Review

- BinaryConnect (1-bit weight)
  - Update real value weights by using gradients of binary weights
- Binarized Neural Network (1-bit weight and 1-bit activation)
  - Use **straight-through estimator (STE)** for back propagation
  - Update real value weights by using gradients of binary weights
- XNOR-NET (1-bit weight + real value scalar and 1-bit activation + real value scalar)
  - Quantize CNN that works well on ImageNet
  - Update real value weights by using gradients of real value weights (STE!)
  - Computation overhead reduction for input value scaling factors
- Quantization for integer-only inference
  - Integer-only multiplication (offline calculation of scaling factor)
  - Weight folding for batch normalization
  - Quantization-aware training (learning weights and quantization parameters together)

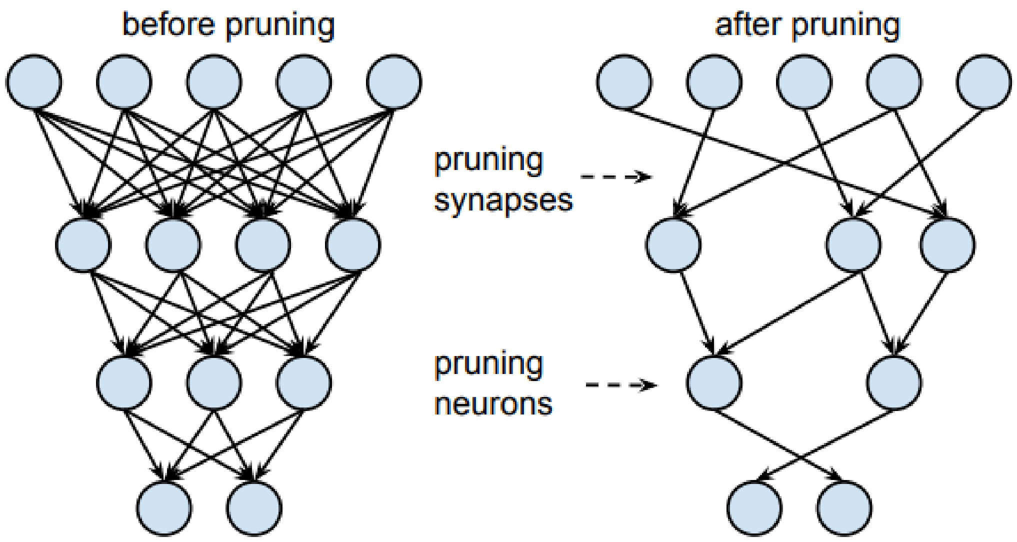
DNN Pruning

Lecture 8

Hyung-Sin Kim

SNU Graduate School of Data Science

Let’s Cut!



[The figure is from S. Han et al., “Learning both weights and connections for efficient neural network.”]

Han et.al [NIPS'15] – Motivation

- LeNet-5 (1MB), AlexNet Caffe (>200MB), VGG-16 Caffe (>500MB)
- Running a heavy DNN is not only memory consuming but also <u>energy consuming</u>!

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**

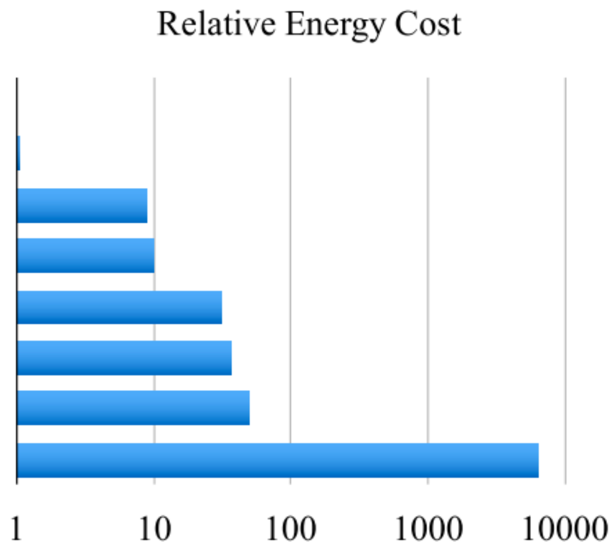
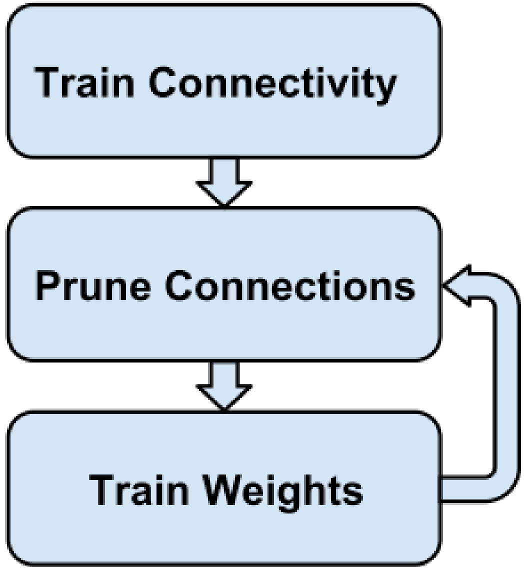


