Edge Deployment - Model Compression

Srinivas Rana, PhD

Talk at IIIT-B
15^{th} Oct 2024

Wadhwani AI

Motivation

- Complexity of the problem being solved → complexity of the neural network model

- Model complexity
  - Models can also be memory intensive (10s to 100s of million parameters and more) - memory footprint
  - Powerful infrastructure required to perform inference in a reasonably acceptable amount of time - computational footprint

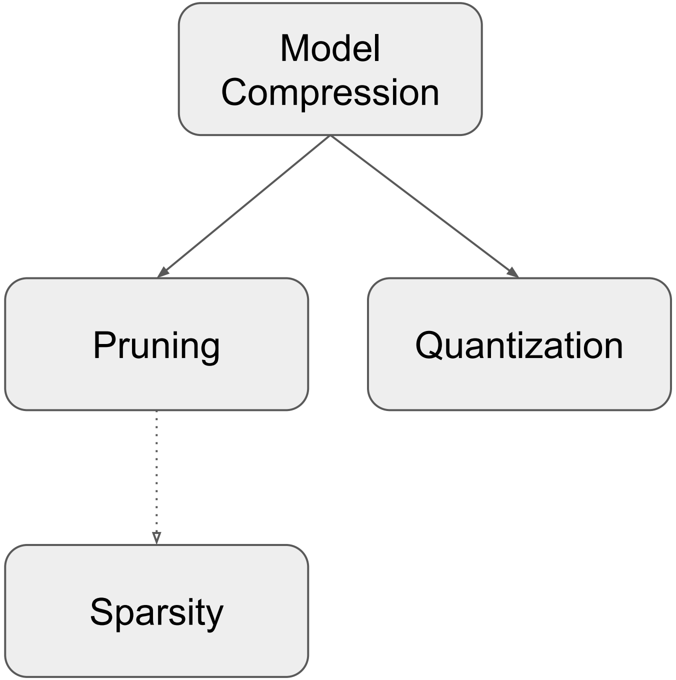
Model	Parameters (millions)	gFLOPs (forward)	FLOPs per parameter	Size (MB)
AlexNet	61	1.4	23	233
ResNet-50	26	8.2	315	98
ResNet-152	60	24	400	230
GNMT	244	24.5	100	933
Transformer	213	13.56	64	813
BERT	340	366.5	1100	1300
GPT-2	1500	3507	2254	5934
3D U-Net	17.3	657	38000	66

Motivation

Can we reduce these two footprints, i.e. reduce the model size and the number of computations, whilst retaining the model accuracy??

This strictly relates to inference alone and not training

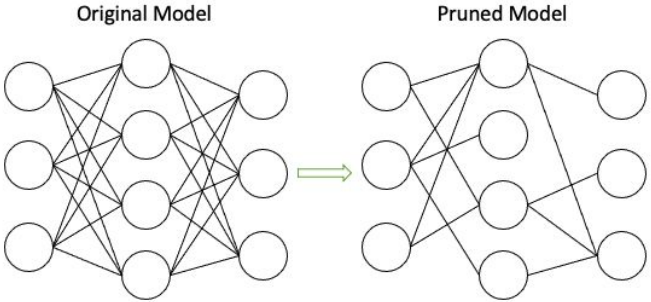
Terminologies



Pruning

Pruning

- DNNs are over-parameterized[1]
  - Many parameters encode the same / similar information in the network



