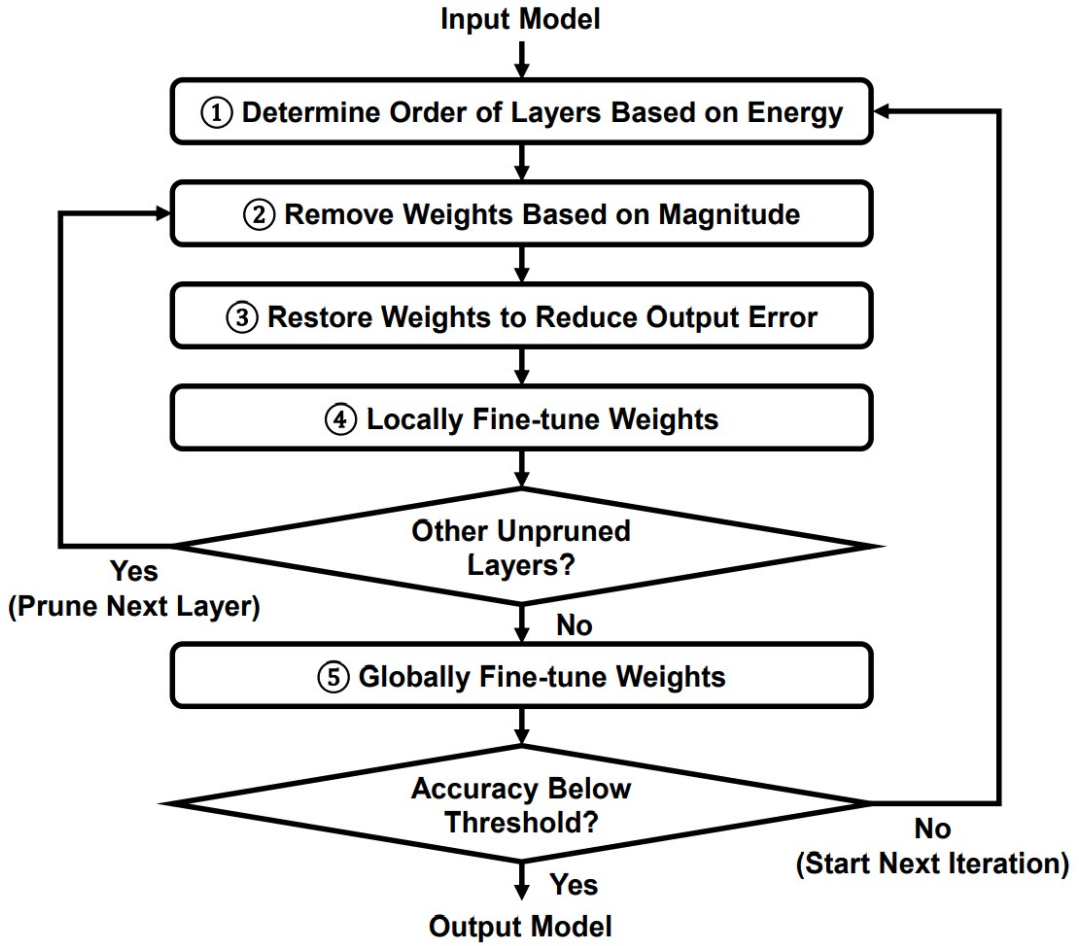
Accelerology



Source: Yannan Nellie Wu et al. Accelerology

Energy Aware Pruning

- **Determine Order of Layers Based on Energy**
  - Layers are ranked based on their energy contribution (such as weight magnitude, importance, or sensitivity).
  - Layers with lower energy are pruned first.
- **Remove Weights Based on Magnitude**
  - Weights with smaller absolute values are considered less important and are removed.
- **Restore Weights to Reduce Output Error**
  - Some removed weights might be restored if their removal leads to a significant accuracy drop.
- **Locally Fine-tune Weights**
  - The remaining weights in the pruned layer are adjusted to compensate for lost information.
- **Check for Other Unpruned Layers**
  - If there are more layers to prune, the process **repeats.**
- **Globally Fine-tune Weights**
  - After all layers have been pruned, the entire model undergoes fine-tuning.
- **Accuracy Check Against Threshold**
  - If the accuracy is still above the acceptable threshold, pruning stops, and the final model is output.
  - If accuracy **drops below** the threshold, a new pruning iteration is started (adjusting pruning strategies or fine-tuning further).



