CS 521: Topics in PL

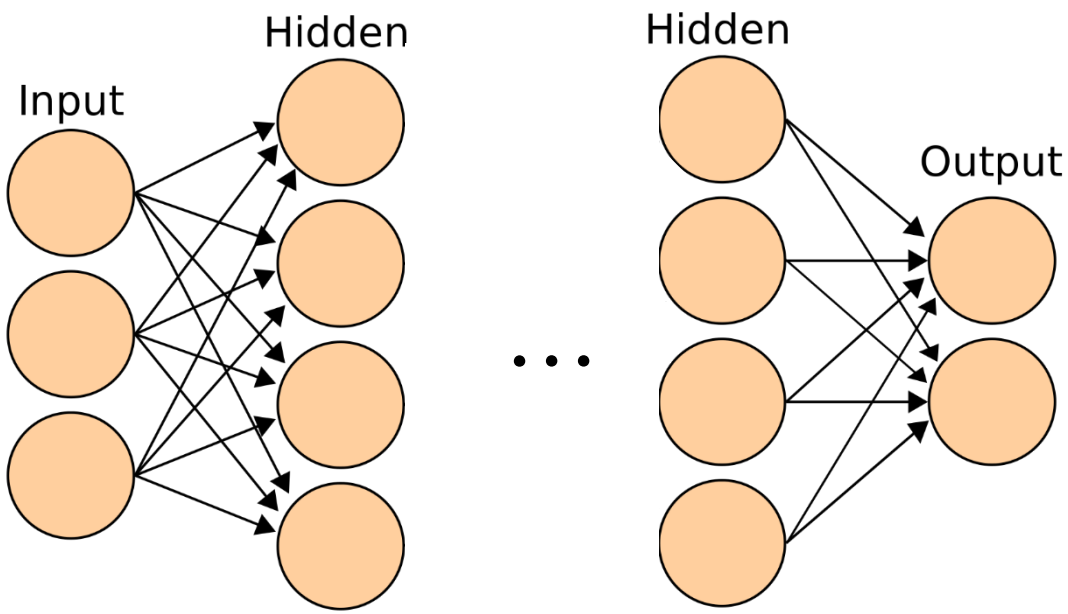
Probabilistic & Approximate Computing

http://misailo.web.engr.Illinois.edu/courses/cs521

QUICK RECAP:
DEEP LEARNING INFERENCE

(BASED ON WILLIAM DALLY'S NIPS 2015 TUTORIAL)

