Efficient models: Sparsification

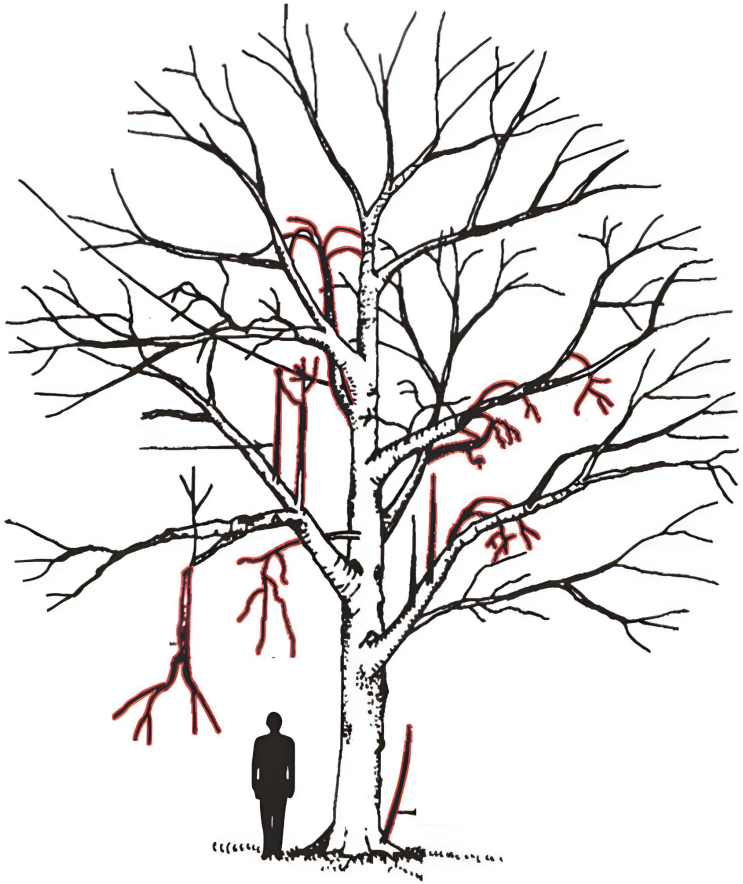
Egor Shvetsov

tg: @dalime e-mail: e.shvetsov@skoltech.ru

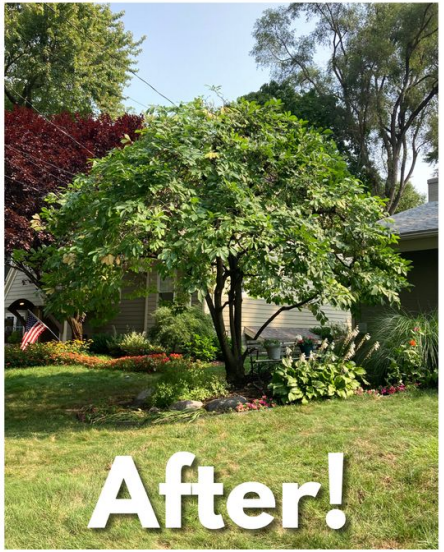
Sparsification

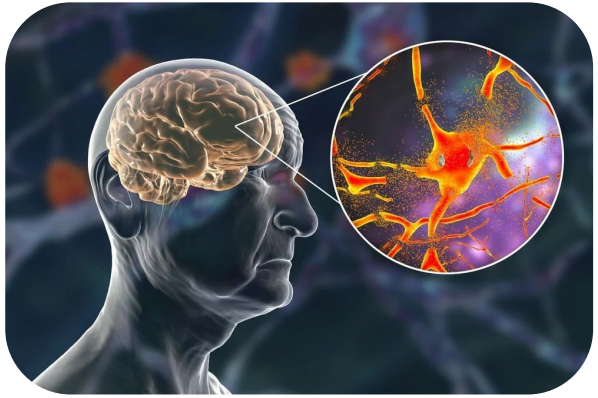
- Motivation
- Hardware view & Magnitude Sparsification
- When to Prune
- OBD, OSB, SparseGPT, Wanda
- Dynamic Sparsification
- PEFT
- Weight Initialization

>>> Motivation



Trees often carry large numbers of dead branches which produce neither leaves nor fruit and yet cause the tree to be unnecessarily heavy. These dead branches can be safely cut without affecting lead branches, which in most cases benefit from the extra room.





Prune or remove neurons we do not use ... real life optimization

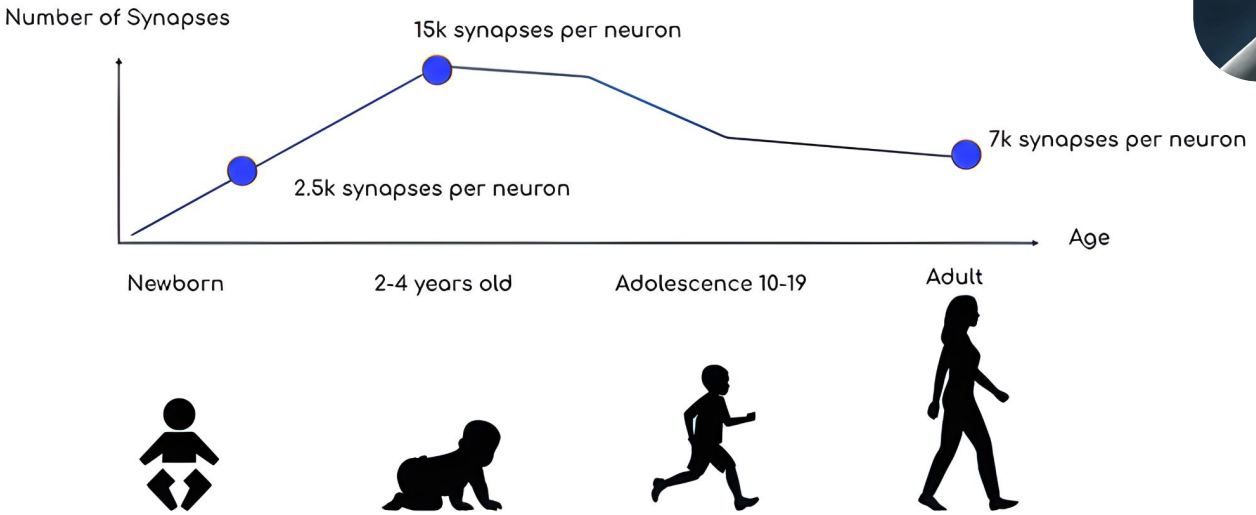


Image Source

Pruning or Sparsification

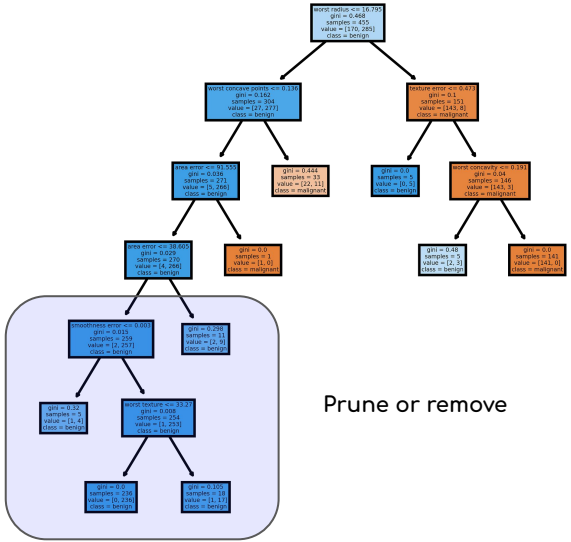
Pruning leads to sparsity

It is not the same

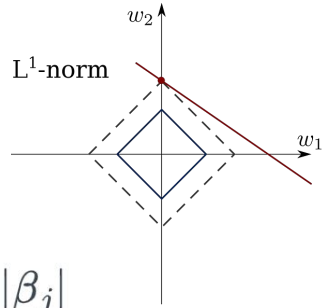
but it is used interchangeably

The idea is not new

Decision Trees



Prune or remove



Lasso regression

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Robust Principal Component Analysis - RPCA

$$W = S + L$$

S - is sparse

L - low rank

Why does it work & why can't we train a smaller model from the beginning?

Models are overparameterized
But why?