Prune layers that consume the most energy first

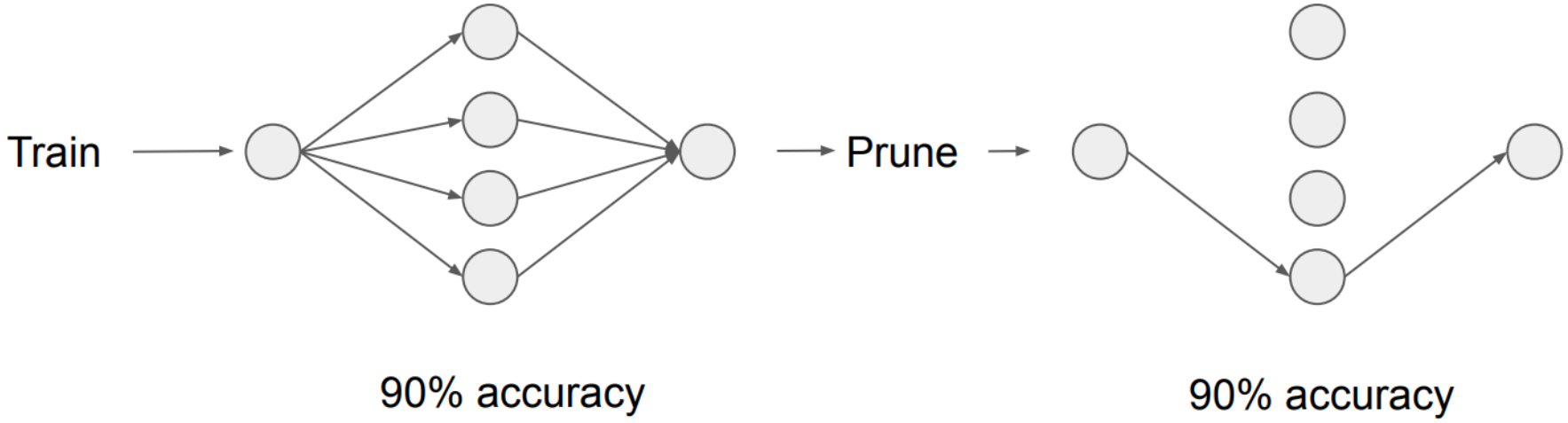
Source: Tien-Ju Yang, et al. Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning

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

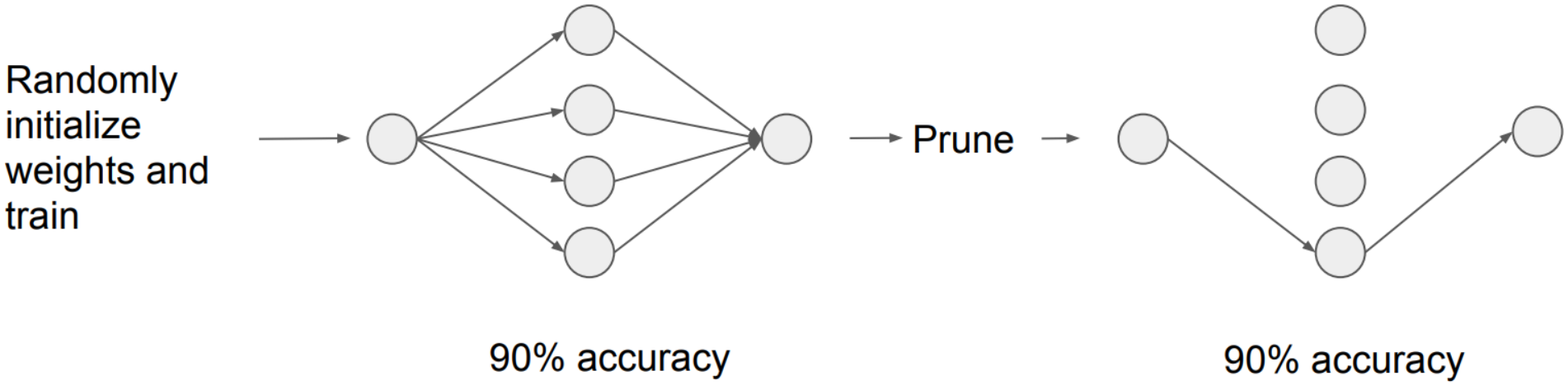
- Motivation
  - Pruning techniques can reduce parameter counts by 90% without harming accuracy