Figure 1: Energy table for 45nm CMOS process [7]. Memory access is 3 orders of magnitude more energy expensive than simple arithmetic.

[The figure is from S. Han et al., "Learning both weights and connections for efficient neural network."]

Han et.al [NIPS’15] – Approach

- Step 1) Train connectivity
  - Just normal training, not for learning the final weights but for figuring out which connections are important
- Step 2) Prune connections
  - Prune low-weight connections with weights below a certain **threshold**
  - Convert a dense network to a sparse network
- Step 3) Retrain weights
  - Train the final values in the spare network
  - Without this retraining, accuracy would be significantly dropped
- Steps 2 and 3 can be repeated!



[The figure is from S. Han et al., “Learning both weights and connections for efficient neural network.”]

Han et.al [NIPS'15] – Approach

- Dropout
  - Different dropout ratio before and after pruning to maintain capacity
  - Number of connections toward layer i: $C_i = N_{i-1}N_i$
    - Original number of connections toward layer i: $\boldsymbol{C_{io}}$
    - After-retraining number of connections toward layer i: $\boldsymbol{C_{ir}}$
  - Dropout ratio for retraining: $D_{ir} = D_{io}\sqrt{\frac{C_{ir}}{C_{io}}}$
    - Why square root?: dropout is for **nodes**, not connections

Han et.al [NIPS'15] – Approach

- Local pruning and adaptation to avoid gradient vanishing problem
  - Fix Conv layers and prune/retrain FC layers
  - Fix FC layers and prune/retrain Conv layers

- Pruning **nodes** (a byproduct)
  - If a node's all input (or output) connections or output connections are zero, all of its output (or input) connections will be zero automatically in the retraining process
  - Zero gradient since these connections do not impact loss at all
  - Only the regularization term will push the weights to zero

Han et.al [ICLR'16] – Storing Pruned Model

- Storing every element of a weight matrix having many zeros is inefficient
  - $$\begin{bmatrix} 1.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.6 & 3.4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 3.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.9 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.7 & 0.0 & 0.0 & -0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.1 & 0.0 \end{bmatrix}$$
  - 90 numbers! (n x m)
  - We want to store only non-zero values and indicate where they are
- Compressed sparse row (CSR) format
  - VALUE = [1.2 0.6 3.4 3.0 0.9 2.7 -0.2 2.1]
  - COL_INDEX = [0 3 4 5 5 7 10 13]
  - ROW_INDEX = [0 1 3 4 5 7 8]
  - 23 numbers (2a + n + 1)

Han et.al [ICLR'16] – Storing Pruned Model

- Compressed sparse row (CSR) format

(0,0) (1,3) (1,4) (2,5) (3,5) (4,7) (4,10) (5,13)

- VALUE = [1.2 0.6 3.4 3.0 0.9 2.7 -0.2 2.1]
- COL_INDEX = [0 3 4 5 5 7 10 13]
- ROW_INDEX = [0 1 3 4 5 7 8]