Initialization

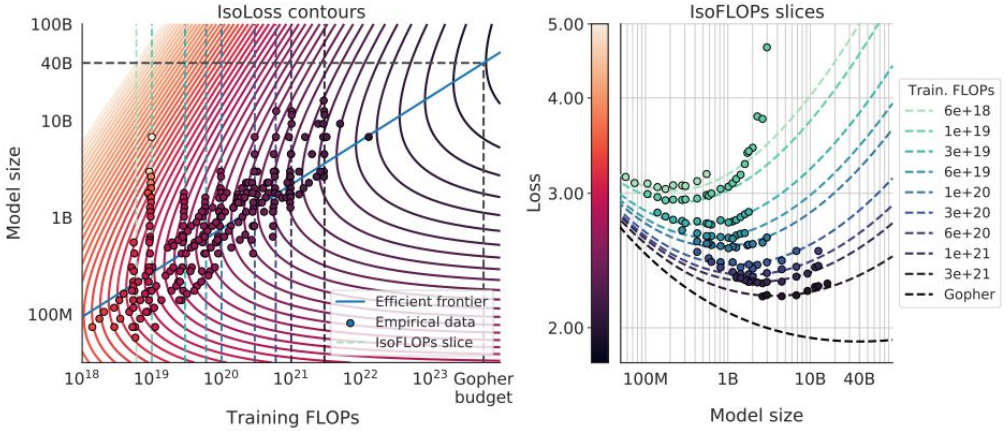
Training (optimization) dynamic

Data size and its quality

Use less compute to gain better results.

What is a relation between:

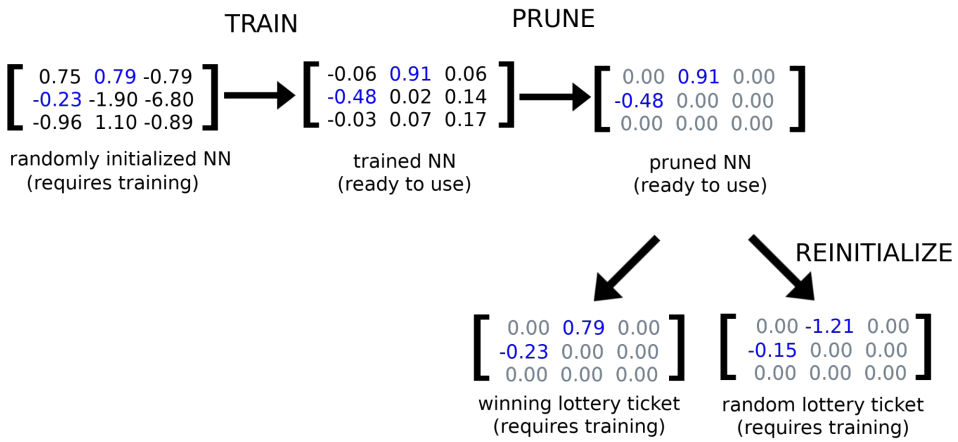
Model size / Data



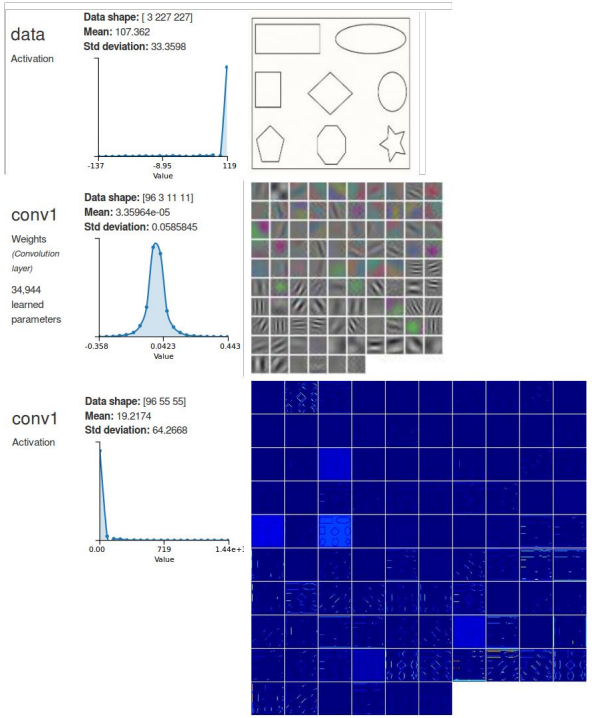
Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

Training Compute-Optimal Large Language Models

The paper hypothesis is that within a big model there is a smaller model that, if trained on its own, would match the performance of the larger one.

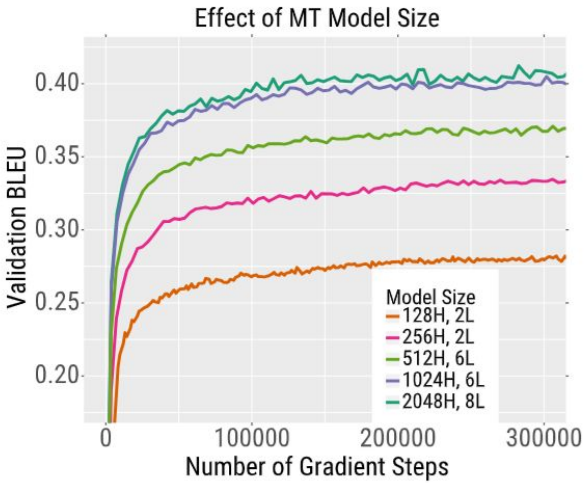


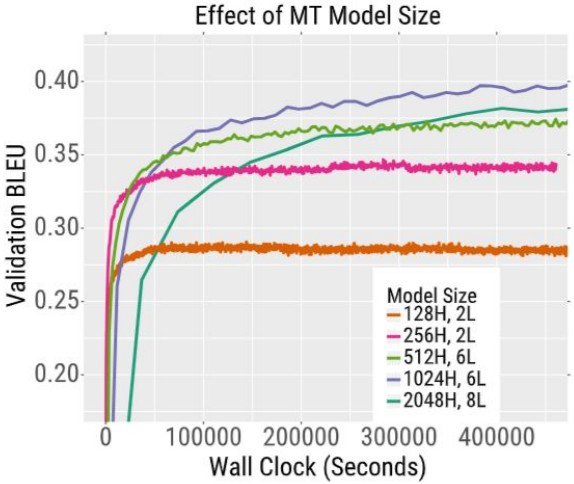
[The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks]



Source

Train bigger models and then compress!





Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers

Train bigger models and then compress!

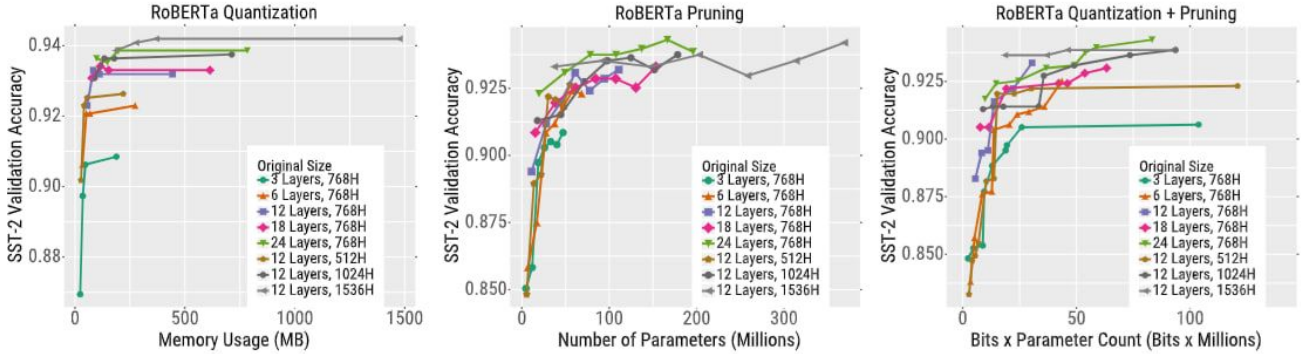
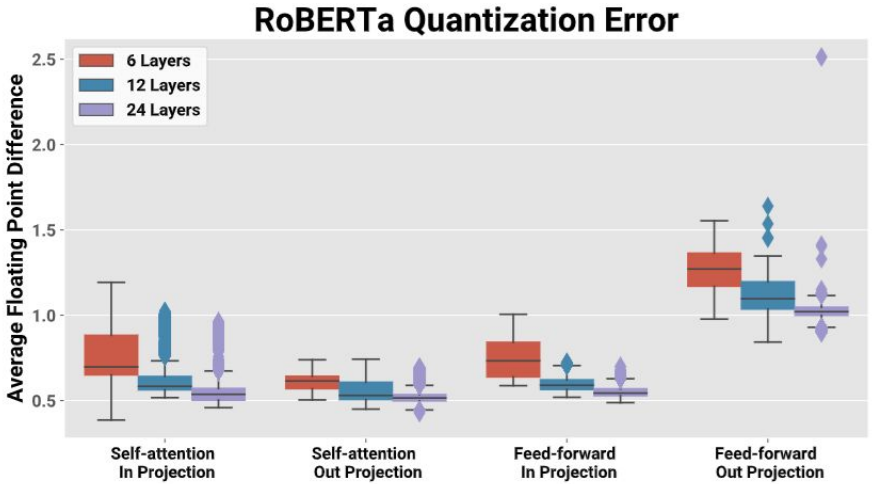
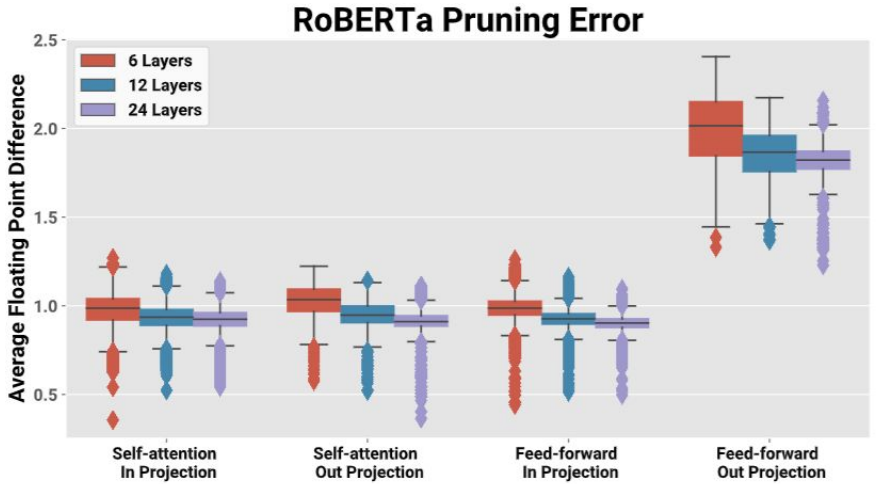