Source: https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

The Lottery Ticket Hypothesis

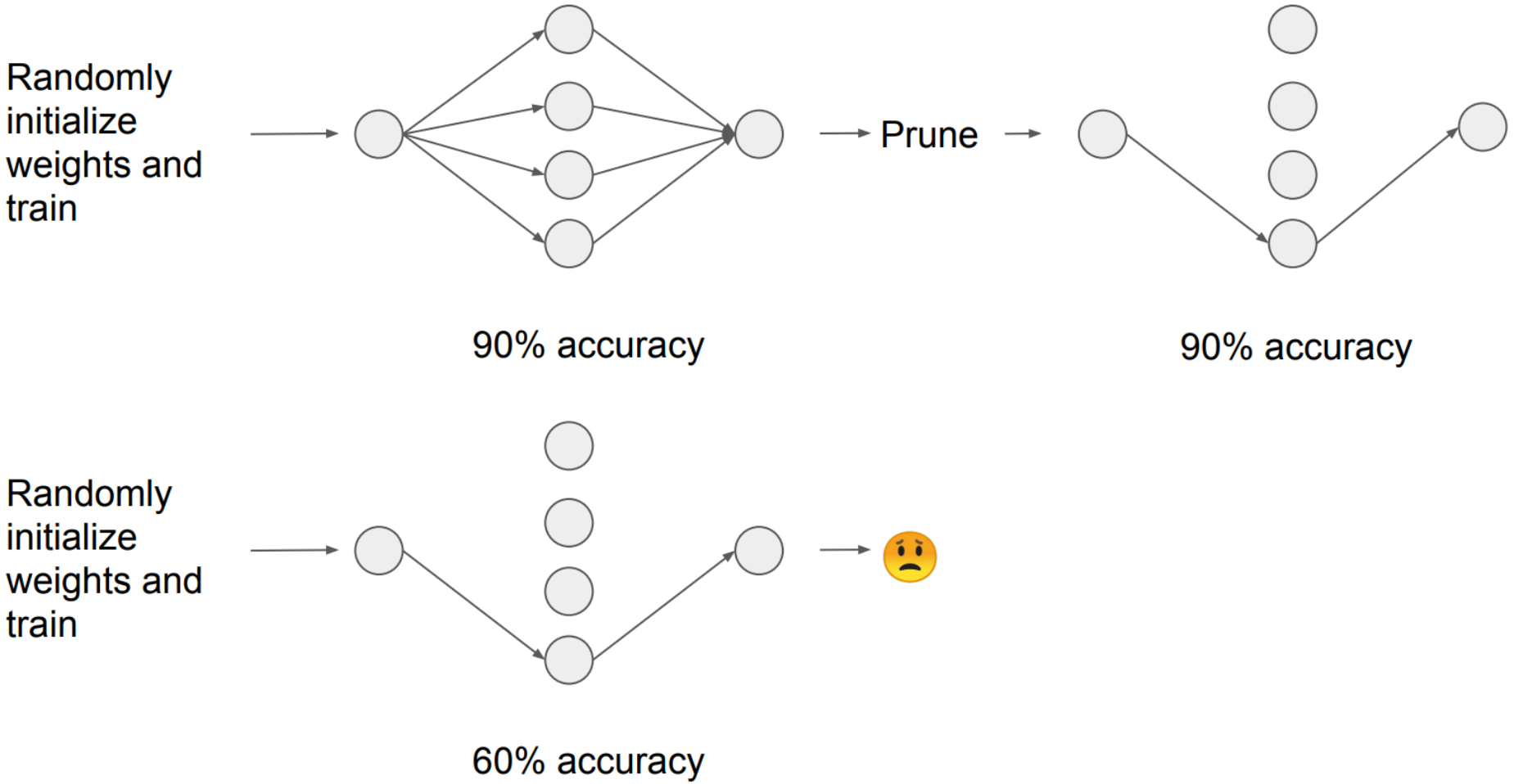
- Motivation
  - Pruning techniques can reduce parameter counts by 90% without harming accuracy