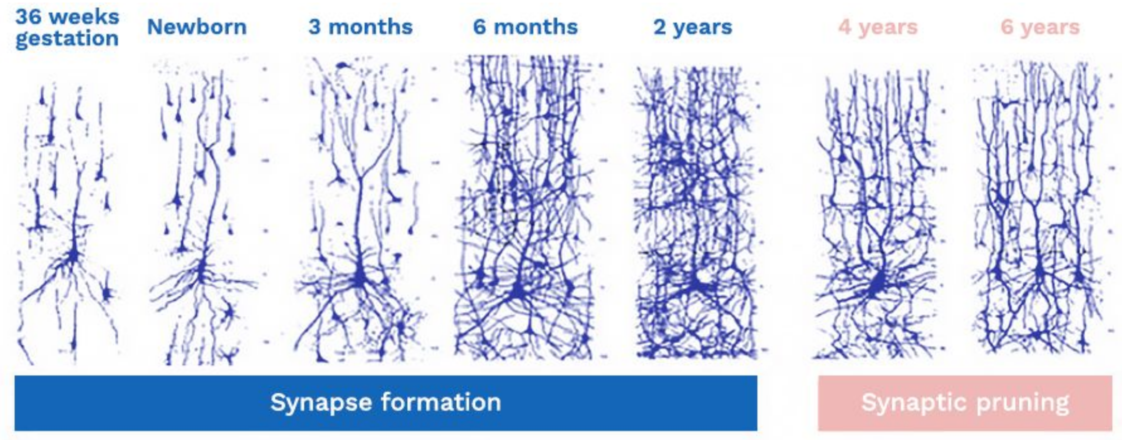
- Pruning eliminates these additional weights that do not contribute / add additional value to model performance

- Sparse tensors leading to fewer FLOPs → faster inference, smaller model
- Inherently regularised and hence more robust to noise
- Model drops in accuracy but can be recovered by finetuning

[1] Blier, Léonard & Ollivier, Yann. (2018). Do Deep Learning Models Have Too Many Parameters? An Information Theory Viewpoint.

Pruning

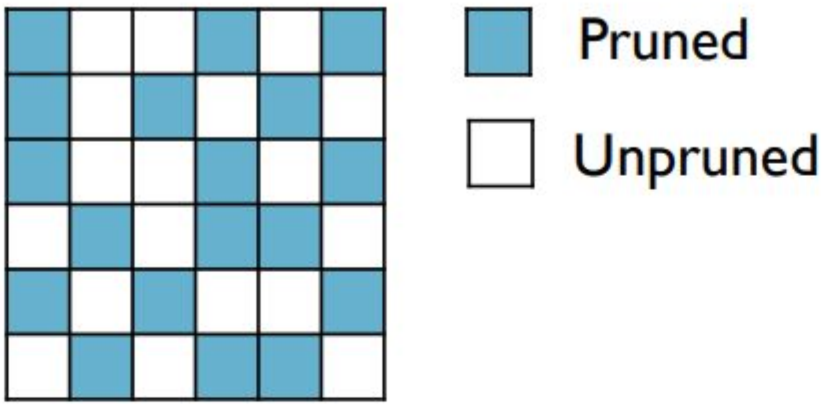
- Pruning is seen in nature too!