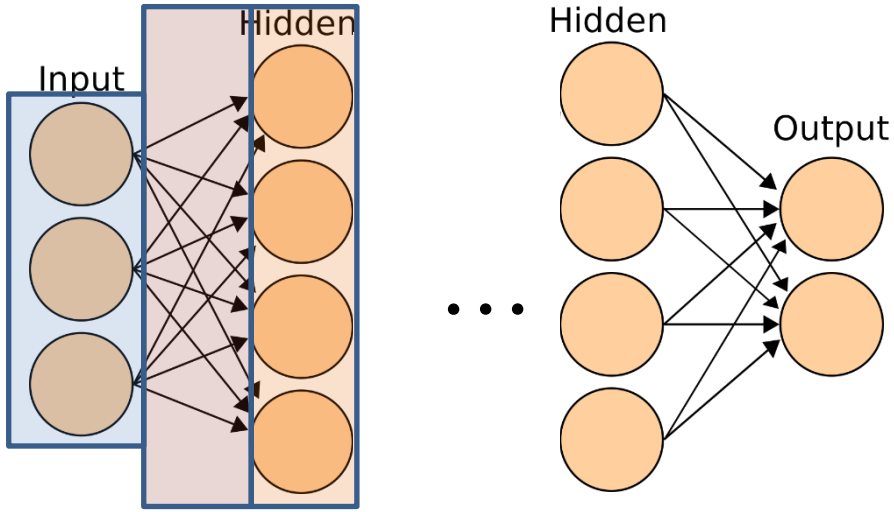
Neural Networks

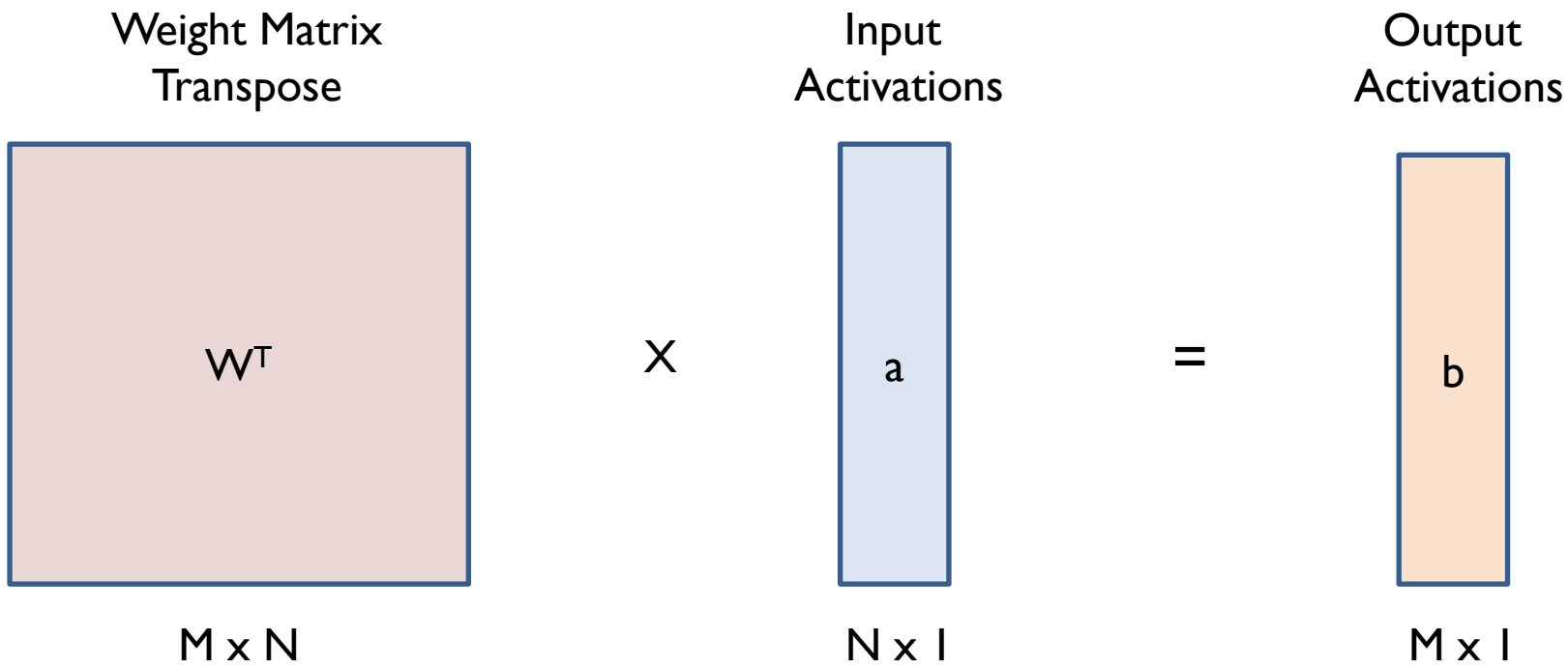


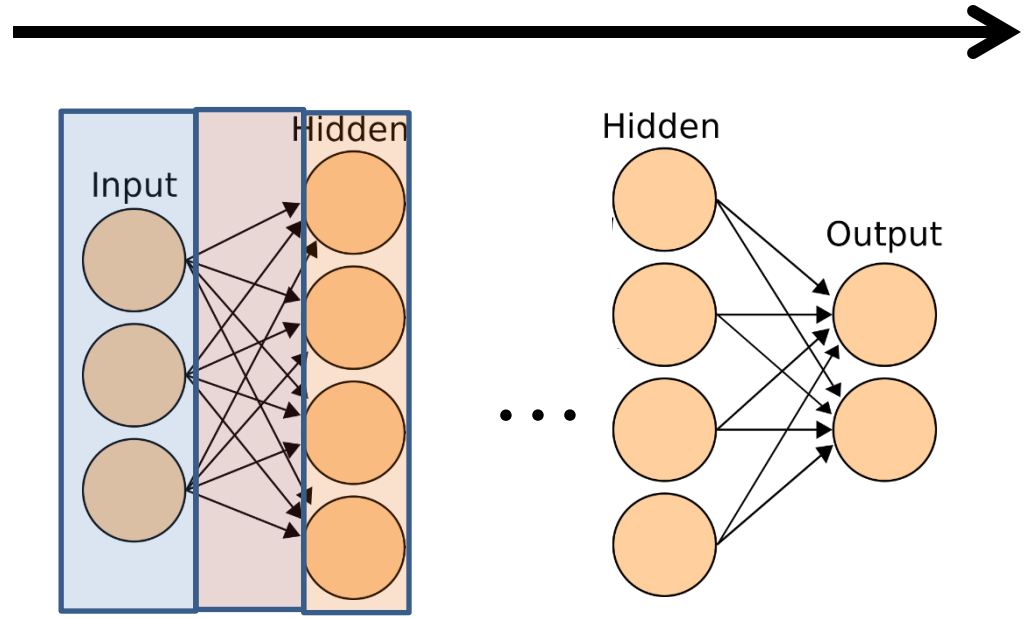
- Neuron
- Synapse
- Layers
- Depth
- Weights
- Activation

Matrix Multiplication

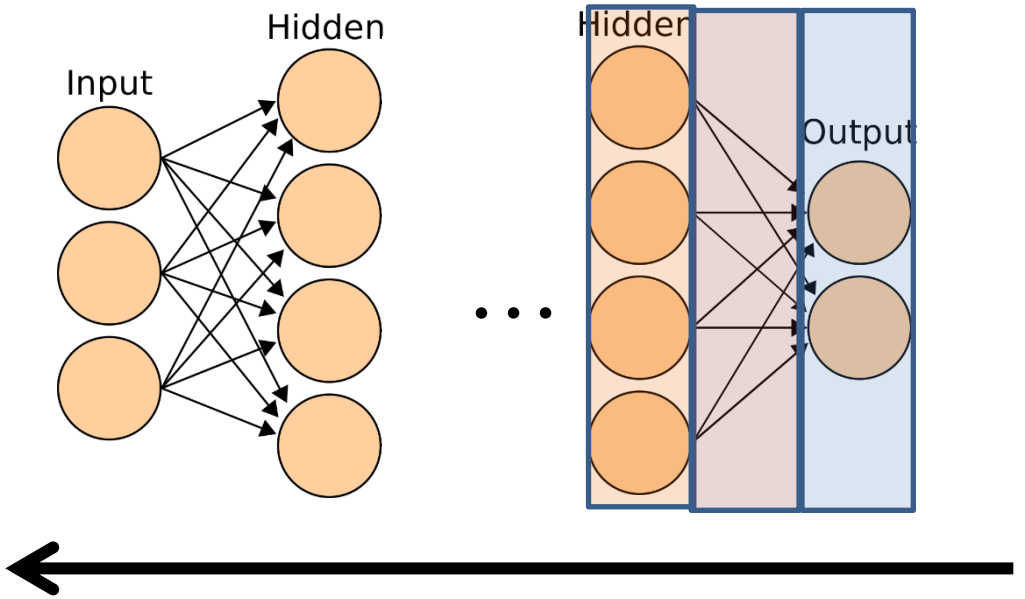
$$W^T \cdot a = b$$







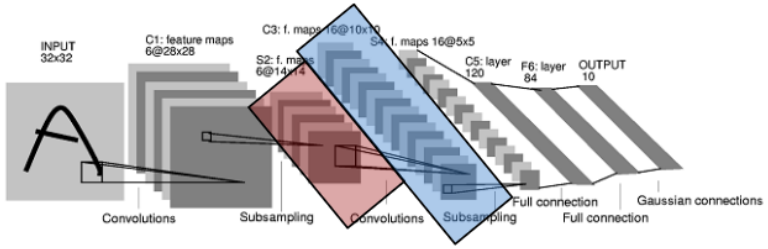
Forward Pass:
Does matrix multiply for each layer



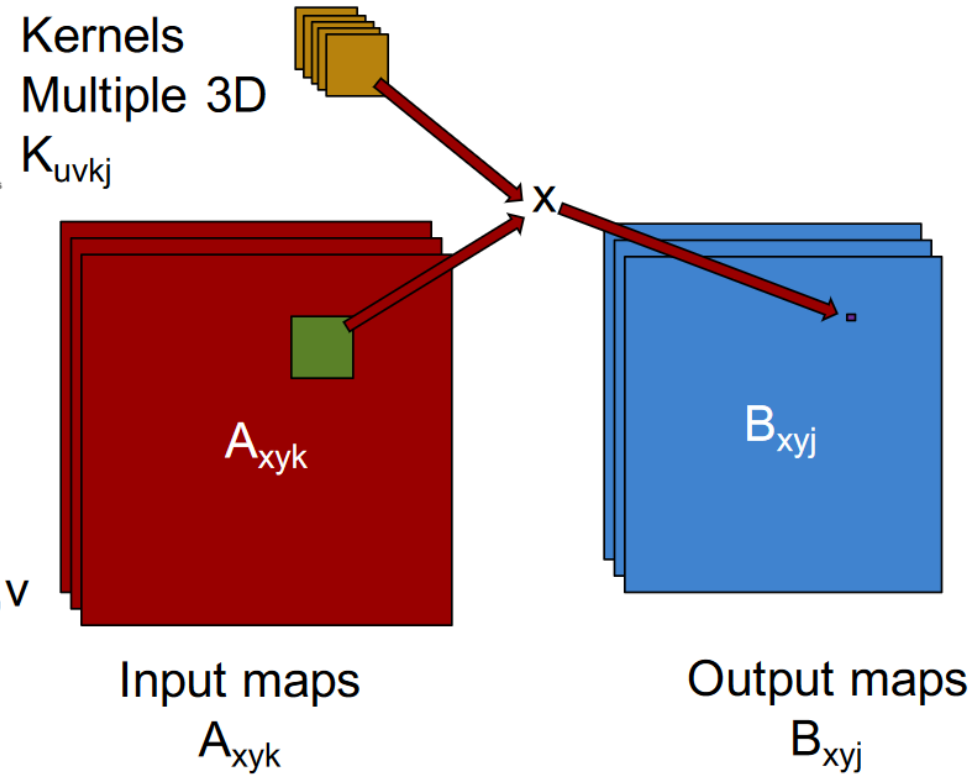
Back-propagation:
Does matrix multiply for each layer, just in the opposite direction

Convolutional Neural Networks

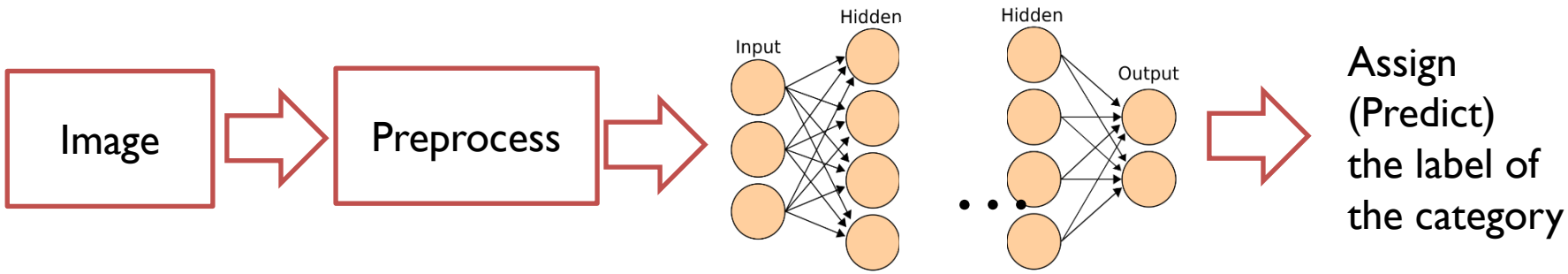
- Convolutional stages act as trained feature detectors
- Operations require convolution + matrix multiply