Figure 12. Compression for SST-2. For most budgets (x-axis), the highest accuracy SST-2 models are the ones which are trained large and then heavily compressed. We show results for quantization (left), pruning (center), and quantization and pruning (right).

Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers

Train bigger models and then compress!

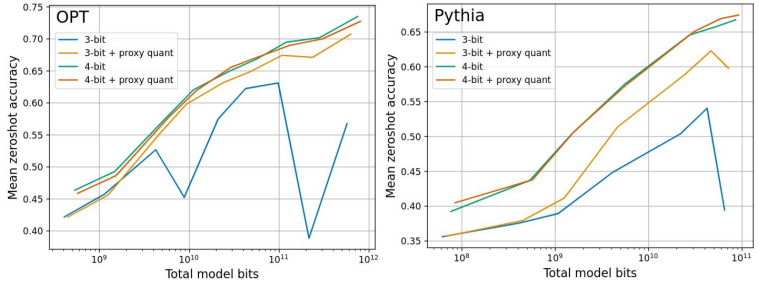




Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers

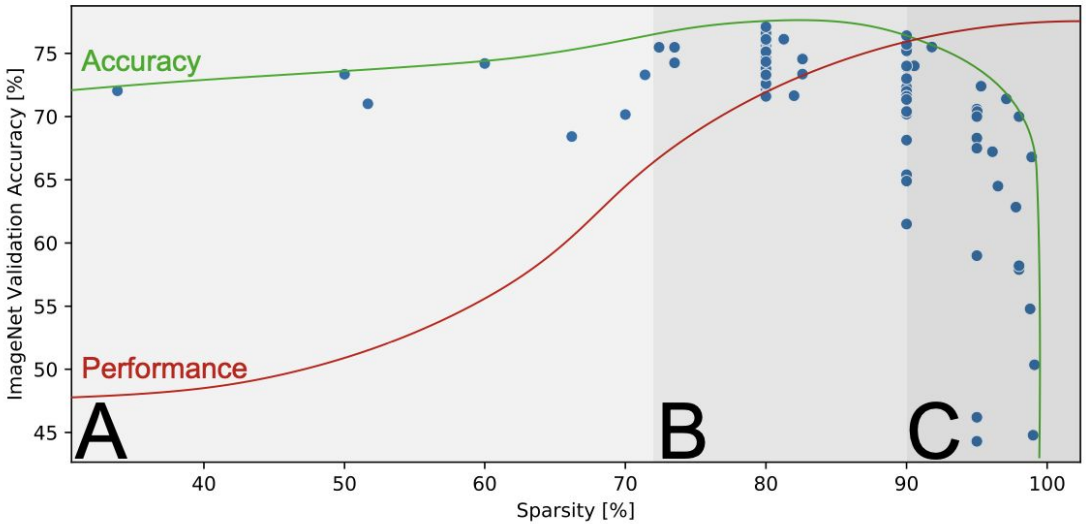
Train bigger models and then compress!

Common Practice	Train Small Model	Stop Training When Converged	Lightly Compress
Optimal	Train Large Model	Stop Training Early	Heavily Compress



The case for 4-bit precision: k-bit Inference Scaling Laws

Train bigger models and then compress!