Source: https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

The Lottery Ticket Hypothesis

- Motivation
  - Pruning techniques can reduce parameter counts by 90% without harming accuracy



Source: https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

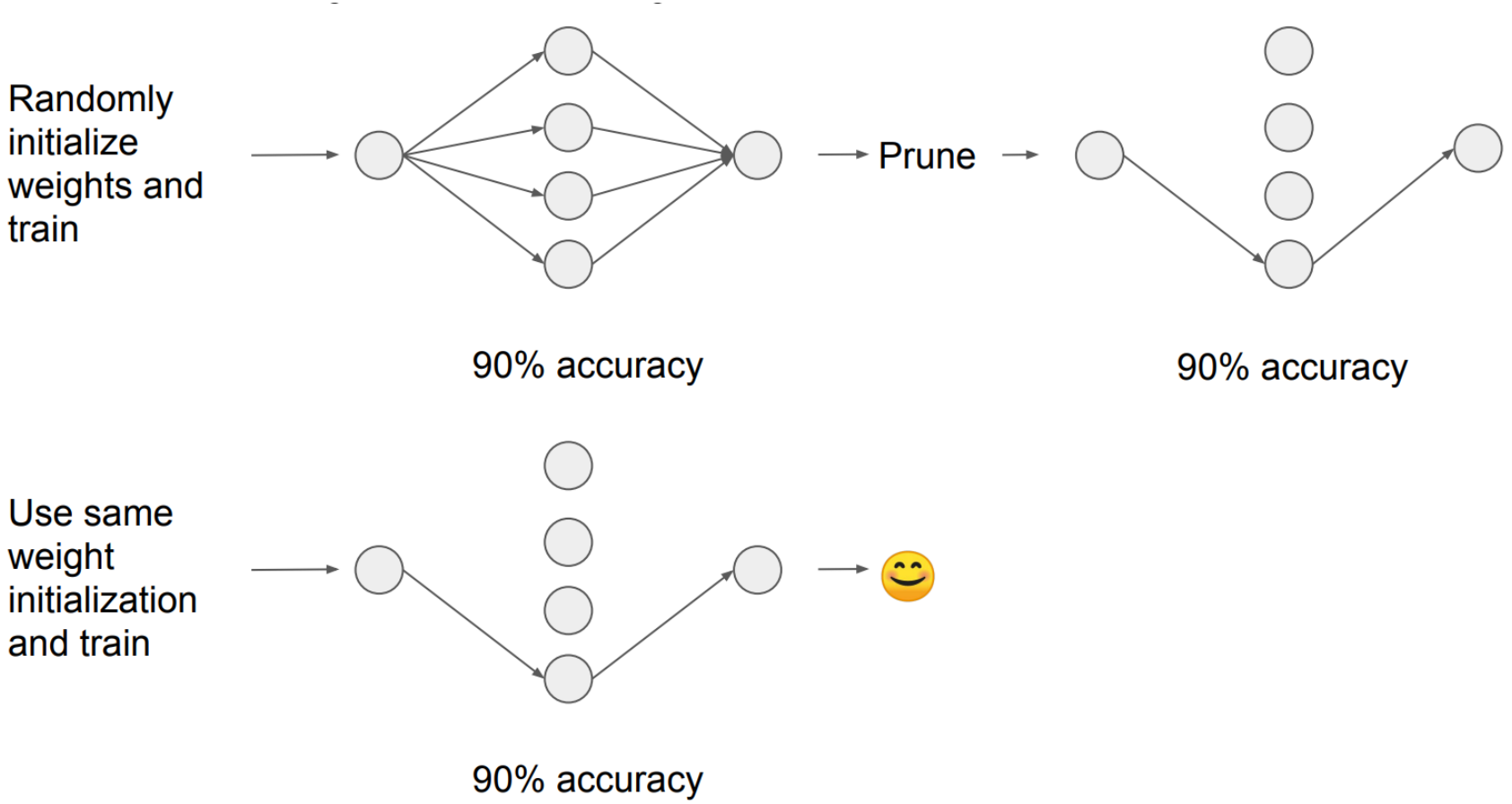
The Lottery Ticket Hypothesis

- If you want to win the lottery, just buy a lot of tickets and some will likely win
- Buying a lot of tickets = having an overparameterized neural network for your task
- Winning the lottery = training a network with high accuracy
- Winning ticket = pruned subnetwork which achieves high accuracy