The Lancet Advancing Early Childhood Development: From Science to Scale

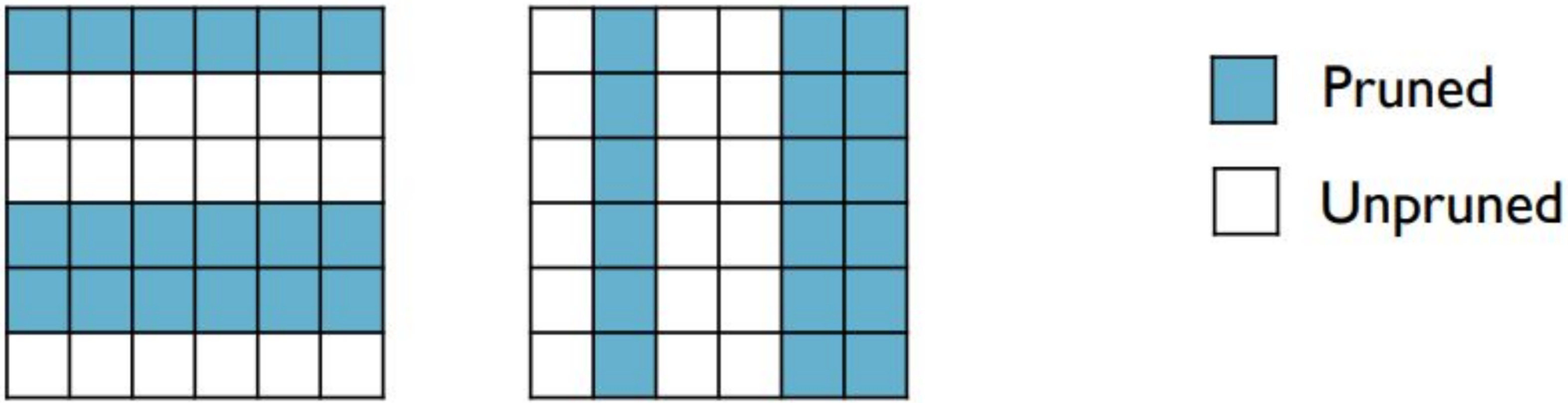
Pruning Strategies - Unstructured

- Pick out x % of the lowest magnitude weights and remove them
- This is known as unstructured pruning



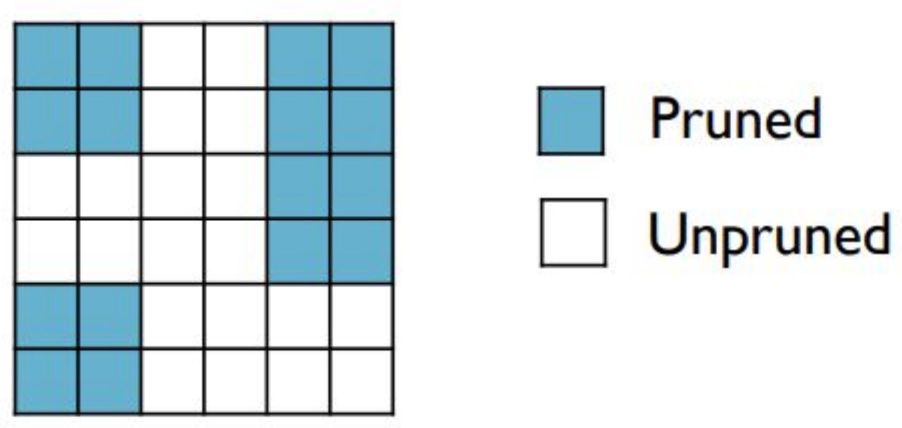
Pruning Strategies - Structured

- Pick out the lowest x % L1-norm for each row / column and remove them
- This is known as structured pruning



Pruning Strategies - Block

- Pick out x % of the lowest L1-norm in 2D blocks and remove the entire block
- This is known as block pruning