Row 0 has 1 element

Row 1 has 2 elements

Row 2 has 1 element

Row 3 has 1 element

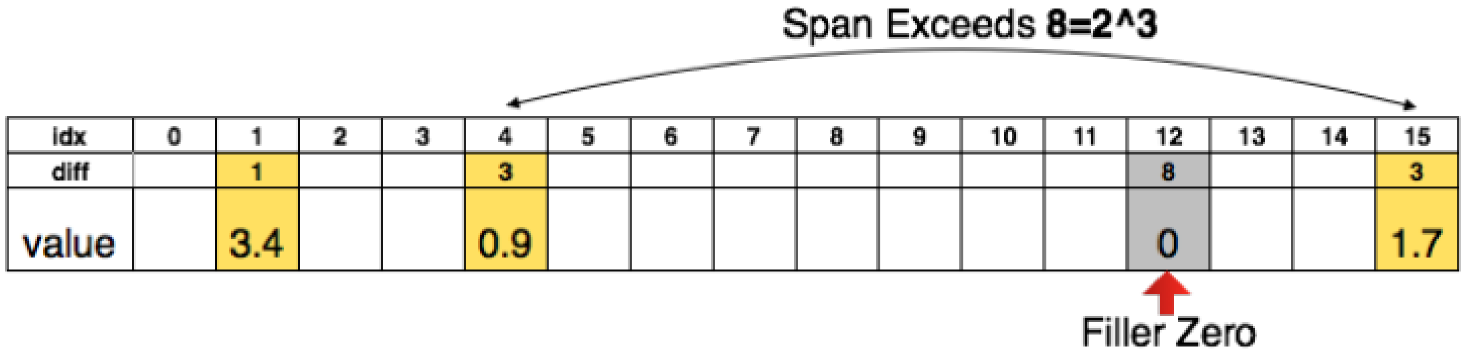
Row 4 has 2 elements

Row 5 has 1 element

Han et.al [ICLR'16] – Storing Pruned Model

- Express COL_INDEX vector by using absolute column positions
  - COL_INDEX = [0 3 4 5 5 7 10 13] (needs **4 bits** to express 0~15)

- Express COL_INDEX vector by using **position differences**
  - COL_INDEX = [0 3 1 1 0 2 3 3] (needs **2 bits** to express 0~3)
  - Using filler technique when position differences are larger than 3

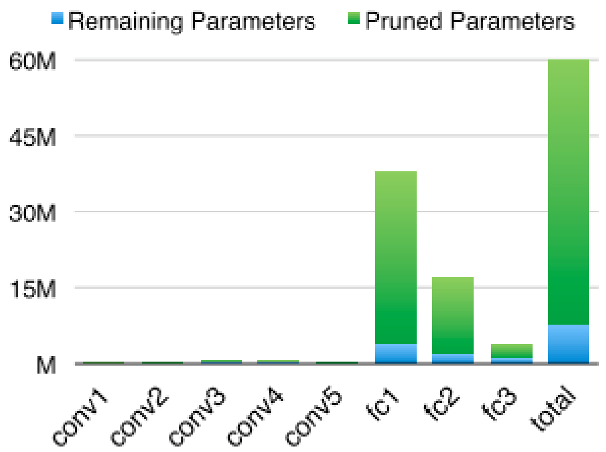


[The figure is from S. Han et al., "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding."]

Han et.al [NIPS'15] – Evaluation

- AlexNet on ImageNet
  - 75 hours for initial training and 173 hours for retraining (1/100 learning rate)
    - So long for retraining… but it is OK because retraining is not part of model prototyping
    - Pruning is used when a model is ready for deployment: retraining is a one-time process
  - Accuracy: Top1 57.2%, Top5 80.3%
  - Model size: 1/9 (See there are much more redundant parameters in FC layers)
  - Computation: 1/3

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



[The figures are from S. Han et al., "Learning both weights and connections for efficient neural network."]