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

- Motivation
  - Pruning techniques can reduce parameter counts by 90% without harming accuracy



Source: https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

Identifying Winning Tickets

One-shot pruning

1. Randomly initialize a neural network
2. Train the network
3. Prune p%** of weights with lowest magnitude from each layer (set them to 0)
4. Reset pruned network parameters to the original random initialization

Iterative pruning

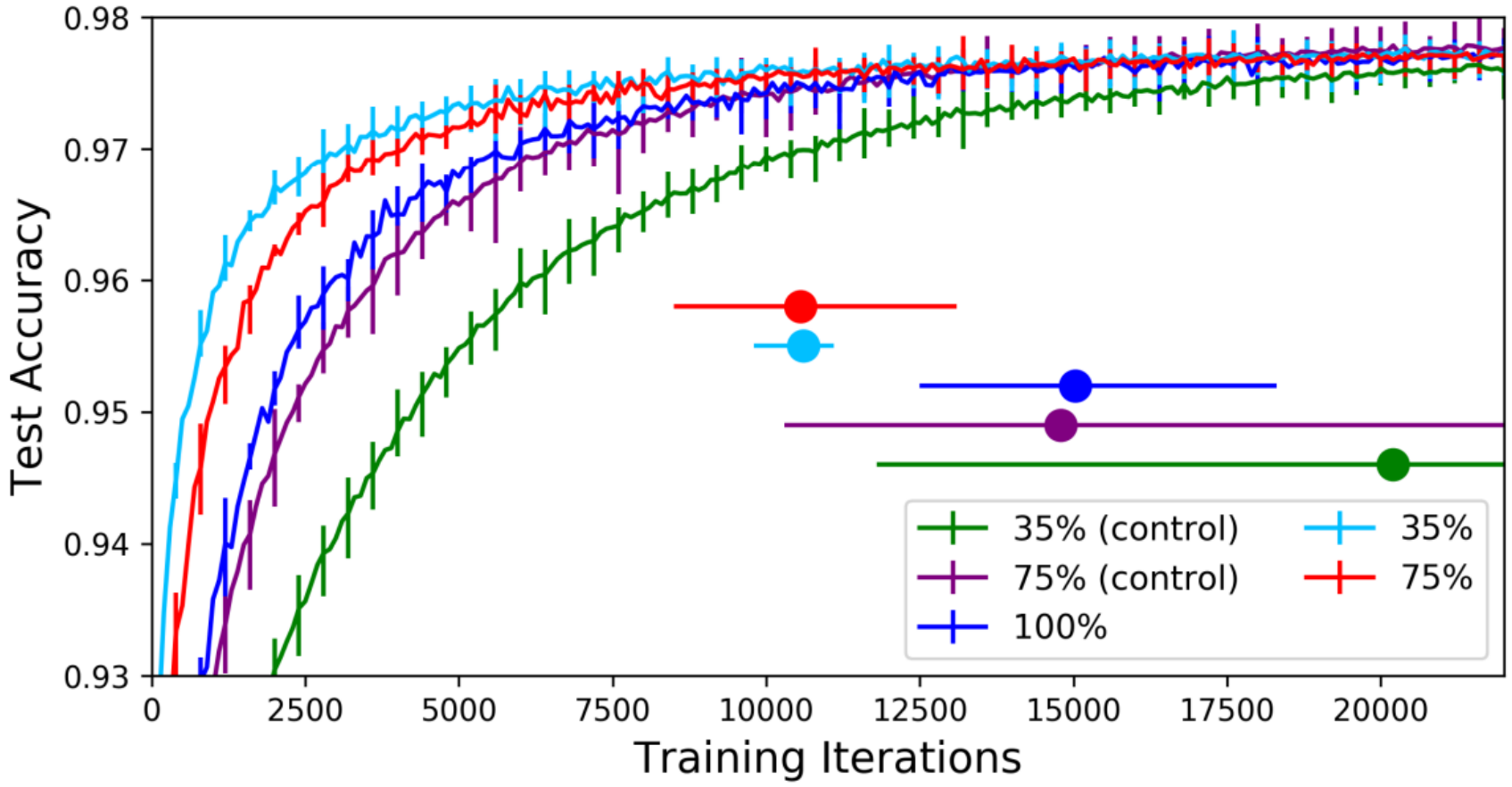
- Iteratively repeat the one-shot pruning process
- Yields smaller networks than one-shot pruning