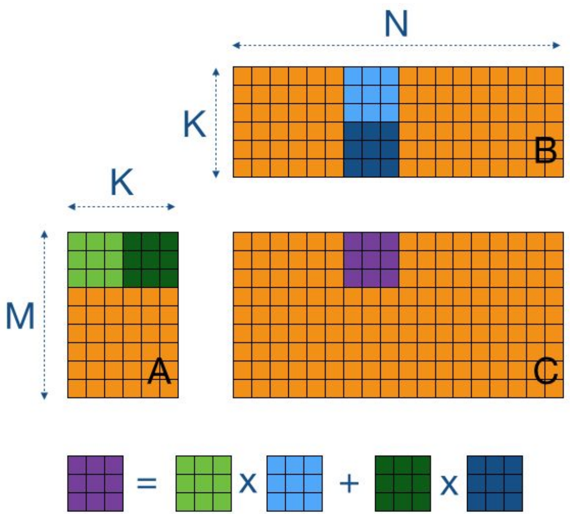


https://openai.com/blog/block-sparse-gpu-kernels/

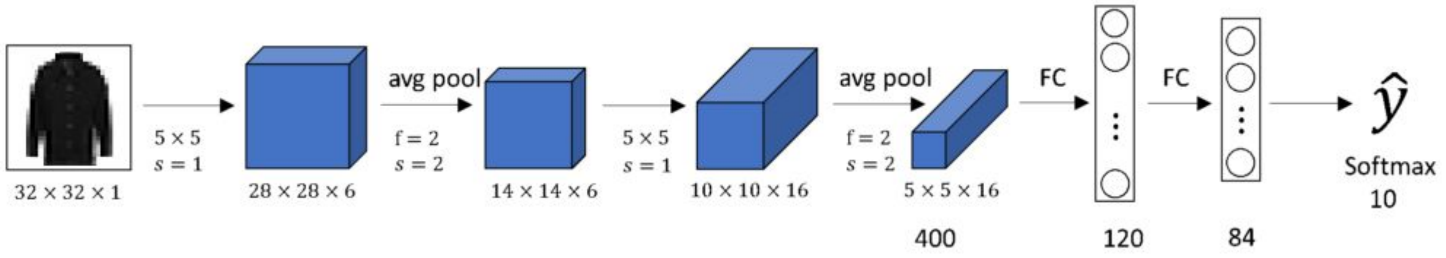
Why structured

- Limited support for sparse matrices in deep learning frameworks which hinder deployment of unstructured sparse matrices
  - May slow down inference which can affect real time applications demanding high throughputs

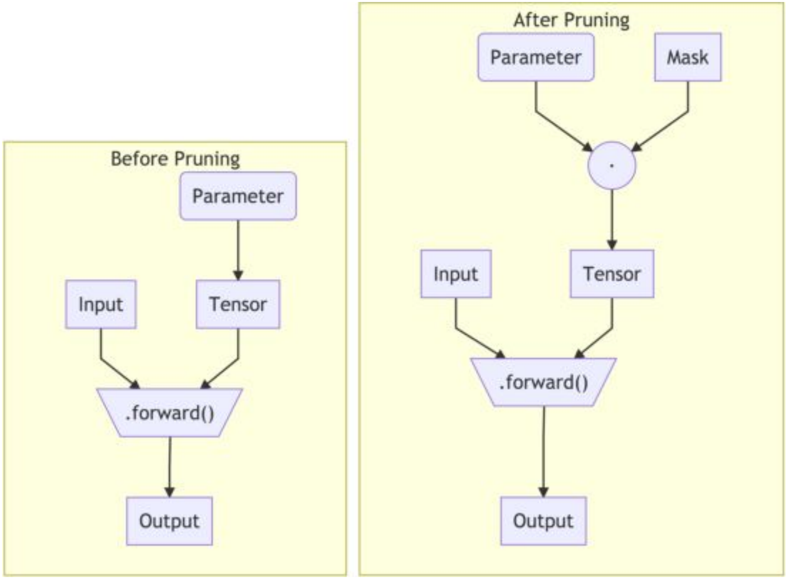
- Tiled matrix multiplications in GPU Tensor Cores
  - Block based methods are suited to take advantage of this architecture



Let's Prune!



32 × 32 × 1
5 × 5
s = 1
28 × 28 × 6
avg pool
f = 2
s = 2
14 × 14 × 6
5 × 5
s = 1
10 × 10 × 16
avg pool
f = 2
s = 2
5 × 5 × 16
400
FC
120
FC
84
ŷ
Softmax
10



Before Pruning
Parameter
Input
Tensor
.forward()
Output
After Pruning
Parameter
Mask
·
Input
Tensor
.forward()
Output

Pruning Recipe

- Load pretrained model
- Choose which parameters to prune and choice of pruning method
- Prune to required percentage
- Fine tune model to regain any lost accuracy
- Plot a *sparsity vs accuracy* curve to visualise and determine the sweet spot!
- Can an equivalent smaller dense model give similar performance?