Han et.al [NIPS'15] – Evaluation

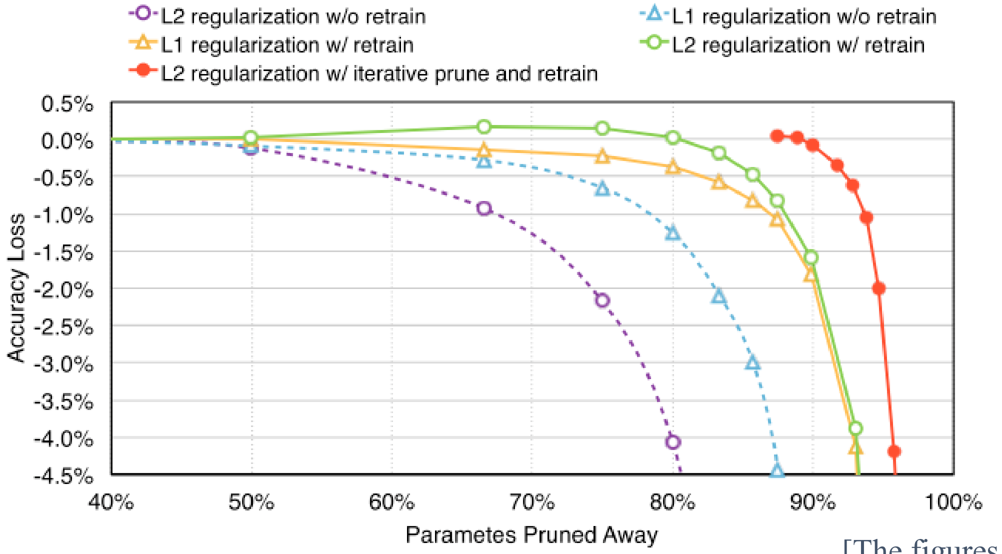
- VGG-16
  - 5 iterations of pruning and retraining
  - Model size: 1/12 (Much more pruned weights in FC layers)
  - Computation: 1/5
  - Accuracy:

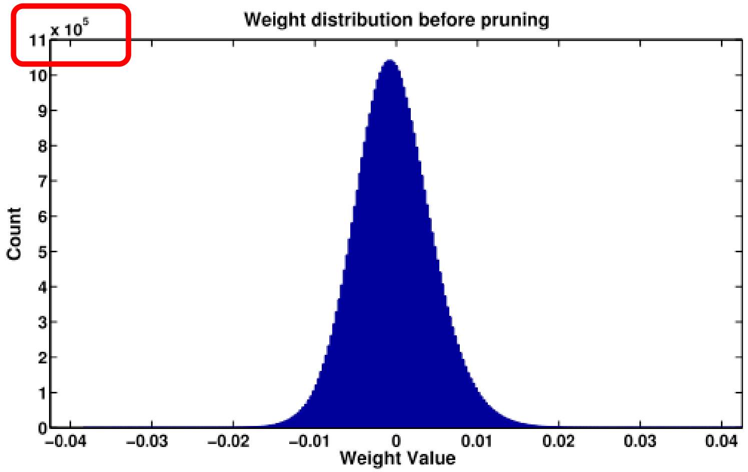
Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1_1	2K	0.2B	53%	58%	58%
conv1_2	37K	3.7B	89%	22%	12%
conv2_1	74K	1.8B	80%	34%	30%
conv2_2	148K	3.7B	81%	36%	29%
conv3_1	295K	1.8B	68%	53%	43%
conv3_2	590K	3.7B	70%	24%	16%
conv3_3	590K	3.7B	64%	42%	29%
conv4_1	1M	1.8B	51%	32%	21%
conv4_2	2M	3.7B	45%	27%	14%
conv4_3	2M	3.7B	34%	34%	15%
conv5_1	2M	925M	32%	35%	12%
conv5_2	2M	925M	29%	29%	9%
conv5_3	2M	925M	19%	36%	11%
fc6	103M	206M	38%	4%	1%
fc7	17M	34M	42%	4%	2%
fc8	4M	8M	100%	23%	9%
total	138M	30.9B	64%	**7.5%**	**21%**

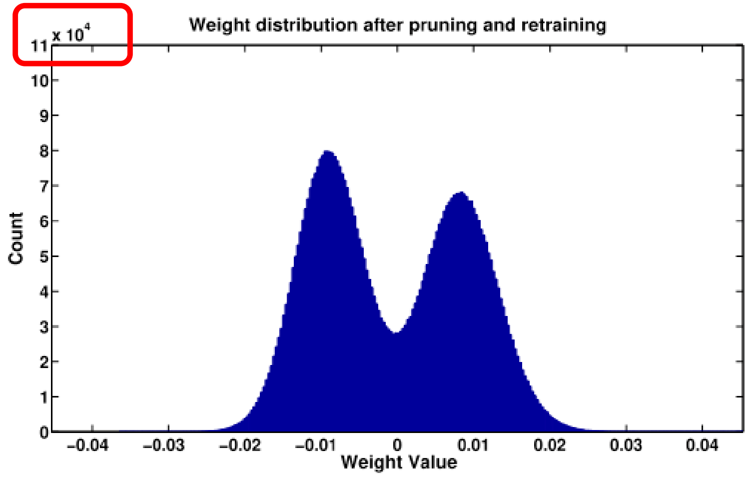
[The figures are from S. Han et al., "Learning both weights and connections for efficient neural network."]