**Connections to outputs are pruned at 50% of the pruning rate

Source: https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

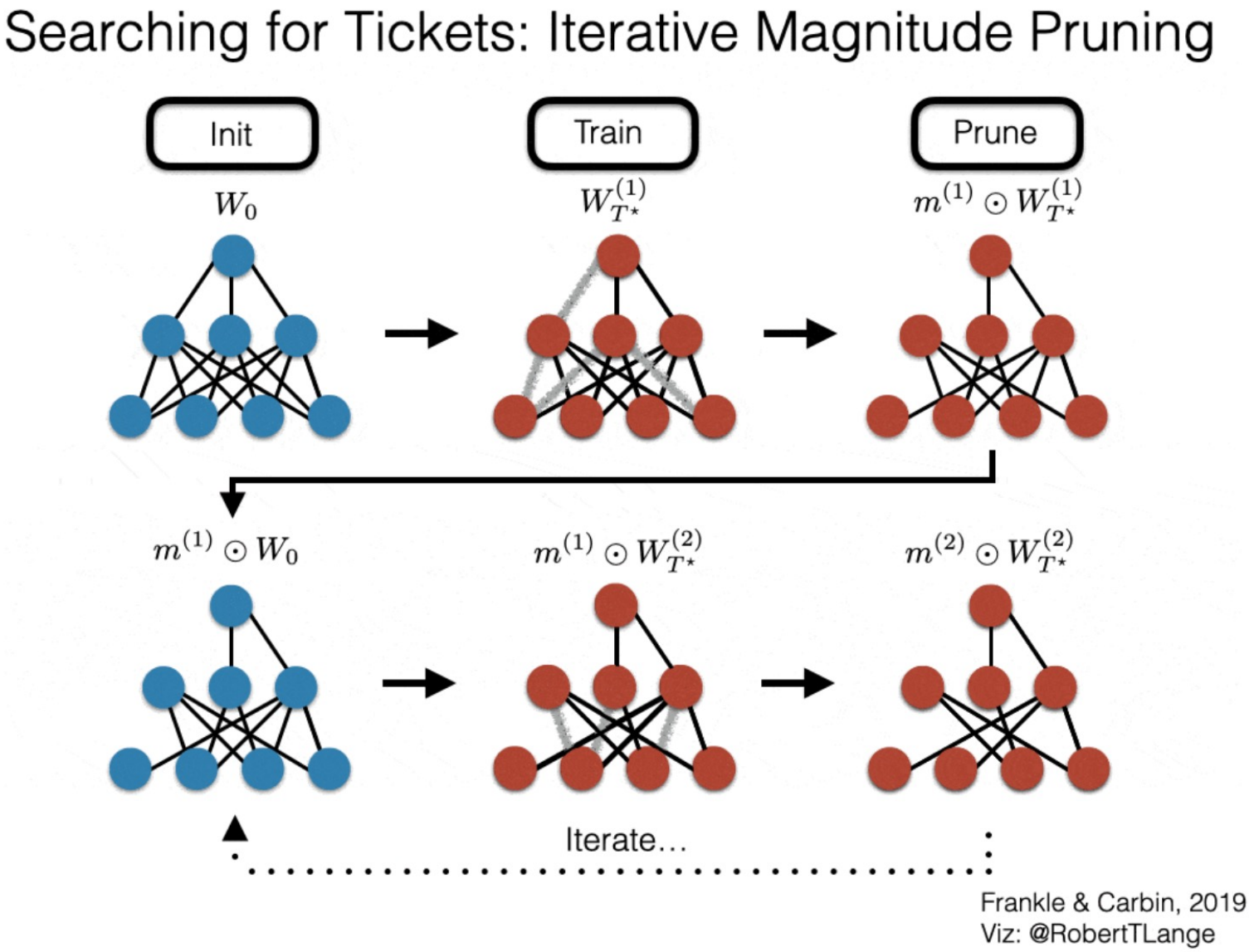
- Retaining the original, randomly assigned weights of the surviving neurons after pruning. This approach diverges from typical pruning strategies that might reinitialize weights.
- Why keep original weights:
  - Critical for success: Original configurations may be more effective for early training stages.
  - Supported by evidence: Studies show networks with original initialization often outperform those with new random weights after pruning.

