Lecture 7: Neural Network Pruning

Puneet Gupta

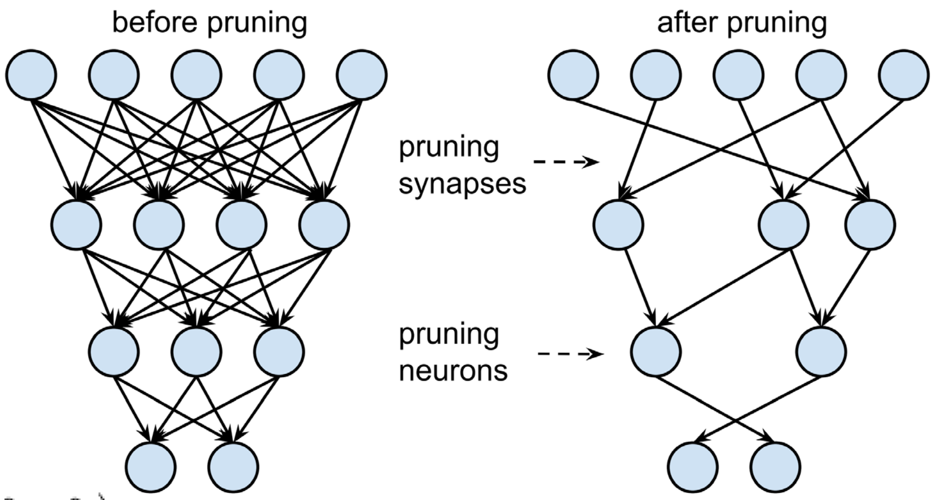
Logistics

- No lecture next class
  - No presentations
- TA will schedule a project office hour at that time over zoom. Link upcoming.

Pruning

- Basic idea: learn which connections are important; delete the rest (i.e., snap weights to 0)
  - Makes the network sparse
- How does pruning save storage ?
  - Don’t really store a 0
  - May be use specialized sparse storage formats
- How does pruning save compute ?
  - Less “real” operations (don’t need to do multiply or add by 0)
  - Reality: very hard to engineer efficient sparse processing architectures
- Common approach: magnitude-based pruning (i.e., prune weights with lowest absolute value)

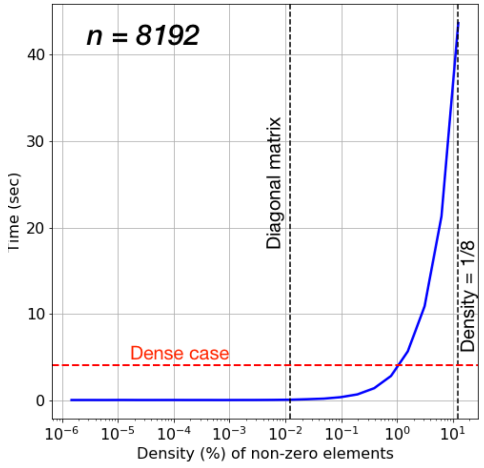
Pruning - Unstructured



- Any weights in a layer can be pruned
- High compression ratio
- Hard to implement
- Two main issues:
  - How to store ?
    - If you store the 0, no savings!
    - E.g., CSR (Compressed Sparse Row) format
    - V: non-zero values
    - COL_INDEX: column in which the corresponding non-zero value occurs
    - ROW_INDEX: #non-zeroes before row *i* . Last entry is total number of non-zeroes in the matrix
    - Need somewhat high sparsity to actually save memory!
    - What if the weights are quantized ?
  - How to compute ?
    - SpGEMM usually requires very high degree of sparsity (98%+) to give speedup.
    - Sparse workloads become memory bound!

$$\begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

```
V = [ 10 20 30 40 50 60 70 80 ]
COL_INDEX = [ 0 1 1 3 2 3 4 5 ]
ROW_INDEX = [ 0 2 4 7 8 ]
```



[Sparse Matrices in Pytorch. This article will analyze runtimes of… | by Sourya Dey | Towards Data Science]

Extreme Case: Binarization & Sparsity

- Binarization benefits:
  - Up to 32x smaller storage.
  - Bitwise XNOR multiplication.
  - Popcount accumulation.
- Naïve (unstructured) pruning:
  - No way of representing a 0.
  - No guarantee of processor word alignment.
  - Cannot easily utilize bitwise XNOR multiplication.
  - Weights stored individually.