Han et.al [NIPS’15] – Evaluation

- Retraining is effective
- L2 regularization is better than L1 regularization
- Iterative pruning is the best
- After pruning, parameter distribution of AlexNet’ changed from [-0.15, 0.15] to [-0.25, 0.25] withou







[The figures are from S. Han et al., “Learning both weights and connections for efficient neural network.”]

*Now we know that pruning **iteratively** performs better.*
Then, what is an effective way of iterative pruning?

Gradual Pruning [arxiv'17] – Introduction

- Question: Given a bound on the model's memory footprint, how can we arrive at the most accurate model?

- Compare two approaches
  - **Large-sparse model**: Train a large model.,,, and prune aggressively to satisfy the memory requirement
  - **Small-dense model**: Train a small model, and prune moderately

- By using a new **gradual pruning** method

Gradual Pruning [arxiv’17] – Approach

- For a layer determined to be pruned, prune the connection with smallest weights until desired sparsity level **s** is reached (no weight threshold)
    - Instead of making these values directly zero, this approach adds a **binary mask**

- Back propagation at the binary masks
    - The gradients flow through the binary masks, but the masked weights are not updated

Gradual Pruning [arxiv'17] – Approach

- Gradually increase sparsity
  - $s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad \text{for} \quad t \in \{t_0,\ t_0 + \Delta t,\ \ldots,\ t_0 + n\Delta t\}$
  - Train 100% network for Δt
  - 1st prune and retrain for Δt
  - 2nd prune retrain for Δt
  - …
  - n-th prune and retrain for Δt
  - Prune the network **rapidly first** and **gradually reduce** the number of pruned weights