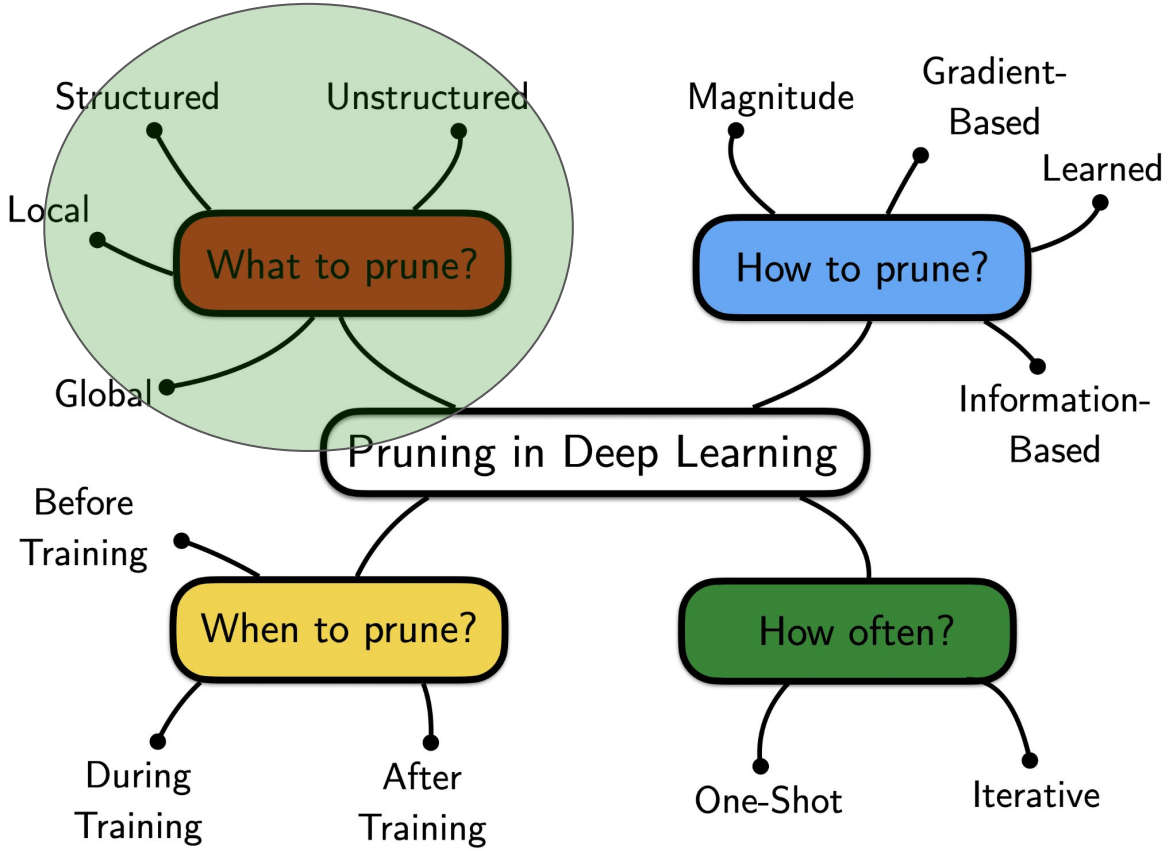


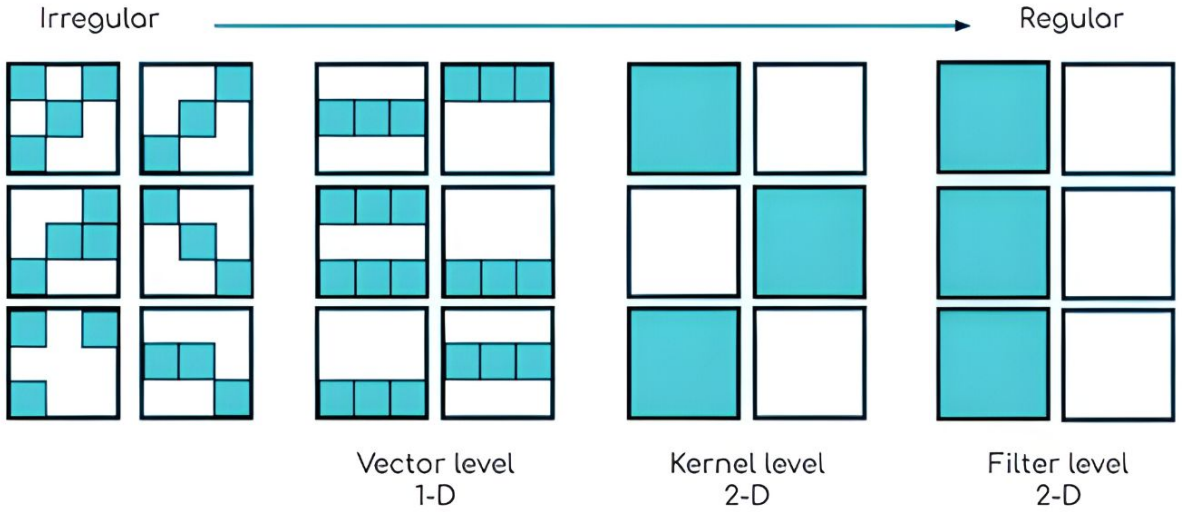
source

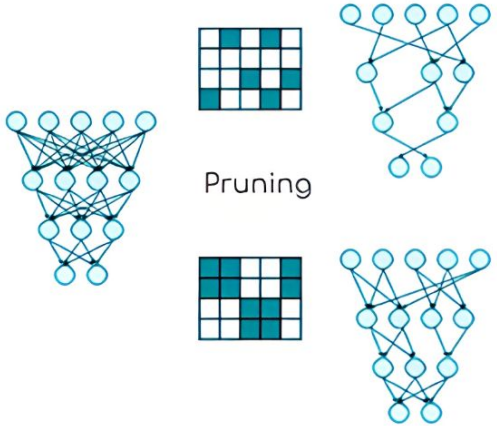
Resnet 50

Sometimes sparsification improves performance.

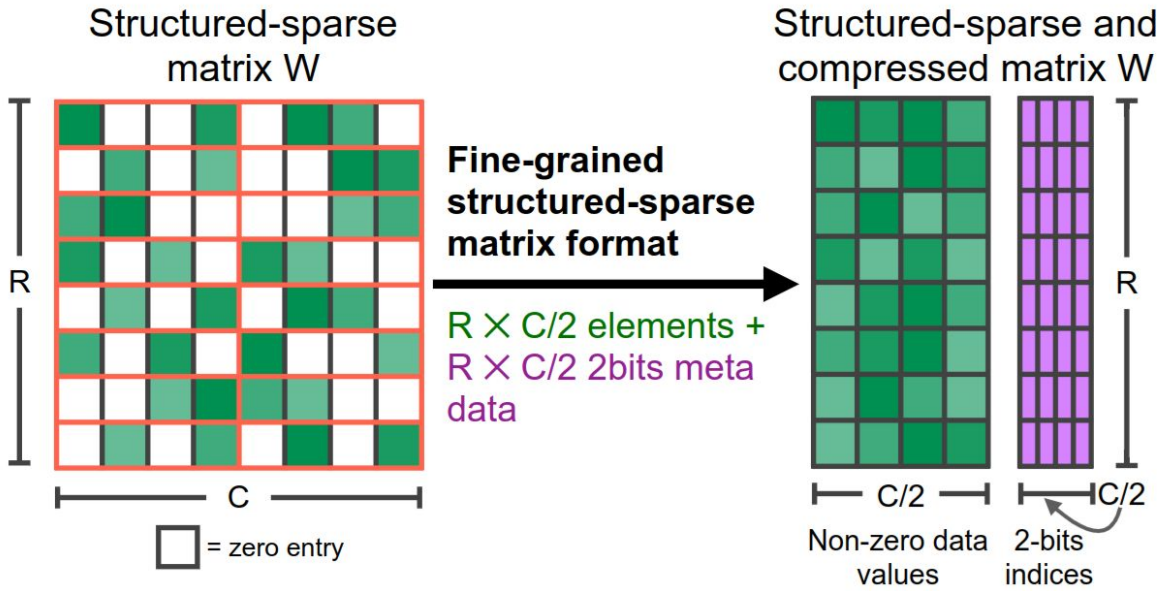
>>> Sparse models :
Hardware view


Structured
Unstructured
Local
What to prune?
Global
Magnitude
Gradient-Based
Learned
How to prune?
Information-Based
Pruning in Deep Learning
Before Training
When to prune?
During Training
After Training
How often?
One-Shot
Iterative


Irregular
Regular
Vector level
1-D
Kernel level
2-D
Filter level
2-D


Pruning

N:M sparsity is a structured sparsity pattern that works well with modern GPU hardware optimization, in which N out of every M consecutive elements are zeros.

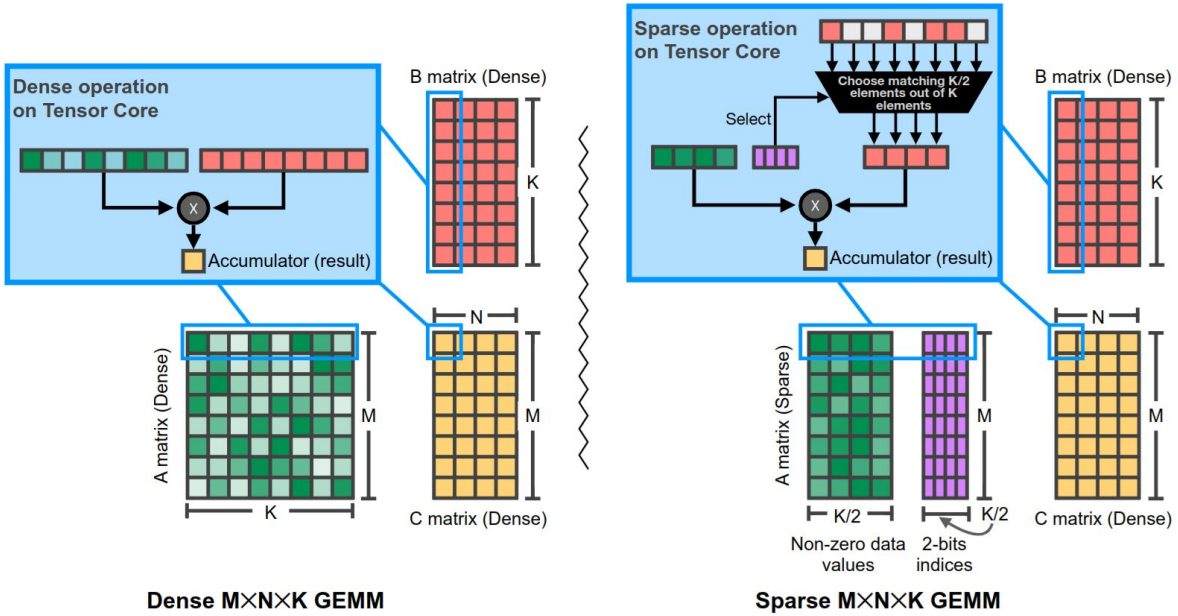


Accelerating Sparse Deep Neural Networks

Compressed format efficiency: Using CSR format for unstructured sparsity can introduce storage overhead due to metadata of up to 200%

Due to its 4-value block size, the 2:4 sparse storage format requires only 2-bits metadata per value, limiting storage overhead to 12.5% and 25% for 16b and 8b values, respectively.