6D Loop
For each output map j
  For each input map k
    For each pixel x,y
      For each kernel element u,v
        $B_{xyj} \mathrel{+}= A_{(x-u)(y-v)k} \times K_{uvkj}$



Full Classification Workflow



Accuracy of prediction:

- Top-1: How often is the predicted label equal to true label
- Top-5: Check if the true label is one of the predicted labels with top 5 scores ("probabilities")

These scores are computed as the #matched inputs / #total inputs

Accuracy loss metric: $\text{AccLoss} = \text{Accuracy}_{\text{original}} - \text{Accuracy}_{\text{optimized}}$ (for approximation)

Supervised Learning

Given:

- Probability distribution D
- Random variable $\mathrm{Z} \sim D$
- Domain of samples X and labels Y
- Hypothesis space H – set of functions $f : X \to Y$

Minimize generalization error:

- Loss function: $L(\boldsymbol{f}) = \Pr_D [\, \boldsymbol{f}(Z) \neq \mathrm{truelabel}(Z) \,]$

Supervised Learning

Common: The functions $\mathrm{f}_w \in H$ expressed using a weight vector w

- Can be a hyperplane separating between samples of two classes
- In NNs, w is defined in multiple layers

The optimization problem is now:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w} \in \boldsymbol{H}}{\mathbf{argmin}}\, \boldsymbol{L}(\boldsymbol{f}_{\boldsymbol{w}}) = \mathbb{E}_{\boldsymbol{Z} \sim \boldsymbol{D}}(\boldsymbol{l}(\boldsymbol{w}, \boldsymbol{Z}))$$

l is the loss of an individual sample.

Supervised Learning

l is the loss of an individual sample. Some examples:

- **Regression:** square loss $l = \left(\mathrm{f_w(z)} - \mathrm{truevalue(z)}\right)^2$
- **Classification 0-1:** $l = 0$ if $\mathrm{f_w(z)} = \mathrm{truevalue(z)}$ else 1
  (simple but not continuous or diff.)
- **Classification cont. and diff** – use softmax: $\mathrm{p_i} = \frac{\exp(z_i)}{\sum \exp(z_k)}$
- **Classification binary – log loss**:
  $$l = -\log(p) \quad \text{if truelabel} = 1 \quad \text{else} \quad -\log(1-\mathrm{p})$$
- **Classification multiple labels:** compute cross entropy

APPROXIMATING DNNS

- PRUNING
- QUANTIZATION
- TRAINING CONSIDERATIONS

Pruning

Techniques for making weights and activations sparse

Accuracy vs. Space(+Computation) tradeoffs

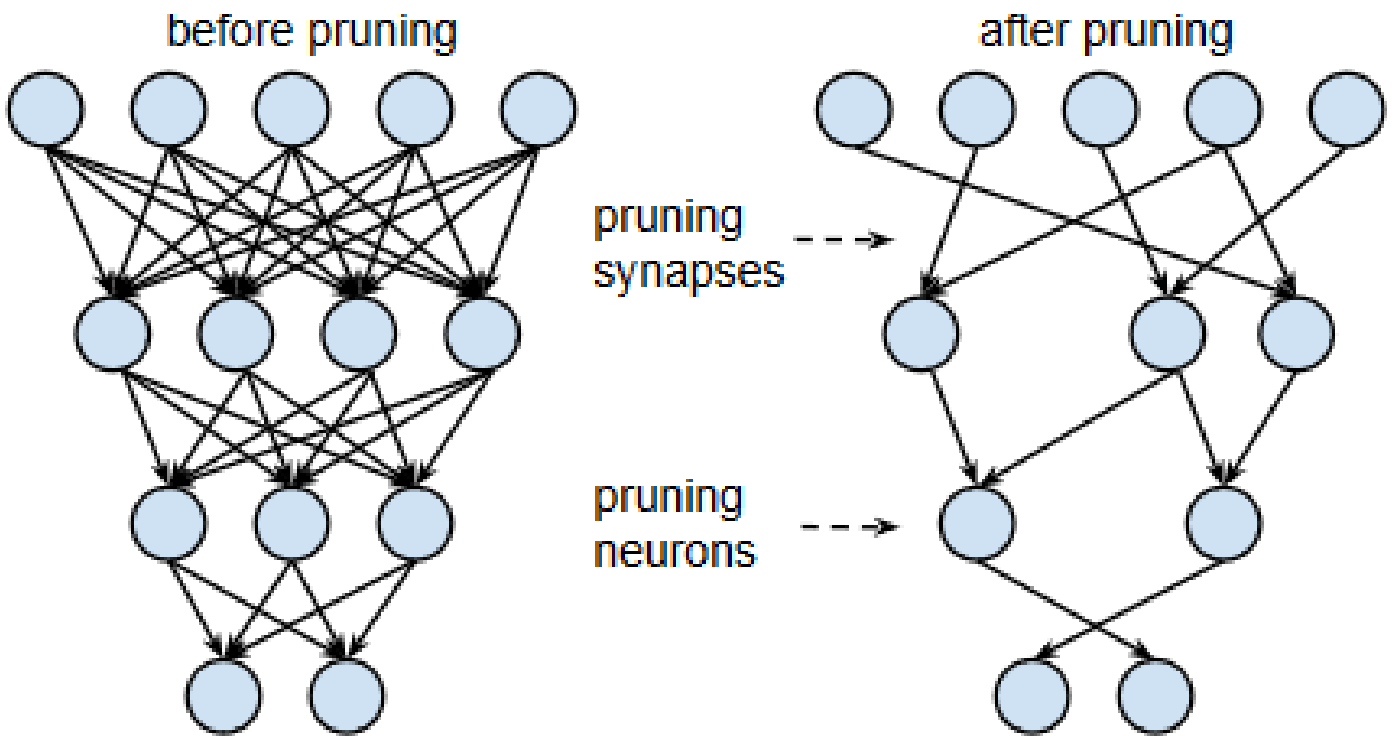


Illustration from Han et al., Learning both Weights and Connections for Efficient Neural Networks ((NIPS 2015)

Pruning

Techniques for making weights and activations sparse

- Sparse tensor = has ***many*** values equal to 0
- L0-"norm" = #zeros in the tensor

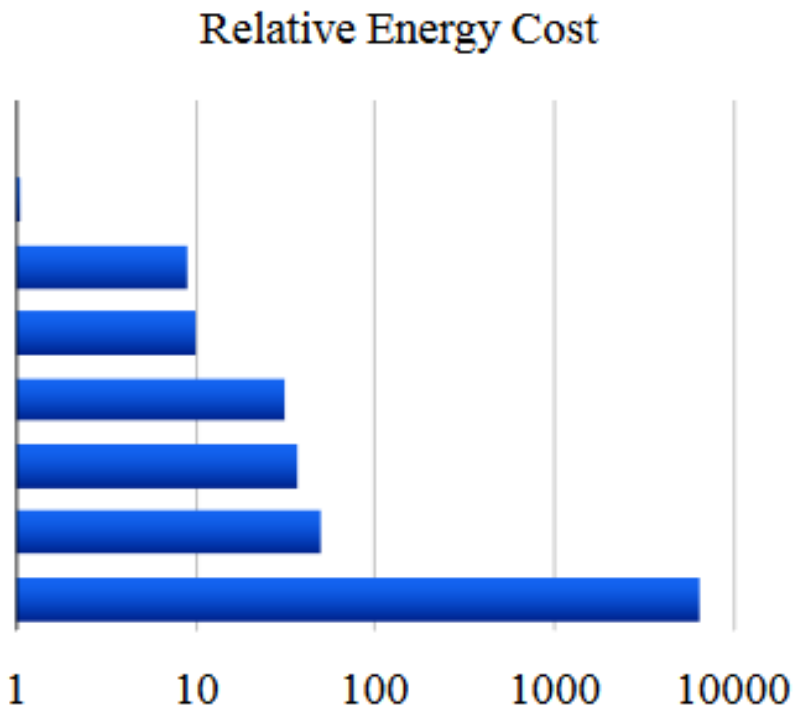
*Disambiguate from L0 norm, which is the maximum element of the tensor