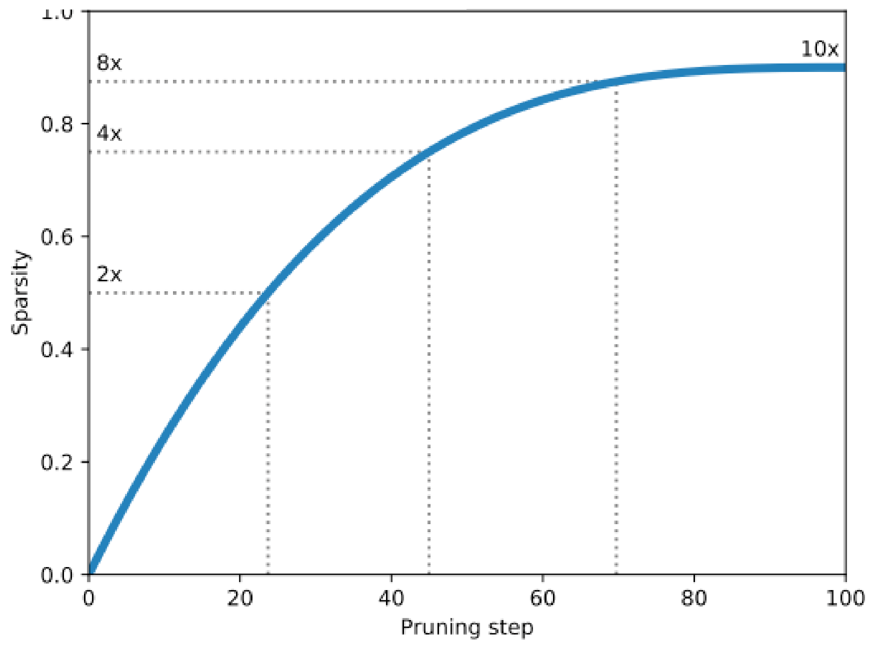


Figure 1: Sparsity function used for gradual pruning

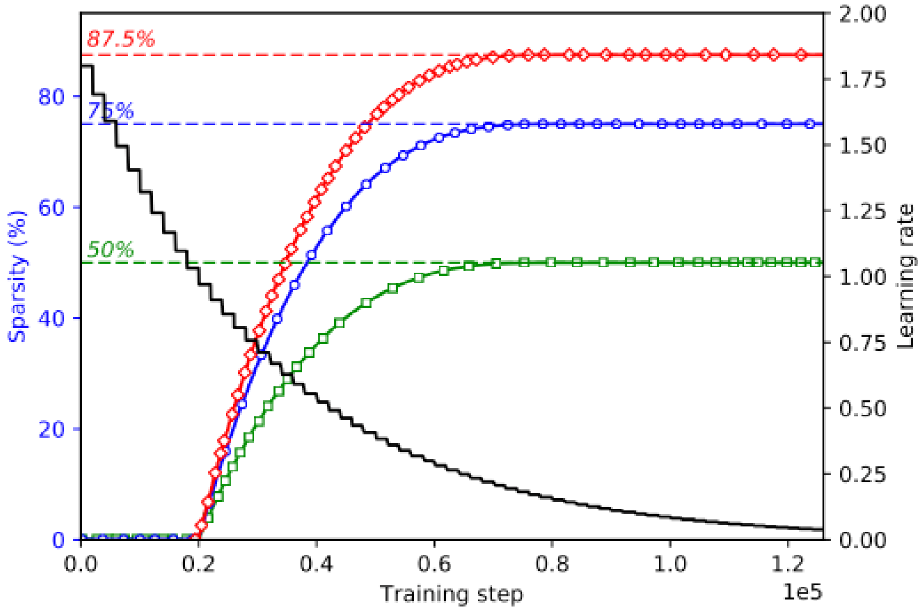
[The figure is from M. Zhu et al., "To prune, or not to prune: Exploring the efficacy of pruning for model compression."]

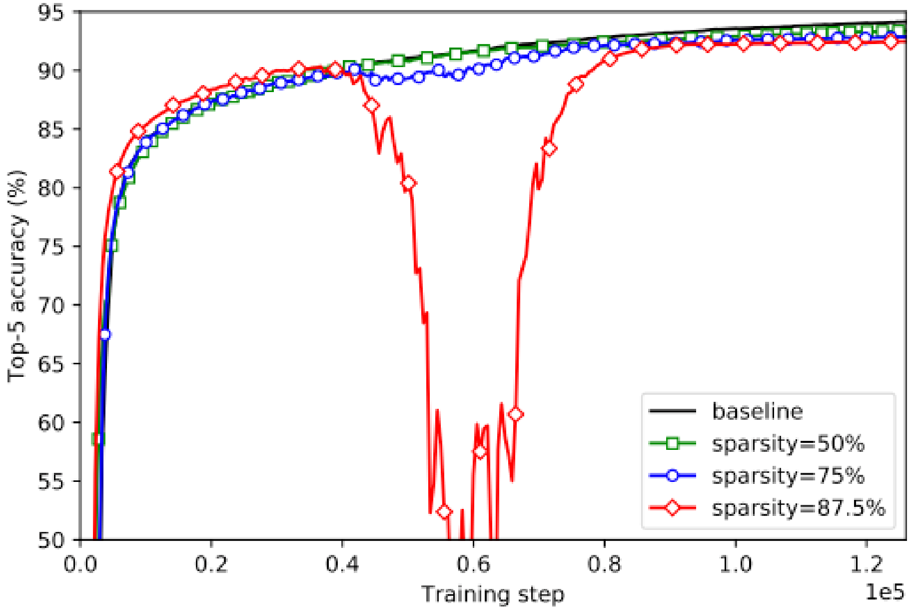
Gradual Pruning [arxiv'17] – Hyperparameters

- t_0 : How long will a model be pretrained before being pruned
- n: How many times will a model be pruned until the target sparsity is reached
    - Should be set considering learning rate schedule
        - When learning rate is too high, it is possible to prune weights before they are converged
        - When learning rate is too low, it is hard to retrain the model after being pruned

Gradual Pruning [arxiv'17] – Hyperparameters

- Applying to InceptionV3
  - If pruning when learning rate is reasonably high, training is done robustly
  - Even when pruning loss is severe, it is recovered fast





[The figures are from M. Zhu et al., "To prune, or not to prune: Exploring the efficacy of pruning for oedel compression."]