2:4 sparsity
~ 50% weight reduction
~ 2x math arithmetics acceleration

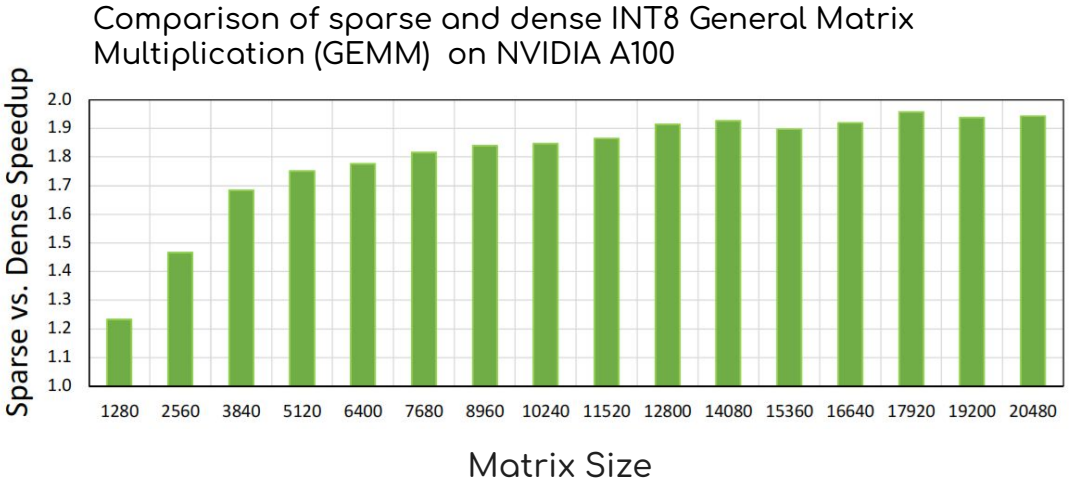


Currently supported only with:

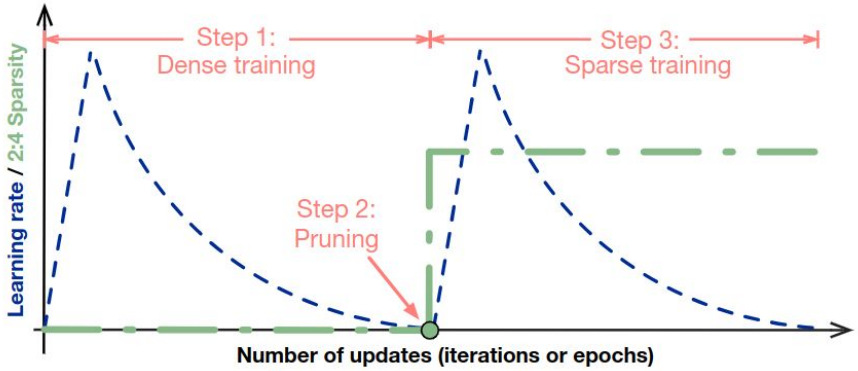
FP32, FP16, int8.

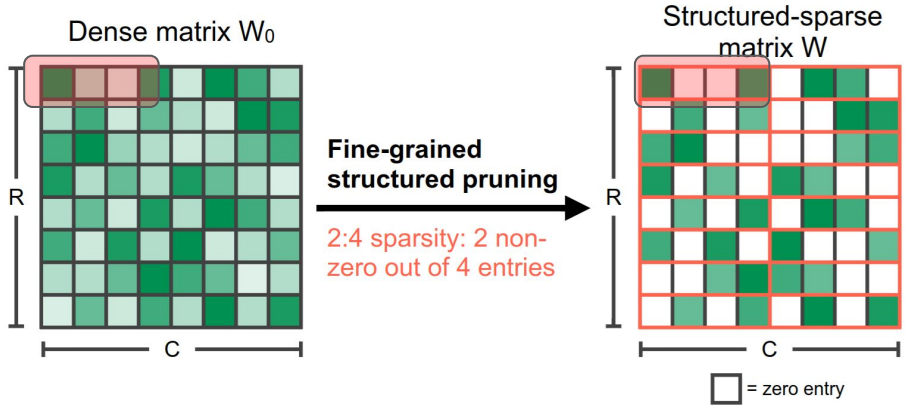
[Accelerating Sparse Deep Neural Networks]

2:4 sparsity
~ 50% weight reduction
~ 2x math arithmetics acceleration



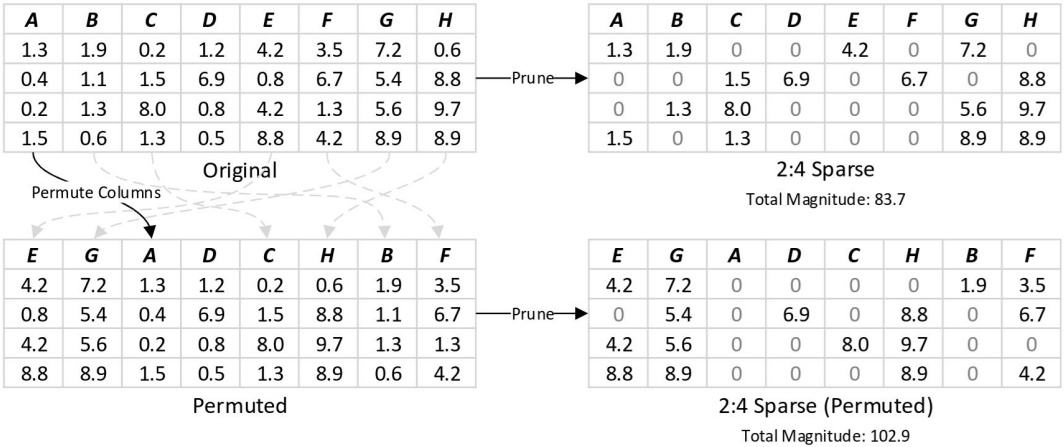
[Accelerating Sparse Deep Neural Networks]





Remove two values with the lowest magnitude
In each cell.

[Accelerating Sparse Deep Neural Networks]



Permute columns to keep more important weights with higher magnitude

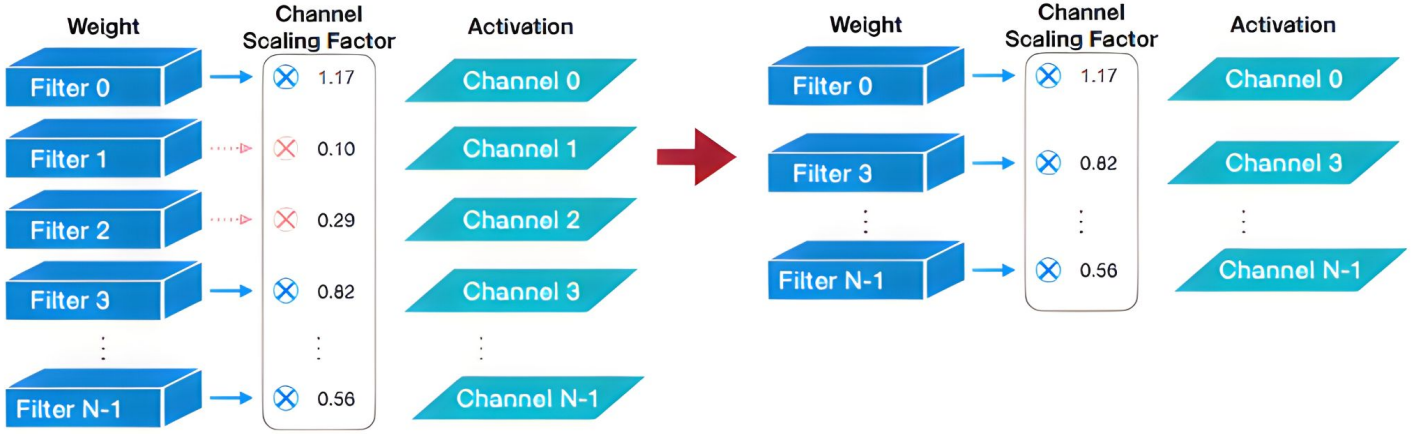
Optimize for bigger total magnitude

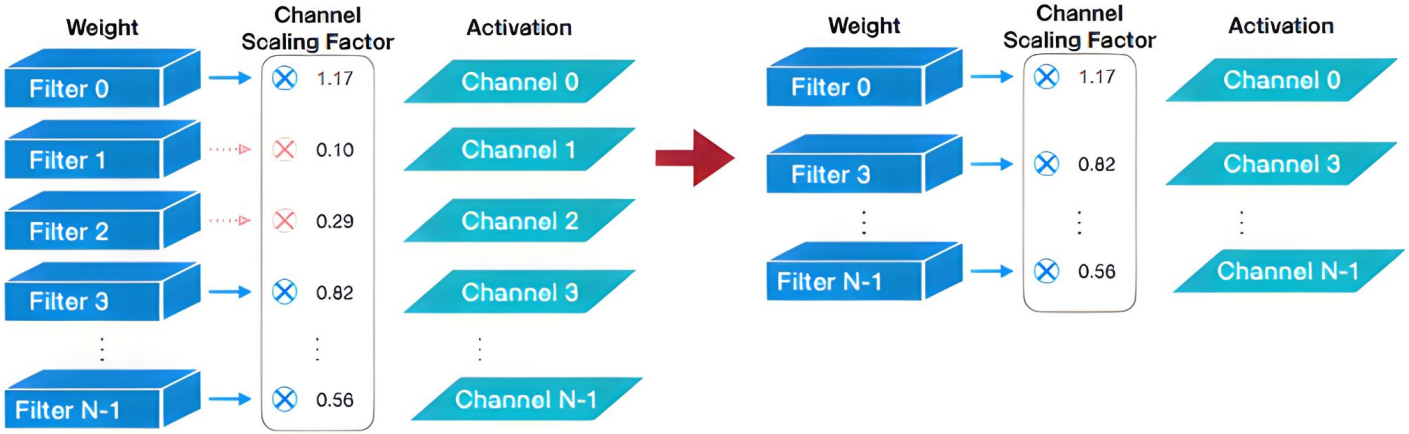
[Accelerating Sparse Deep Neural Networks]