Criterion for pruning elements

- Compare the absolute value with threshold, return zero if below
- For sparse tensors, we want to maximize L0-"norm"
- At the same time, keep end-to-end prediction accuracy high

Cost of Operations

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**



Mark Horowitz. Energy table for 45nm process, Stanford VLSI wiki
via Han et al., Learning both Weights and Connections for Efficient Neural Networks

Data Structures for Sparse Tensors

Sparse matrices representation contains only non-zero values:

- Coordinate list (COO): list of tuples (row, column, value)
- List of lists: (LIL) each row is a list, containing pair (column, value)
- Dictionary: (DOK) a mapping from (row, column) to value
- Compressed sparse row (CSR): three arrays, each for row, column and value.

Some properties:

- The operations are more expensive on real systems, and harder to rely on locality; operator implementations are more complex
- Need to ensure that the sparsity is sufficient to justify the costs, essentially trading communication for less data/communication

Pruning Intuition

The optimization space is very high-dimensional.

“Around” the original solution (dense network), there are likely solutions that are sparse.

The goal of our procedure is to find those sparse networks (sets up the stage for local search)

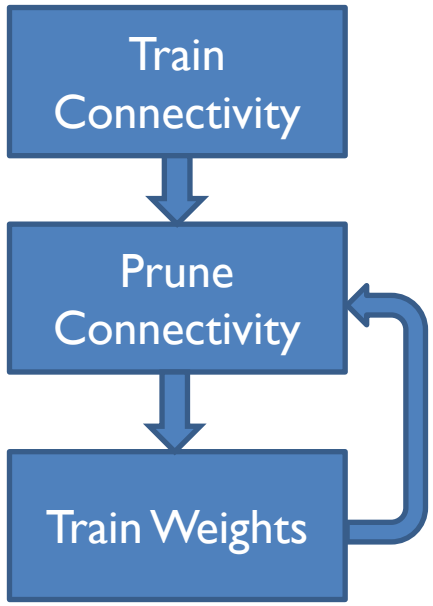
Pruning Schedules

Styles of pruning:

- One-shot: no retraining, 2x connections removed, *almost* no accuracy loss
- Iterative: retraining follows pruning; more sparsity, no accuracy loss

Algorithm for iterative "Deep Compression" (Han et al. NIPS'15):

- Conventional training produces ***dense network***
- Prune low-weight connections – zeros out weights with value below a specified threshold, to prudice a ***sparse network***
- Then, retrain the network to learn the new weights for the remaining connections.
  (improves the accuracy of the sparse network)



Pruning Schedules

Schedule of the iterative process: determining the changes of the pruning criteria, iteration count, how often specific tensors are pruned, stopping rule (e.g., a desired level of sparsity).

Pruning granularity:

- Element-wise: individual weights
- Structured pruning: pruning groups of elements (e.g., filters)

Granularity of Pruning

Unstructured (fine grained) neuron level

Intra-kernel pruning

Kernel pruning

Filter pruning

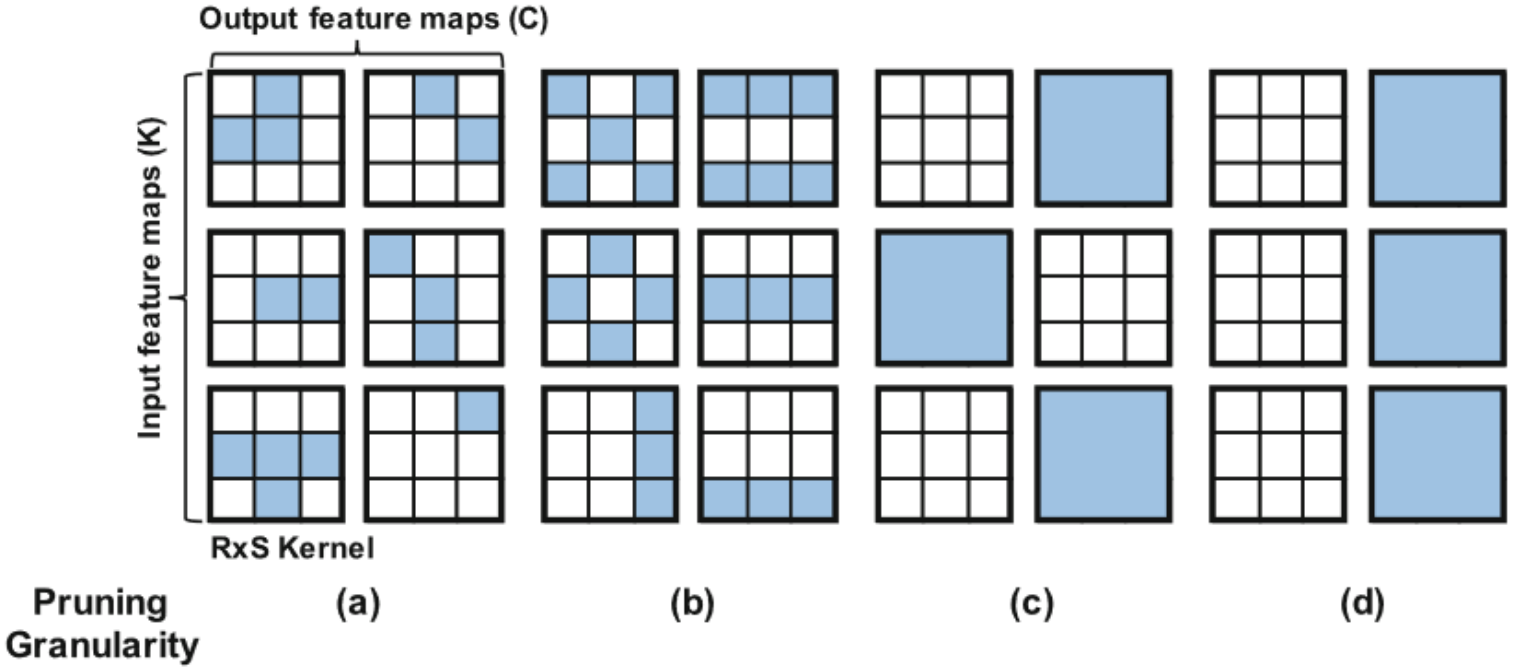


Fig. 15.1 Types of pruning granularity (figure modified from [34]). (**a**) Fine-grain. (**b**) Intra-kernel vector. (**c**) Kernel. (**d**) Filter

Pruning – Early Results

Han et al., Learning both Weights and Connections for Efficient Neural Networks (NIPS'15)

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12×**
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12×**
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9×**
VGG-16 Ref	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13×**

Small error, even better accuracy in some cases

Significant reduction. But, how does that affect performance or energy?