A	B	AxB
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

=

A	B	~(A ^ B)
0	0	1
0	1	0
1	0	0
1	1	1

=

1.5 GOPS,
32-lane
vector unit

1	-1	0	0	0	1	0	0

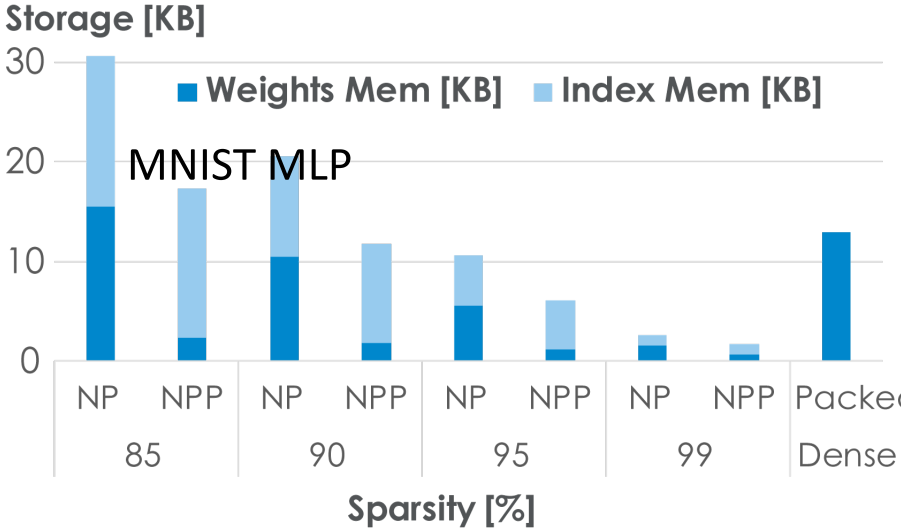
→

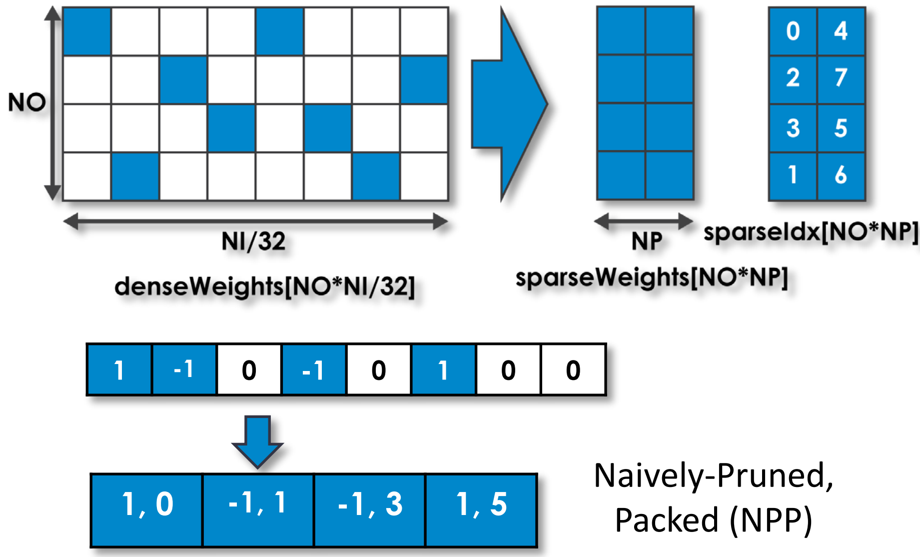
1	-1	1	?

Naively-Pruned (NP)

Binarization & Sparsity

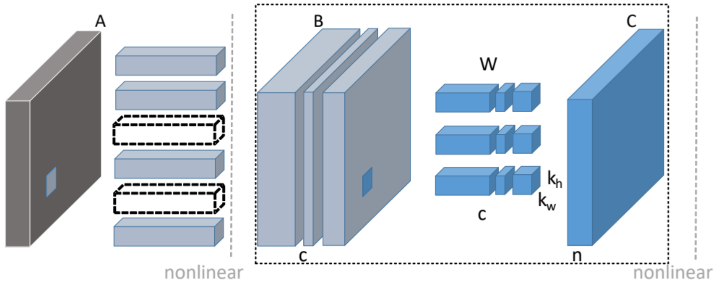
- Enforce number of non-zero weights to match processor word size.
- Pack weights into vectors/words.
- Naively-Pruned, Packed (NPP)
- Need 95%+ sparsity to get storage benefit even with a little cleverness
- Inputs (the dense operand) still needs to be read one by one and packed into a word to leverage bitwise XNOR.
  - Runs 15X *SLOWER* than the unpruned XNOR network!
  - One possible solution: nanocad.ee.ucla.edu/wp-content/papercite-data/pdf/j63.pdf
- Underlying problem: utilizing full SIMD width when sparse
  - Bigger issue if SIMD width is large
  - E.g., a large hardware MAC..