Network	Accuracy		
	Dense FP16	Sparse FP16	Sparse INT8
ResNet-34	73.7	73.9	73.7
ResNet-50	76.1	76.2	76.2
ResNet-50 (SWSL)	81.1	80.9	80.9
ResNet-101	77.7	78.0	77.9
ResNeXt-50-32x4	77.6	77.7	77.7
ResNeXt-101-32x16	79.7	79.9	79.9
ResNeXt-101-32x16 (WSL)	84.2	84.0	84.2
DenseNet-121	75.5	75.3	75.3
DenseNet-161	78.8	78.8	78.9
Wide ResNet-50	78.5	78.6	78.5
Wide ResNet-101	78.9	79.2	79.1
Inception v3	77.1	77.1	77.1
Xception	79.2	79.2	79.2
VGG-11	70.9	70.9	70.8
VGG-16	74.0	74.1	74.1
VGG-19	75.0	75.0	75.0
SUNet-128	75.6	76.0	75.4
SUNet-7-128	76.4	76.5	76.3
DRN26	75.2	75.3	75.3
DRN-105	79.4	79.5	79.4

Accelerating Sparse Deep Neural Networks

Channel-Wise sparsification



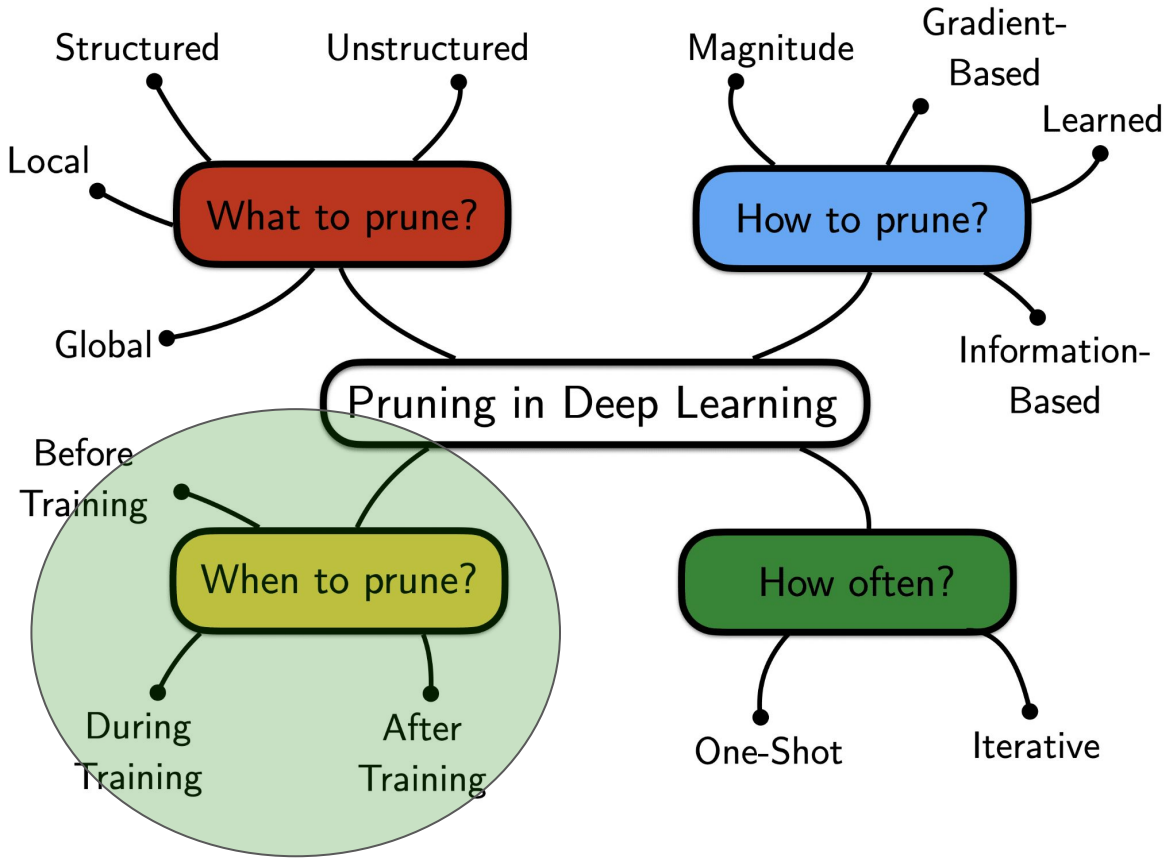


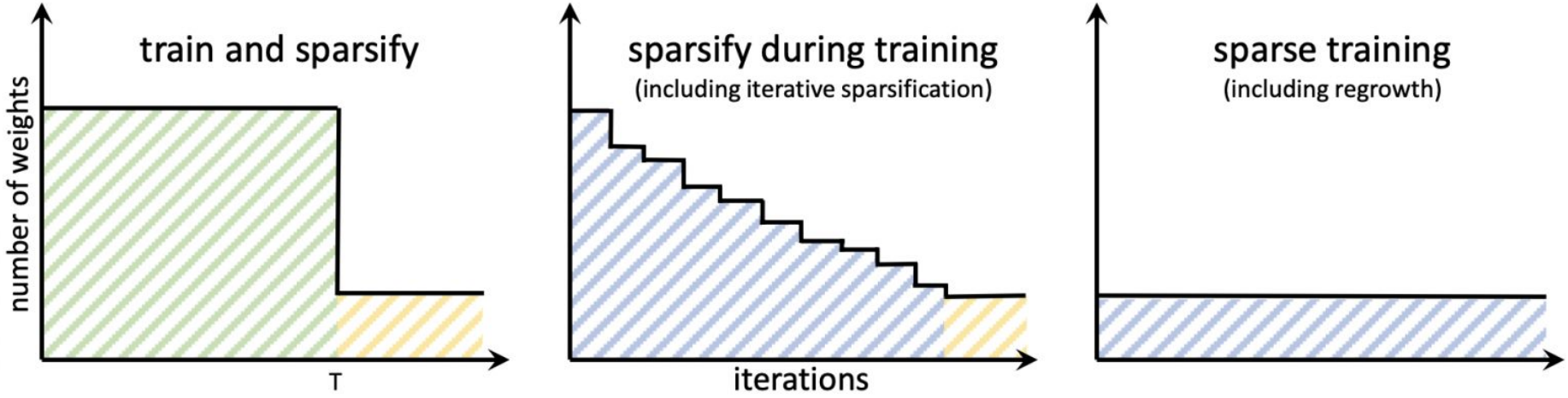
Use BatchNorm values to remove channels

$$\hat{x}_i = \frac{x_i - \mu_b^k}{\sqrt{(\sigma_b^k)^2 + \epsilon}}$$

$$y = \hat{x}_i \gamma + \beta$$

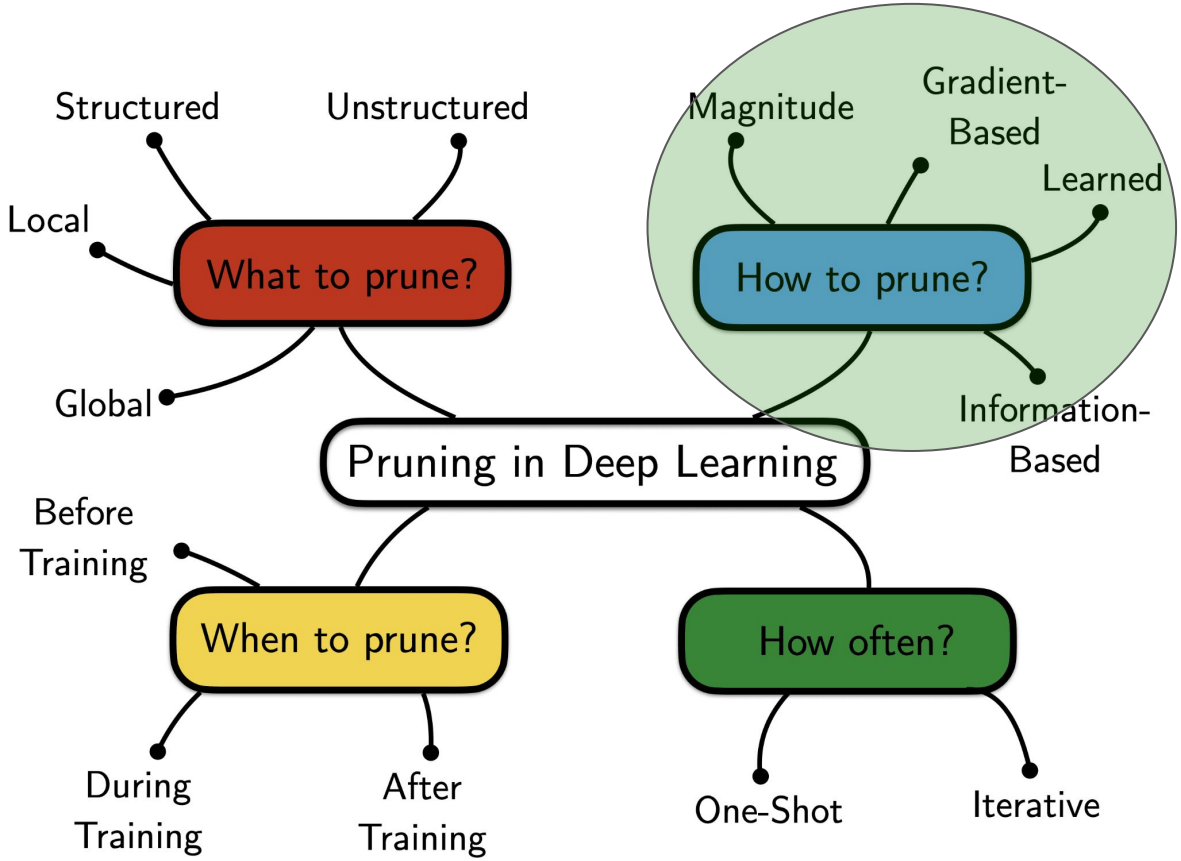
>>> When to prune


Structured
Unstructured
Local
What to prune?
Global
Magnitude
Gradient-Based
Learned
How to prune?
Information-Based
Pruning in Deep Learning
Before Training
When to prune?
During Training
After Training
How often?
One-Shot
Iterative



Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

1. SNIP: Single-shot Network Pruning based on Connection Sensitivity
2. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science
3. Rigging the Lottery: Making All Tickets Winners
4. Picking Winning Tickets Before Training by Preserving Gradient Flow


Structured
Unstructured
Local
What to prune?
Global
Magnitude
Gradient-Based
Learned
How to prune?
Information-Based
Pruning in Deep Learning
Before Training
When to prune?
During Training
After Training
How often?
One-Shot
Iterative

Typical sparsification pipeline

– Train and prune a dense model, then fine-tune the remaining weights in the model to recover accuracy

– Train a dense model with gradual pruning to obtain a sparse model

– Train a sparse model with a sparsity pattern selected a priori, or

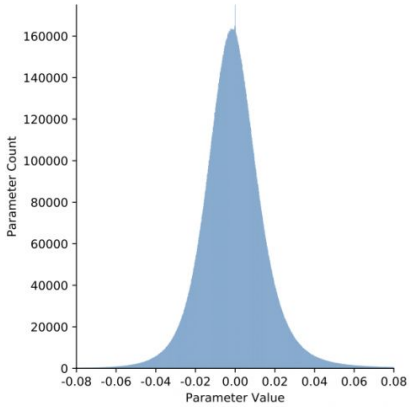
– Train a sparse model with a sparsity pattern determined based on trained dense version.

Accelerating Sparse Deep Neural Networks

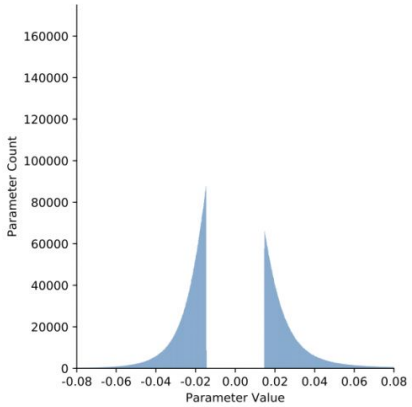
How to choose weight importance?

- Weight magnitude
- Activations magnitude
- First derivative
- Else ?

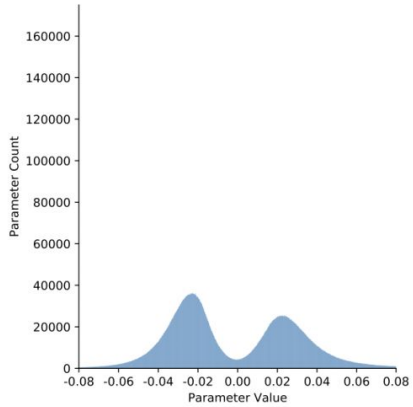
Magnitude Based - Data Free



(a) Dense Network (76.0%)