Han et al., Learning both Weights and Connections for Efficient Neural Networks (NIPS'15)

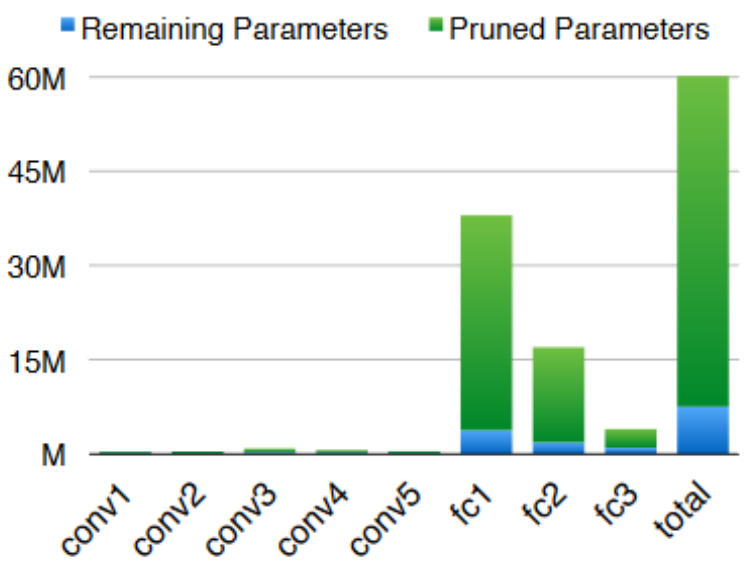
Pruning – Early Results

Conv = convolution layer
Fc = fully connected layer

Flop = Floating point operations
Act = Activations

Table 4: For AlexNet, pruning reduces the number of weights by 9× and computation by 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%**



Fully connected layers are much more amenable than convolutional

Does the reduction in operations correspond to the speedup on real-world hardware?

Pruning State-of-the-Art

Based on Survey "What is the State of Neural Network Pruning?" (SysML 2020)

- Many methods effectively compress networks > 10x with minimal impact on accuracy. Sometimes they have better accuracy than the dense models.
- For aggressive pruning, intelligent pruning methods (selecting parameters, or globally tuning) outperform random pruning.
- Iterative pruning often outperforms retraining with the same sparsity pattern (i.e., model is randomly initialized with all the weights identified by iterative pruning clamped to zero).
- Comparisons are hard to make systematically

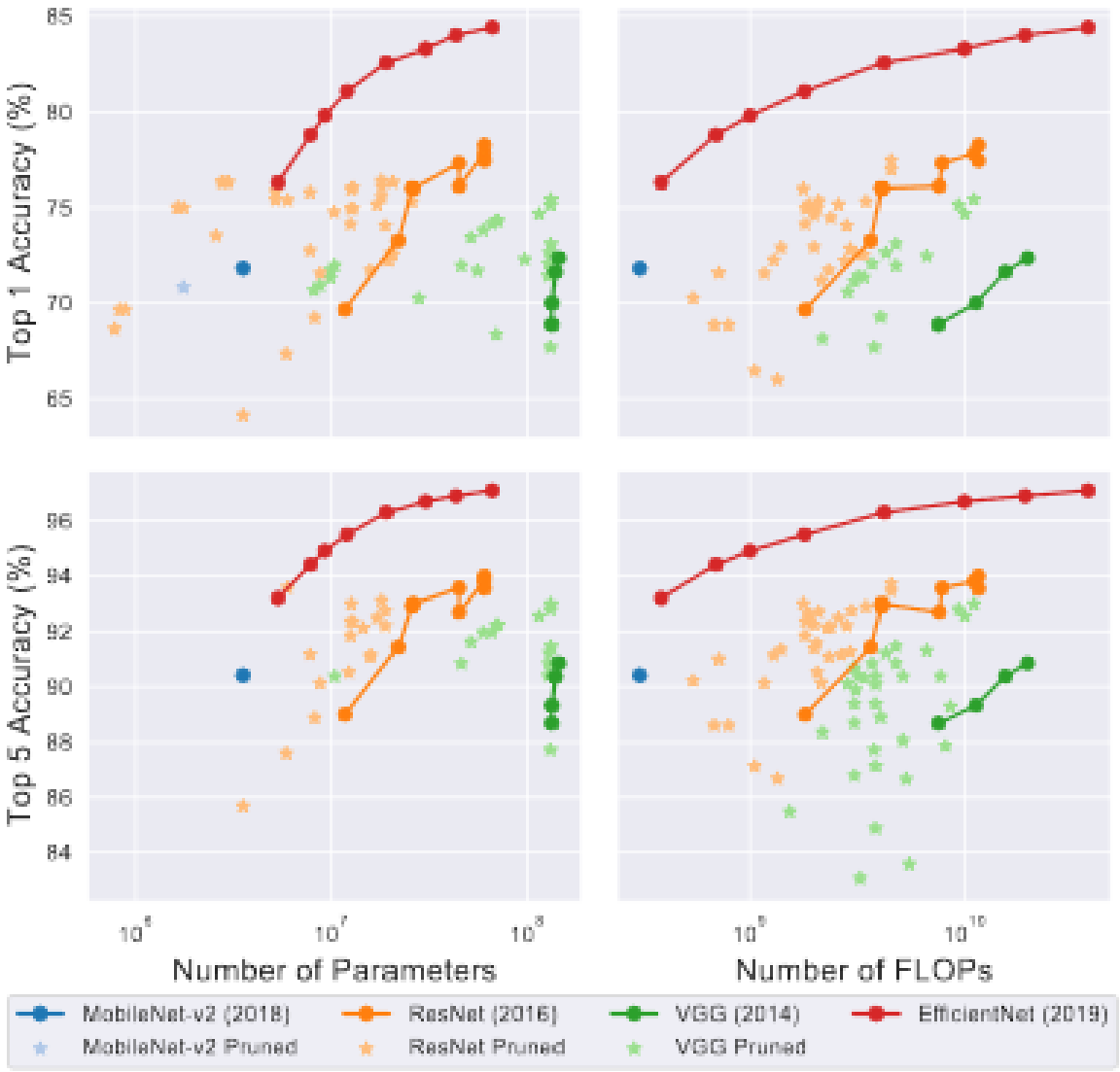
Ignoring Pre-2010s Methods There was already a rich body of work on neural network pruning by the mid 1990s (see, e.g., Reed's survey (Reed, 1993)), which has been almost completely ignored except for Lecun's Optimal Brain Damage (LeCun et al., 1990) and Hassibi's Optimal Brain Surgeon (Hassibi et al., 1993). Indeed, multiple authors have rediscovered existing methods or aspects thereof, with Han et al. (2015) reintroducing the magnitude-based pruning of Janowsky (1989), Lee et al. (2019b) reintroducing the saliency heuristic of Mozer & Smolensky (1989a), and He et al. (2018a) reintroducing the practice of "reviving" previously pruned weights described in Tresp et al. (1997).

Pruning State-of-the-Art

From What is the State of Neural Network Pruning?" (SysML 2020)

"Size and speed vs accuracy tradeoffs for different pruning methods and families of architectures.

Pruned models sometimes outperform the original architecture, but rarely outperform a better architecture."