Other Pruning Techniques

- Iterative Pruning
  - Prune, train, and repeat for *n* iterations
  - Usually shown to beneficial for higher sparsity levels

- Non magnitude techniques
  - Adaptive pruning - Monitor gradients and prune the weights corresponding to the gradients which move away more
  - Low rank matrix factorization

- Pruning from scratch
  - Open research problem
  - No single right idea on what sort of pruning mask to start from
  - Imposing pruning on pretrained models underperforms anyway likely because the model is stuck in a local minimum, and the optimizer settings being used aren't the best to get out of it

Quantization

Quantization

- Perform computations and store tensors using lower precision data types instead of the conventional FP32 precision
  - Typically INT8 but lower widths are within scope too

- FP32 → INT8
  - 4x reduction in size

Quantization

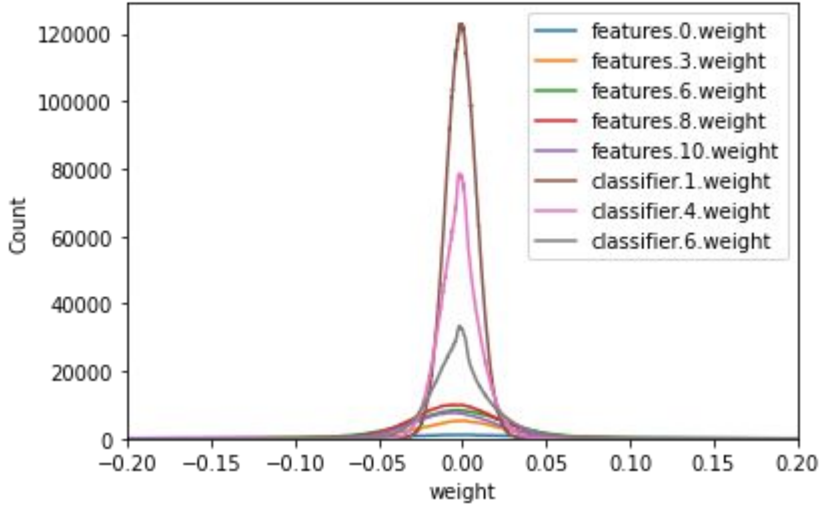
- Casting directly from higher precision to lower precision can lead to very large errors

Data Type	Value	% Deviation	Memory (bits)
FP64	3.141592653589793	-	64
FP32	3.141592653	5.97e-9	32
FP16	3.1415	9.39e-4	16
INT8	3	4.5	8

- 2^n values → *n* bits but in a ML context can lead to a lot of quantization noise
    - FP32 → INT8 : 6.8e38 → 256

Quantization

- Fortunately neural network model weights usually have a very small range close to 0



Alex Net Weight Distribution

Quantization

$$zero\ point = \frac{f_{max} + f_{min}}{2} \qquad scale\ factor = \frac{f_{max} - f_{min}}{q_{max} - q_{min}}$$

$$Q = \frac{F}{scale\ factor} + zero\ point$$

Quantization Strategies

- Dynamic Quantization
  - Weights quantized, activations in FP and quantized at compute time

- Static Quantization
  - Weights quantized, activations quantized, calibration post training

- Quantization Aware Training (QAT)
  - Weights and activations quantized, quantization numerics modelled while training

Recipe

Wadhwani AI

How to Proceed

- Examine your model architecture
  - Which layers need compression - bar charts are very helpful here!
  - Sanity check: Model file size == state_dict size
  - Use the target size to determine which layers need compression
- Pruning
  - L1, L2, structured, unstructured, … - sparsity vs accuracy curves to determine your accuracy ceilings
  - Own masks
  - Can smaller dense models with a similar number of parameters give you similar performance?
- Quantization
  - Dynamic, static
  - QAT

Suggested Reading

- Pruning and Quantization for Deep Neural Network Acceleration: A Survey
- PyTorch Pruning
- OpenAI Block Sparsity
- GPU Architecture
- Tiled Matrix Multiplication
- Matrix Multiplication NVIDIA
- PyTorch Quantization
- Dynamic Quantization
- Static Quantization and QAT
- LLM Pruning
- MobileLLM
- The Era of 1-bit LLMs

Thank you

Wadhwani AI is a program of the AI Unit of National Entrepreneurship Network (NEN).

WADHWANI AI | www.wadhwaniai.org