Gradual Pruning [arxiv'17] – Performance

- Large sparse vs. Small dense
  - Sparse InceptionV3 (3.3M, 74.6%) outperforms MobileNets full (4.21M, 70.6%)
  - Sparse MobileNets 1.0 (0.25M, 53.6%) outperforms MobileNets 0.25 (0.46M, 50.6%)

Table 1: Model size and accuracy tradeoff for sparse-InceptionV3

Sparsity	NNZ params	Top-1 acc.	Top-5 acc.
0%	27.1M	78.1%	94.3%
50%	13.6M	78.0%	94.2%
75%	6.8M	76.1%	93.2%
87.5%	3.3M	74.6%	92.5%

Table 2: MobileNets sparse vs dense results

Width	Sparsity	NNZ params	Top-1 acc.	Top-5 acc.
0.25	0%	0.46M	50.6%	75.0%
0.5	0%	1.32M	63.7%	85.4%
0.75	0%	2.57M	68.4%	88.2%
1.0	0%	4.21M	70.6%	89.5%
	50%	2.13M	69.5%	89.5%
	75%	1.09M	67.7%	88.5%
	90%	0.46M	61.8%	84.7%
	95%	0.25M	53.6%	78.9%

[The tables are from M. Zhu et al., "To prune, or not to prune: Exploring the efficacy of pruning for oedel compression."]

Gradual Pruning [arxiv'17] – Implications

- Researchers have designed lots of DNN architectures to make it small and accurate
- A DNN architecture given by pruning a large, redundant model may be better than a novel, human-designed DNN architecture…
- This is the pruning technique in TensorFlow Lite!

TensorFlow Lite

If a DNN architecture found by pruning is really better than human-designed DNN architectures,

Why not just training the weights of the pruned network ***from scratch****, instead of just fine tuning them?*

Frankle & Carbin [ICLR'19]

- 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

- Identifying winning tickets
  - Randomly initialize a large DNN
  - Repeat pruning process **n** times
    - Train the DNN for a while and prune $\boldsymbol{p^{1/n}}$% of the weights (p% is the target pruning rate)
  - **Two approaches**
    - **Reset** the remaining weights to the **initial values** and **retrain** them
    - **Randomly re-initiate** the remaining weights and **retrain** them

Frankle & Carbin [ICLR'19]

- Iterative pruning VGG-19 and training the pruned network for given iterations
  - Learning rate 0.1 (high): Approaches 1 and 2 perform similarly
  - Learning rate 0.01 (low): Approach 1 performs better
    - Low LR performs better at 30k iterations, but high LR performs better at higher iterations
  - Learning rate 0.1 (high) with warmup: Increasing from 0 to 0.1 for the first 10k iterations
    - Best performance, approach 1 is better than approach 2

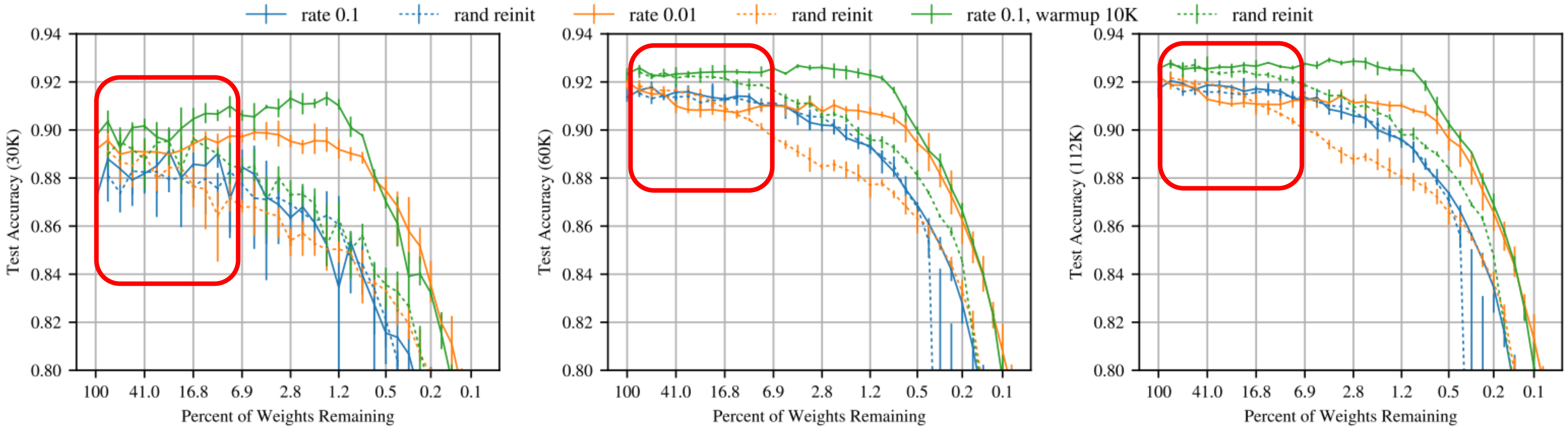


Figure 7: Test accuracy (at 30K, 60K, and 112K iterations) of VGG-19 when iteratively pruned.