(b) 70% Pruned (36.1%)



(c) After 3-epoch Retraining (71.4%)

>>>

Data Aware “No Training”
Methods:
OBS, OBD, EBD

OBD - Optimal Brain Damage 1989 (LeCun)

OBS - Optimal Brain Surgeon 1993

EBD - Early Bird Damage 1996

Wood - Fisher 2020 (on estimates of the inverse Hessian)

M - FAC 2021 (on estimates of the inverse Hessian)

Optimal Bert Surgeon 2022

Assume that $W \in \mathcal{R}^d$ and Hessian is $W \in \mathcal{R}^{d \times d}$ The Taylor expansion of change in the training error is given by:

$$\mathcal{E}(W^*) = \mathcal{L}(W) - \mathcal{L}(W^*) = \frac{\mathcal{L}'(W^*)}{1!}(W - W^*) + \frac{\mathcal{L}''(W^*)}{1!}(W - W^*)^2 + ...$$

$$\mathcal{E}(W^*) = (\frac{\partial L}{\partial W})^T \cdot \delta W + \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W + \mathcal{O}(||\delta W||^3)$$

>> Detailed derivation and explanation can be found here

Assume that $W \in \mathcal{R}^d$ and Hessian is $W \in \mathcal{R}^{d \times d}$ The Taylor expansion of change in the training error is given by:

$$\mathcal{E}(W^*) = \mathcal{L}(W) - \mathcal{L}(W^*) = \frac{\mathcal{L}'(W^*)}{1!}(W - W^*) + \frac{\mathcal{L}''(W^*)}{1!}(W - W^*)^2 + ...$$

$$\mathcal{E}(W^*) = (\frac{\partial L}{\partial W})^T \cdot \delta W + \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W + \mathcal{O}(||\delta W||^3)$$

Given that Wvis well-optimized, it is reasonable in practice to assume that $\frac{\partial L}{\partial W^*} \approx 0$

$\delta W = W^* - W_0$ ay correspond to quantization or pruning.

For sparsification we have $\delta W = W^* - W^* \cdot M$, where M is a binary matrix.

For quantization we have $\delta W = W^* - Q(W^*)$, where Q is a quantization funciton.

Then, the change in loss incurred by pruning or quantizing a subset of weights can be expressed as

$$\mathcal{E}(W^*) \approx \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W$$

Assume that $W \in \mathcal{R}^d$ and Hessian is $W \in \mathcal{R}^{d \times d}$ The Taylor expansion of change in the training error is given by:

$$\mathcal{E}(W^*) = \mathcal{L}(W) - \mathcal{L}(W^*) = \frac{\mathcal{L}'(W^*)}{1!}(W - W^*) + \frac{\mathcal{L}''(W^*)}{1!}(W - W^*)^2 + ...$$

$$\mathcal{E}(W^*) = (\frac{\partial L}{\partial W})^T \cdot \delta W + \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W + \mathcal{O}(||\delta W||^3)$$

Computing $\mathcal{H}$ would require computing $\frac{1}{2}N^2$ elements, where N a number of parameters in a model.