Pruning State-of-the-Art

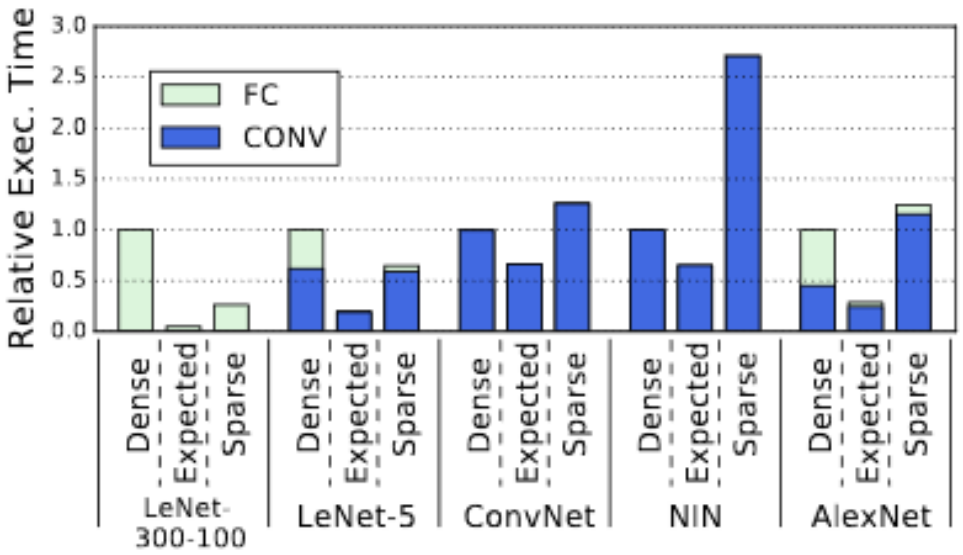
5.2 Metrics Ambiguity

It can also be difficult to know what the reported metrics mean. For example, many papers include a metric along the lines of "Pruned%". In some cases, this means fraction of the parameters or FLOPs remaining (Suau et al., 2018). In other cases, it means the fraction of parameters or FLOPs removed (Han et al., 2015; Lebedev & Lempitsky, 2016; Yao et al., 2018). There is also widespread misuse of the term "compression ratio," which the compression literature has long used to mean $\frac{\text{original size}}{\text{compressed size}}$ (Siedelmann et al., 2015; Zukowski et al., 2006; Zhao et al., 2015; Lindstrom, 2014; Ratanaworabhan et al., 2006; Blalock et al., 2018), but many pruning authors define (usually without making the formula explicit) as $1 - \frac{\text{compressed size}}{\text{original size}}$.

Reported "speedup" values present similar challenges. These values are sometimes wall time, sometimes original number of FLOPs divided by pruned number of FLOPs, sometimes a more complex formula relating these two quantities (Dong et al., 2017; He et al., 2018a), and sometimes never made clear. Even when reporting FLOPs, which is nominally a consistent metric, different authors measure it differently (e.g., (Molchanov et al., 2016) vs (Wang & Cheng, 2016)), though most often papers entirely omit their formula for computing FLOPs. We found up to a factor of four variation in the reported FLOPs of different papers for the same architecture and dataset, with (Yang et al., 2017) reporting 371 MFLOPs for AlexNet on ImageNet, (Choi et al., 2019) reporting 724 MFLOPs, and (Han et al., 2015) reporting 1500 MFLOPs.

From What is the State of Neural Network Pruning?" (SysML 2020)

Pruning (even with FLOPs reported) don't always result in better speedup:



Run time for the dense and the sparse DNN models with Deep Compression on the CPU; GPU results are similar

From Yu et al. Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism (ISCA 2017)

Quantization in Inference

Make the model representation compact:

FP32 → FP16

- Float to half-float, reduces the range

FP32 → INT8

- From 10^38 values to 256

FP32 → binary (+1, -1)

- Extreme, but works for e.g., MNIST, CIFAR-10

Cost of Operations

INT8 Operation	Energy Saving vs FP32	Area Saving vs FP32
Add	30x	116x
Multiply	18x	27x

William Dally. High-Performance Hardware for Machine Learning. Tutorial, NIPS, 2015

But also consider the cost of transferring data!

Quantization with Int8

Some examples:

- **Dynamic Fixed Point:** Adapts the range of fixed point to each network segment (offline). The challenge is the wide range of weights e.g., in AlexNet 97% weights are in $[2^{-11}, 2^{-3}]$, 99% layer outputs are in $[2^{-2}, 2^{8}]$. Activations are 2^{9}x larger than weights.
- **MiniFloat:** float with less than 32bit. Using 8 bit float (1-3-4 bits).
- **Multiplier-free Arithmetic:** bit-shifts instead of multiplications

$$z_i = \sum\left(x_j \cdot w_j\right) + b_i \approx \sum\left(x_j \ll \mathrm{round}\left(\log_2 w_j\right)\right) + b_i$$

  It first approximates weights by their closest power of 2 number, then does a sequence of shifts and accumulates

Ristetto* evaluates these three quantization schemes. We can apply these optimizations with or without fine tuning

*Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks (IEEE Transactions on Neural networks and Learning Systems Nov 2018)

Quantization with Int8

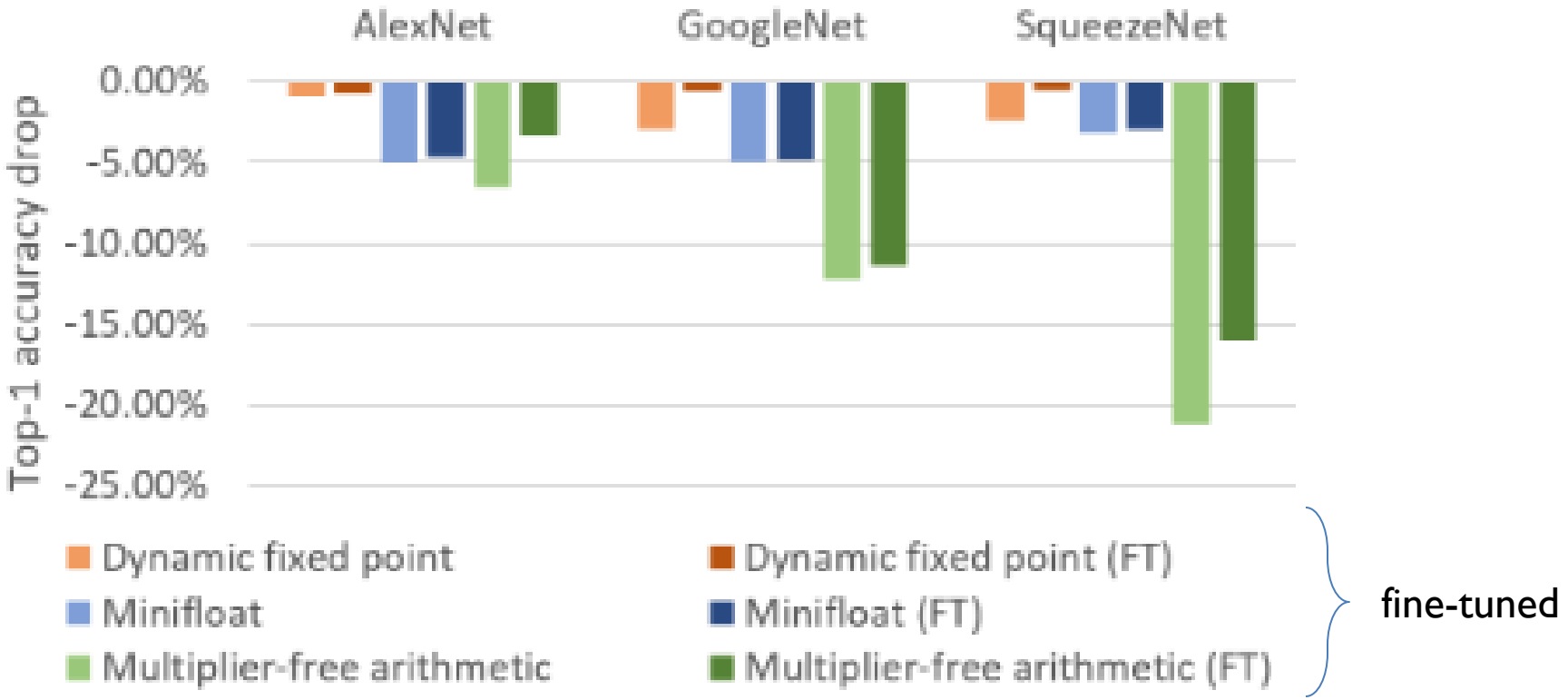


Fig. 4. Top-1 accuracy drop of 8-bit networks. We show the accuracy difference between the approximated and the 32-bit FP network. Fine-tuned networks are denoted with *FT*. Our dynamic fixed point networks achieve the best performance and are within 1% of the original network.

Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks (IEEE Transactions on Neural networks and Learning Systems Nov 2018)

Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding (NIPS 2016)

Combining Quantization and Pruning

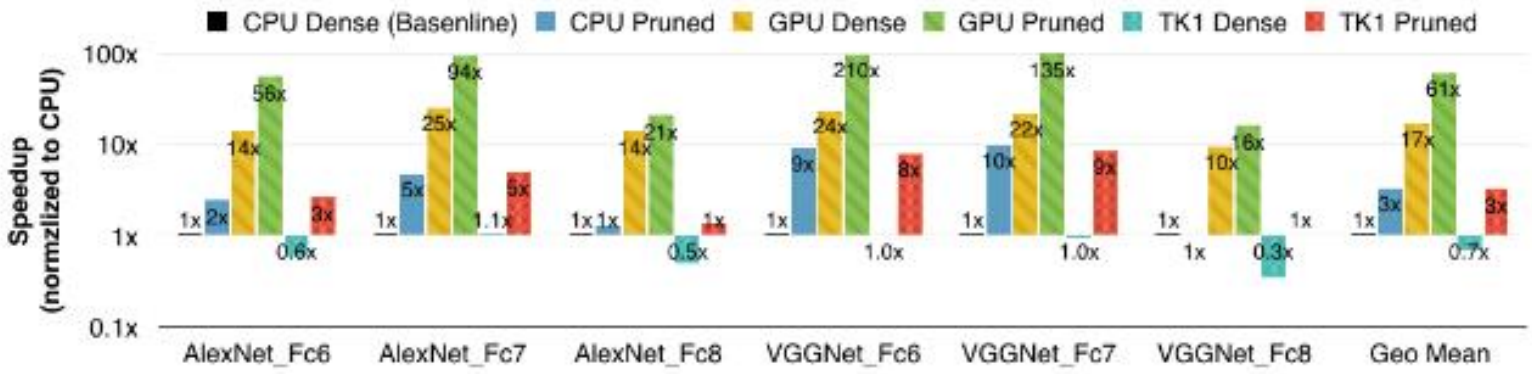


Figure 9: Compared with the original network, pruned network layer achieved 3× speedup on CPU, 3.5× on GPU and 4.2× on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

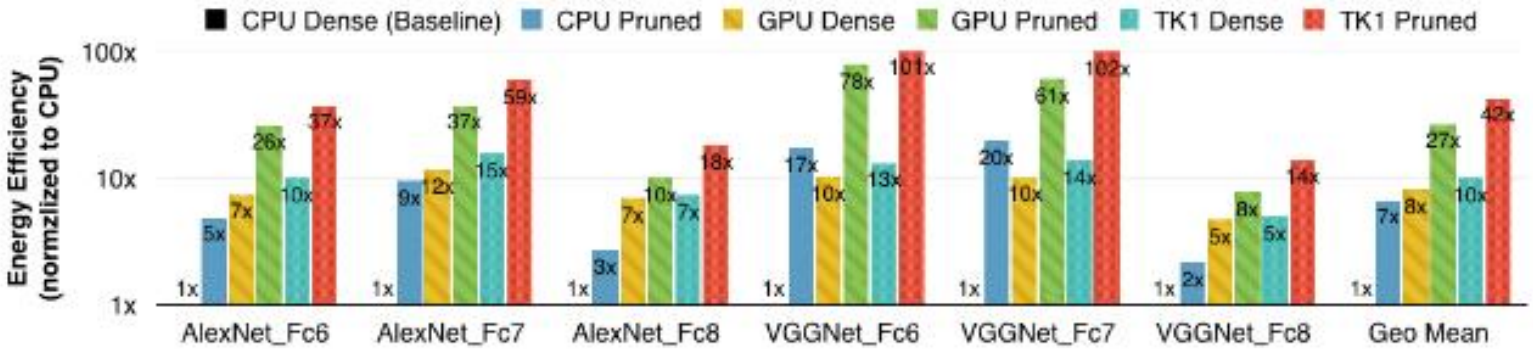


Figure 10: Compared with the original network, pruned network layer takes 7× less energy on CPU, 3.3× less on GPU and 4.2× less on mobile GPU on average. Batch size = 1 targeting real time processing. Energy number normalized to CPU.

Combining Quantization and Pruning

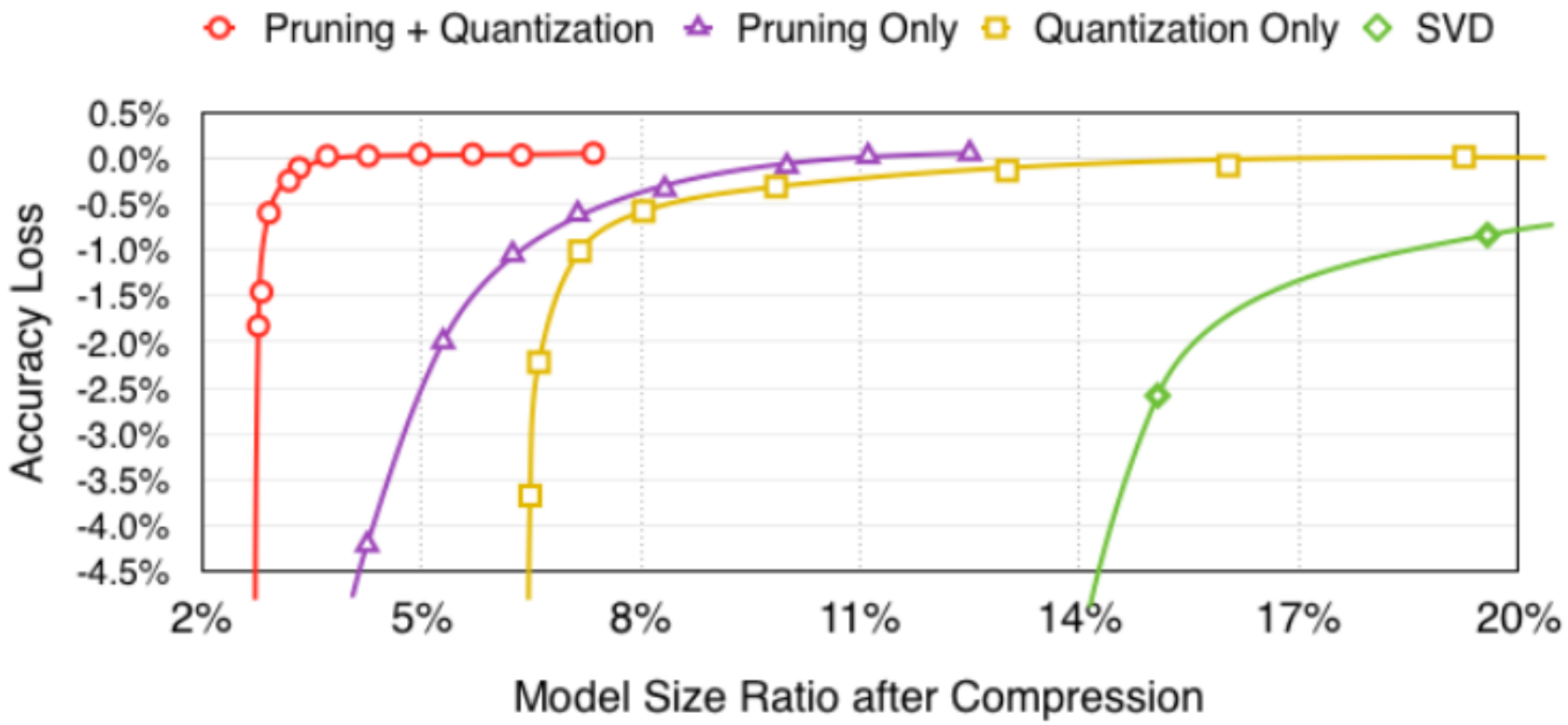


Figure 6: Accuracy v.s. compression rate under different compression methods. Pruning and quantization works best when combined.