[The figures are from J. Frankle et al., "The lottery ticket hypothesis: Finding sparse, trainable neural networks."]

Liu et al. [ICLR’19] – Contradictory Results

- Approach 2 can perform better than Approach 1
  - Random re-initialization does not degrade performance!
  - Results can vary depending on how the model is trained (hyperparameters…)

- VGG-16 and ResNet-50
  - Initial learning rate = 0.1 or 0.01
  - Momentum SGD

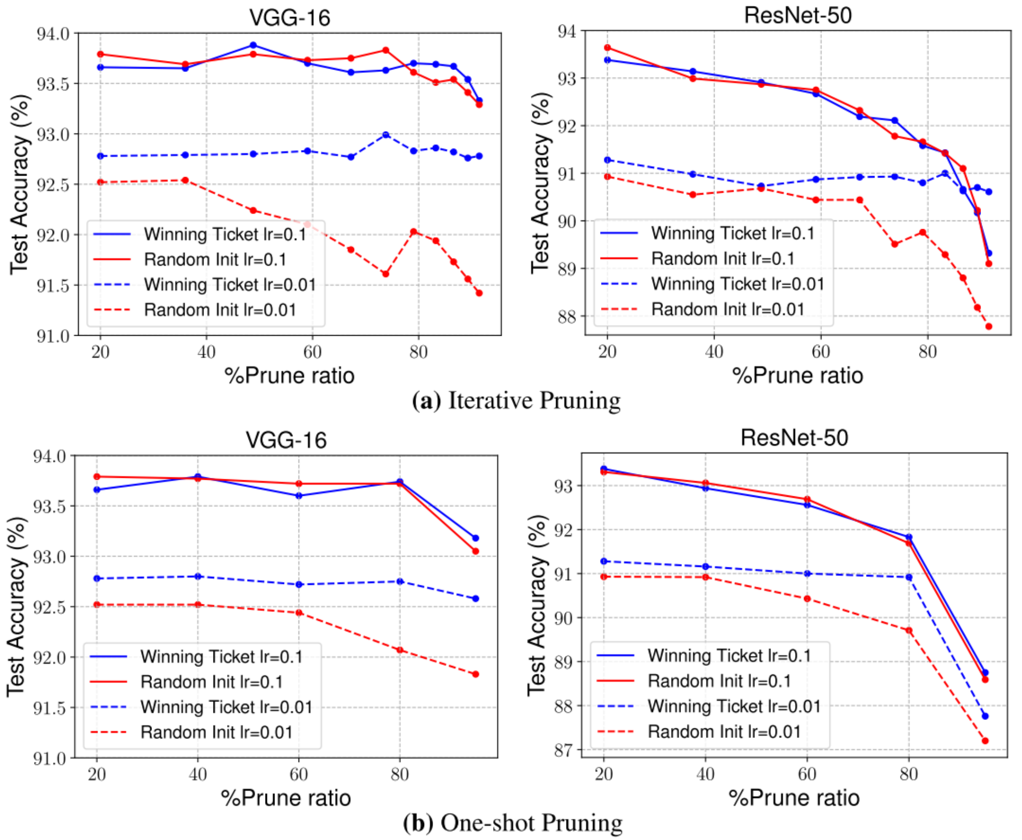


Figure 7: Comparisons with the Lottery Ticket Hypothesis (Frankle & Carbin, 2019) for iterative/one-shot unstructured pruning (Han et al., 2015) with two initial learning rates 0.1 and 0.01, on CIFAR-10 dataset. Each point is averaged over 5 runs. Using the winning ticket as initialization only brings improvement when the learning rate is small (0.01), however such small learning rate leads to a lower accuracy than the widely used large learning rate (0.1).

[The figures are from Z. Liu et al., “Rethinking the value of network pruning.”]

Anyway… we can see that pruning is good for searching efficient neural network architecture!

Thanks!