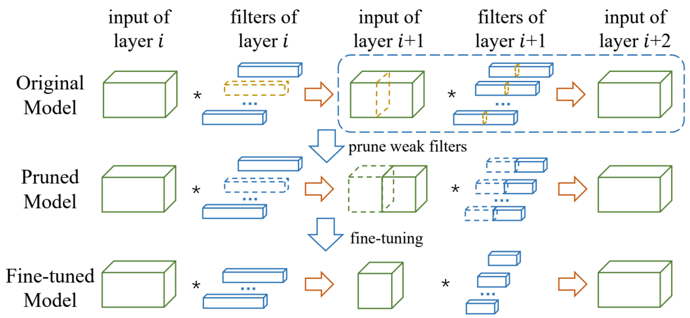


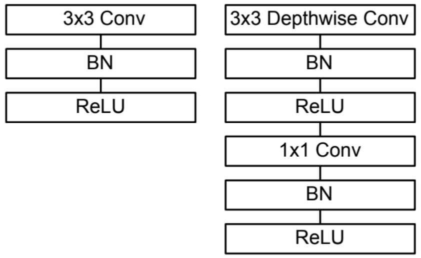


Structured Pruning

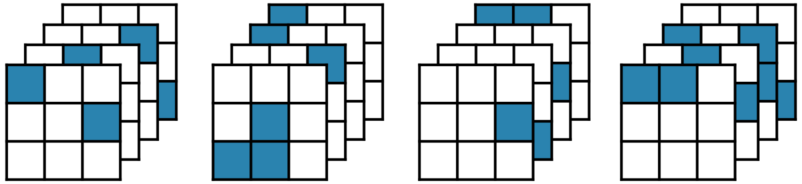
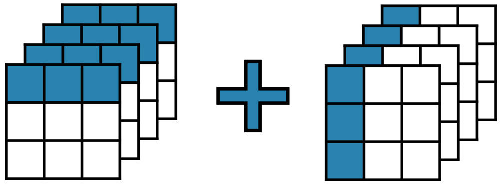
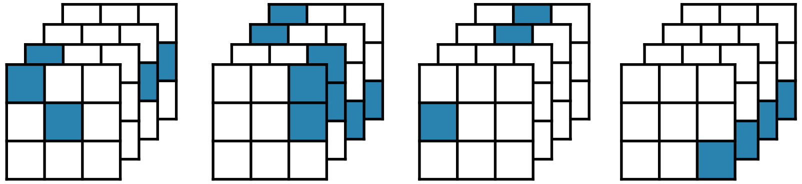
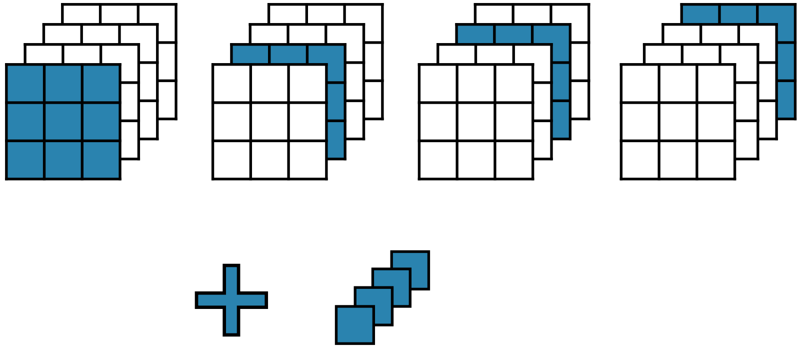
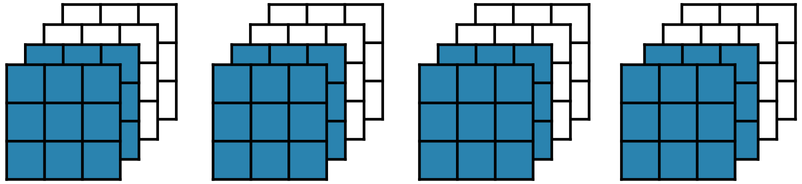
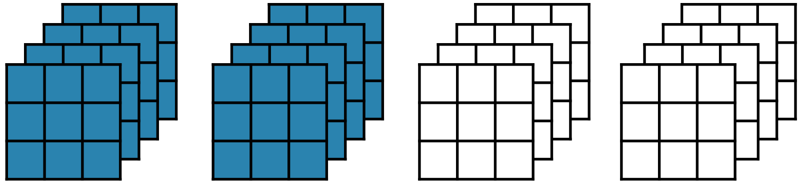
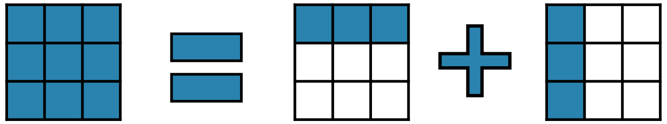
- Impose some structure (sometimes optimized with hardware in mind) in what edges get deleted
- Channel pruning
  - Filters in the previous layer corresponding to the pruned channels also removed
- Filter pruning
  - Prune entire filters
- Factorization-based pruning
  - Essentially different NN architecture
  - E.g., Factor nxn convolutions into 1xn and nx1
  - E.g., MobileNet







Pruning – Summary

Type	Implementation	Type	Implementation
Unstructured		2d factorize	
Block		2+1d factorize	
Channel			
Filter		Matrix factorize	

Pruning Example Flow

- Iterative Pruning + Retraining
  1. Choose a neural network architecture
  2. Train the network until a reasonable solution is obtained.
  3. Prune the weights of which magnitudes are less than a threshold τ.
  4. Train the network until a reasonable solution is obtained.
  5. Iterate to step 3.