Experimental Results



- **35% (control) (Green)** – A model with 35% pruning but trained without lottery ticket reinitialization.
- **75% (control) (Purple)** – A model with 75% pruning but trained without lottery ticket reinitialization.
- **100% (Blue)** – The original full model without pruning.
- **35% (Blue) & 75% (Red)** – Pruned models retrained from their original initialization (lottery ticket hypothesis).

Summary


Searching for Tickets: Iterative Magnitude Pruning
Init
Train
Prune
W_0
$W_{T^\star}^{(1)}$
$m^{(1)} \odot W_{T^\star}^{(1)}$
$m^{(1)} \odot W_0$
$m^{(1)} \odot W_{T^\star}^{(2)}$
$m^{(2)} \odot W_{T^\star}^{(2)}$
Iterate…
Frankle & Carbin, 2019
Viz: @RobertTLange

Lottery Ticket Summary

- **Train the Original Network**
  - Start with a **large, randomly initialized** neural network.
  - Train it on a dataset until it reaches good performance.
- **Identify Important Weights (Pruning Step)**
  - Prune away a **fraction of the weights** based on a chosen criterion (e.g., lowest magnitude weights).
  - This reduces the network's size but keeps the most impactful connections.
- **Reset and Reinitialize**
  - Instead of continuing training from the pruned model, **reset the remaining weights to their original values** (i.e., the initialization they had before training).
  - This ensures that the subnetwork **inherits favorable initial conditions**.
- **Train the Pruned Network Again**
  - Train the smaller subnetwork using the same dataset and hyperparameters.
  - If the pruned model achieves similar or better accuracy, then this subnetwork is the **"winning ticket."**
- **Repeat Iteratively**
  - This process can be repeated multiple times, **gradually pruning** more weights while maintaining performance.

Summary

Method	Pruning Strategy	Reinitialization?	Key Benefit
Magnitude-Based Pruning	Removes small-magnitude weights	No (keeps trained weights)	Simple and widely used
Lottery Ticket Pruning	Finds an optimal sparse subnetwork	Yes (resets to original init)	Finds **efficient subnetworks** that generalize well
Gradient-Based Pruning	Prunes weights with low gradient contribution	No	Adaptively removes less useful connections
Structured Pruning	Prunes entire neurons/layers instead of individual weights	No	Maintains hardware efficiency