Quantization During Training

Caveat with offline quantization: potential overflows in the computation may hinder the accuracy

Quantization may significantly impact some layers (e.g., first and last)

For smaller models (e.g., MobileNet), quantization after training may not preserve accuracy.

Quantization aware training:

- Maintain a full-precision weights to accumulate small changes in the gradients
- Quantize weights in each step to estimate the accuracy loss in every step
- Effect of quantization is backpropagated using straingth-through estimators (passes the gradient through the functions unchanged)
- At the end keep only the quantized weights

Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1 (NIPS 2016) Bengio et al

Binarized Neural Networks

All weights and activations are binary -1 or +1

Transform real valued variables to binary:

- Deterministic $x^b = Sign(x)$
- Probabilistic $x^b = \begin{cases} +1 \text{ with probability } p = clip_{[0,1]}\left(\frac{1}{2}(x+1)\right) \\ -1 \text{ with probability } 1-p \end{cases}$

Probabilistic may give better results, but harder to implement

Training still operates on floating point, the gradients need to be floats. SGD explores the parameter space in small noisy steps

The main challenge is propagating the gradients through discretized network – straight through estimator (lets the gradient unmodified)

Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1
(NIPS 2016) Bengio et al

Figure 1. Training curves of a ConvNet on CIFAR-10 depending on the method. The dotted lines represent the training costs (square hinge losses) and the continuous lines the corresponding validation error rates. Although BNNs are slower to train, they are nearly as accurate as 32-bit float DNNs.

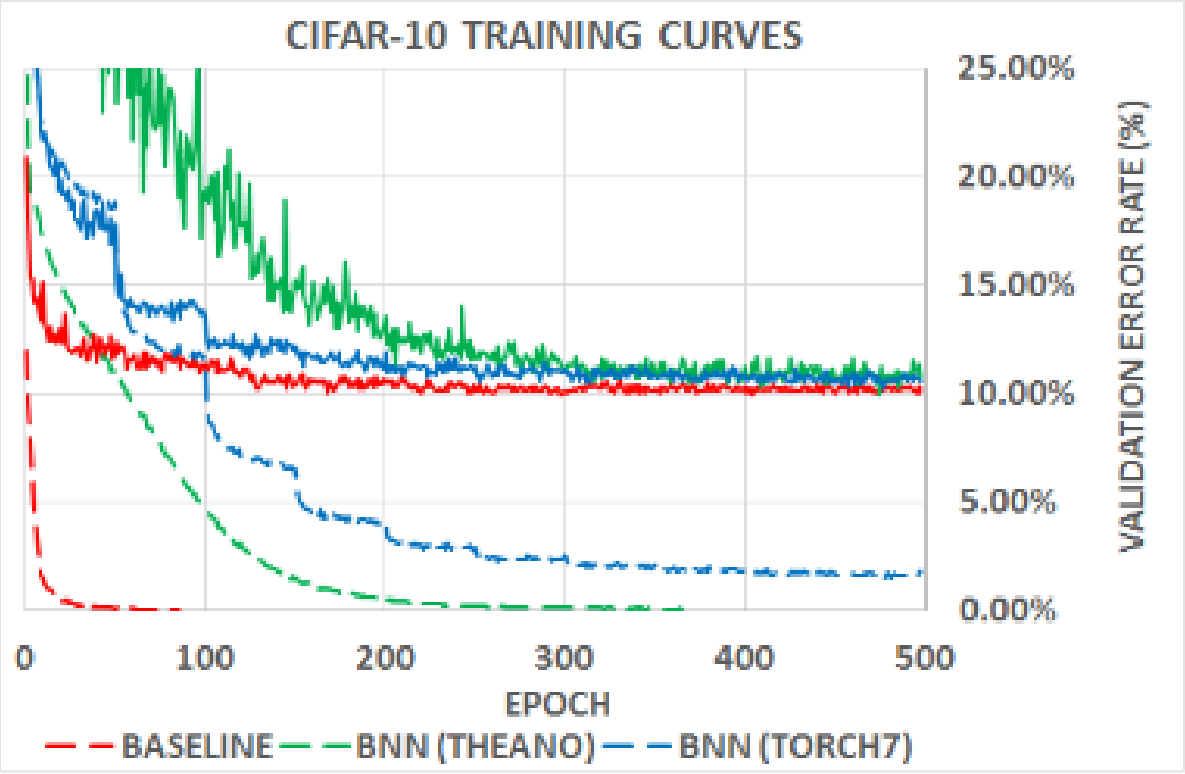
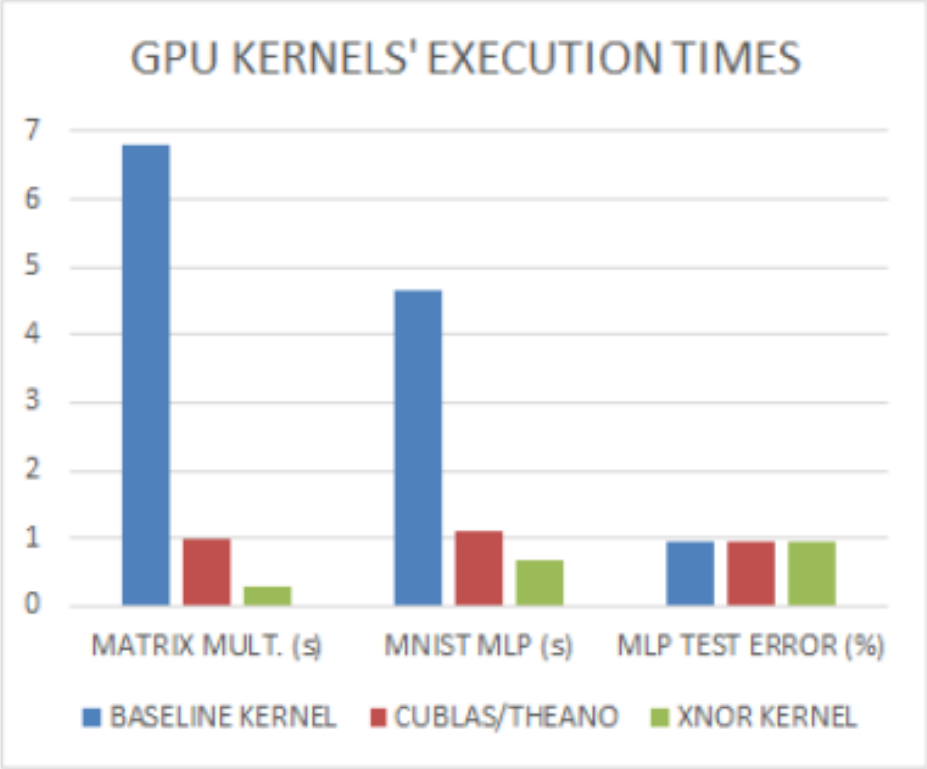


Figure 3. The first three columns represent the time it takes to perform a $8192 \times 8192 \times 8192$ (binary) matrix multiplication on a GTX750 Nvidia GPU, depending on which kernel is used. We can see that our XNOR kernel is 23 times faster than our baseline kernel and 3.4 times faster than cuBLAS. The next three columns represent the time it takes to run the MLP from Section 2 on the full MNIST test set. As MNIST's images are not binary, the first layer's computations are always performed by the baseline kernel. The last three columns show that the MLP accuracy does not depend on which kernel is used.



Neural Network Training

Common wisdom: Training high accuracy and inference low precision

- Stochastic gradient descent more sensitive to quantization
- Postprocess highly accurate (FP32) network

Stochastic gradient descent has a nice property for approximation:

- We can parallelize it without locks.
- While updating, it can read/write stale data, so some updates are lost
- The algorithm is iterative, a few more iterations (epochs) are much more efficient than a full-locking scheme

Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent(NIPS'11)

Check out Kailash Gopalakrishnan's talk:
https://www.youtube.com/watch?v=sYzqCT7KUHQ