Let's assume that change in $\mathcal{E}$ is caused by weights perturbations individually. And so that $\mathcal{H}$ is diagonal.

$$\mathcal{E}(W^*) \approx \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W \approx \frac{1}{2}\sum_i h_{ii}\delta W_i^2 \quad (1)$$

$$h_{ij} = \frac{\partial L^2}{\partial W_i \partial W_j} \quad (2)$$

Assume that $W \in \mathcal{R}^d$ and Hessian is $W \in \mathcal{R}^{d \times d}$ The Taylor expansion of change in the training error is given by:

$$\mathcal{E}(W^*) = \mathcal{L}(W) - \mathcal{L}(W^*) = \frac{\mathcal{L}'(W^*)}{1!}(W - W^*) + \frac{\mathcal{L}''(W^*)}{1!}(W - W^*)^2 + ...$$

$$\mathcal{E}(W^*) = (\frac{\partial L}{\partial W})^T \cdot \delta W + \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W + \mathcal{O}(||\delta W||^3)$$

Assume that $W \in \mathcal{R}^d$ and Hessian is $W \in \mathcal{R}^{d \times d}$ The Taylor expansion of change in the training error is given by:

$$\mathcal{E}(W^*) = \mathcal{L}(W) - \mathcal{L}(W^*) = \frac{\mathcal{L}'(W^*)}{1!}(W - W^*) + \frac{\mathcal{L}''(W^*)}{1!}(W - W^*)^2 + ...$$

$$\mathcal{E}(W^*) = (\frac{\partial L}{\partial W})^T \cdot \delta W + \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W + \mathcal{O}(||\delta W||^3)$$

So, we can choose weights to remove which minimize change in the loss function.

$$\mathcal{E}(W^*) \approx \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W$$

Can be used for pruning, quantization or any other weight perturbations

Optimal Brain Damage - Algorithm:

- For each network weight W_i, determine its corresponding $h_{ii} = \frac{\partial L^2}{\partial W_i \partial W_i}$
- For each h_{ii} compute its saliency $\frac{1}{2} h_{ii} w_i^2$
- Sort all values by its saliency scores and delete or quantize the weights

>> Detailed derivation and explanation can be found here

Assume that $W \in \mathcal{R}^d$ and Hessian is $W \in \mathcal{R}^{d \times d}$

$$\mathcal{L}(W^*) = \mathcal{L}_0(W^*) + \cancel{(\frac{\partial L}{\partial W^*})^T \cdot \delta W} + \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W + \mathcal{O}(||\delta W||^3)$$

$$Hessian \in \mathbf{R}^{d \times d}$$
$$W \in \mathbf{R}^d$$

$$\mathcal{H} = \frac{\partial L}{\partial W_i \partial W_j}$$

$$\delta W = W^* - W_0$$

How to compute?

$$\delta \mathcal{L}(\delta W) \approx \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W$$

>> Detailed derivation and explanation can be found here

Computing $\mathcal{H}$ would require computing $\frac{1}{2}N^2$ elements, where N a number of parameters in a model.

Let's assume that change in $\mathcal{E}$ is caused by weights perturbations individually. And so that $\mathcal{H}$ is diagonal.

$$\mathcal{E} \approx \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W \approx \frac{1}{2}\sum_i h_{ii}\delta W_i^2$$

$$h_{ij} = \frac{\partial L}{\partial w_i \partial w_j}$$

Algorithm 1 The algorithm of Optimal Brain Damage

1. Define a sufficiently large network structure.
2. Train the network until its converges.
3. For each network weight w_i, determine its corresponding $h_{ii} = \frac{\partial^2 E}{\partial w_i \partial w_i}$
4. For each h_{ii}, compute its saliency $\frac{1}{2}h_{ii}w_i^2$
5. Sort all saliency values and delete network weight(s) with the lower saliencies

>> Detailed derivation and explanation can be found here

Computing $\mathcal{H}$ would require computing $\frac{1}{2}N^2$ elements, where N a number of parameters in a model.

Let's assume that change in $\mathcal{E}$ is caused by weights perturbations individually. And so that $\mathcal{H}$ is diagonal.

$$\mathcal{E} \approx \frac{1}{2}\delta W^T \cdot \mathcal{H} \cdot \delta W \approx \frac{1}{2}\sum_i h_{ii}\delta W_i^2$$

$$h_{ij} = \frac{\partial L}{\partial w_i \partial w_j}$$

The same can be used for quantization

Algorithm 1 The algorithm of Optimal Brain Damage

1. Define a sufficiently large network structure.
2. Train the network until its converges.
3. For each network weight w_i, determine its corresponding $h_{ii} = \frac{\partial^2 E}{\partial w_i \partial w_i}$
4. For each h_{ii}, compute its saliency $\frac{1}{2}h_{ii}w_i^2$
5. Sort all saliency values and delete network weight(s) with the lower saliencies

>> Detailed derivation and explanation can be found here

Optimal Brain Surgeon

Find such δW that one of the weights becomes zero or such that one of the weights becomes quantize. Assume e_i is a unit vector for with ith element being modified.

- Pruning: $e_i^T \times \delta W + W_i = 0$
- Quantization: $e_i^T \times \delta W + (W_i - Q(W_i) = 0$

Then we need to optimize $\mathcal{E}$ with one of the constrains above, we consider pruning:

$$\min_{i \in W} \Big\{ \min_{\delta W} \big(\frac{1}{2} \delta W^T \cdot \mathcal{H} \cdot \delta W \big) | e_i^T \cdot \delta W + W_i = 0 \Big\}$$

$$\min_{i \in W} \Big\{ \min_{\delta W} \big(\frac{1}{2} \delta W^T \cdot \mathcal{H} \cdot \delta W \big) | e_i^T \cdot \delta W + W_i = 0 \Big\}$$

For that, we use Lagrange multlipliers:

$$L = \frac{1}{2} \delta W^T \cdot \mathcal{H} \cdot \delta W + \lambda (e_i^T \cdot \delta W + W_i)$$

$$\mathcal{H} \delta W + \lambda e_i^T = 0$$

$$e_i^T \cdot \delta W + W_i = 0$$

$$\lambda e_i^T \mathbf{H}^{-1} e_i^T = W_i$$

$$\lambda = \frac{W_i}{[\mathcal{H}^{-1}]_{ii}}$$

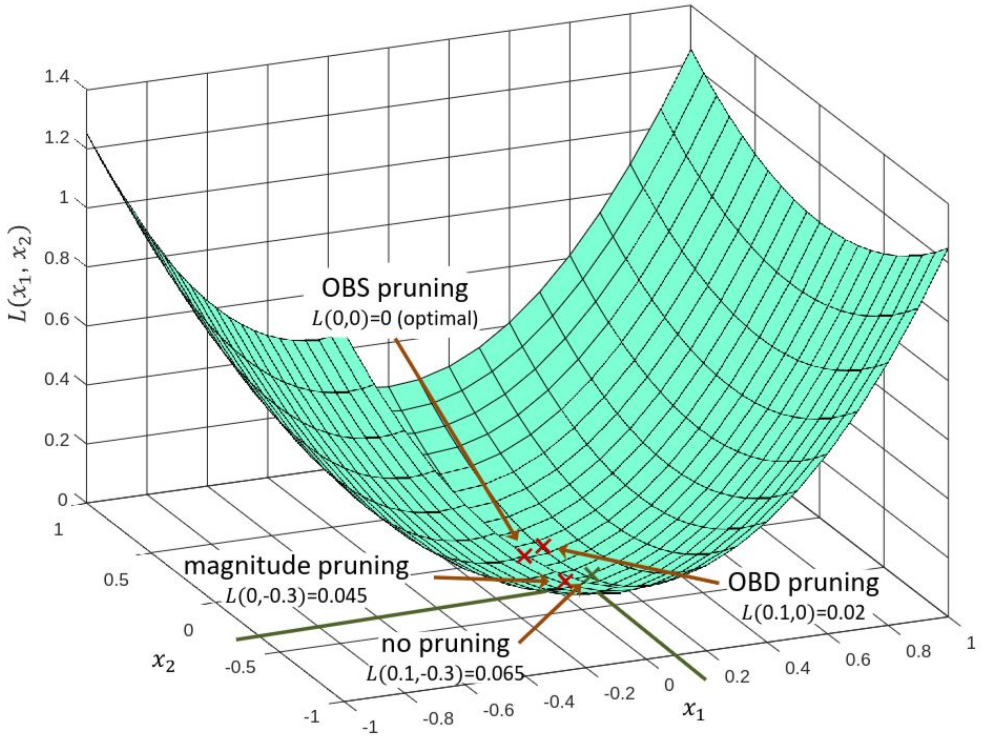
How other weights should change if we remove one weight.

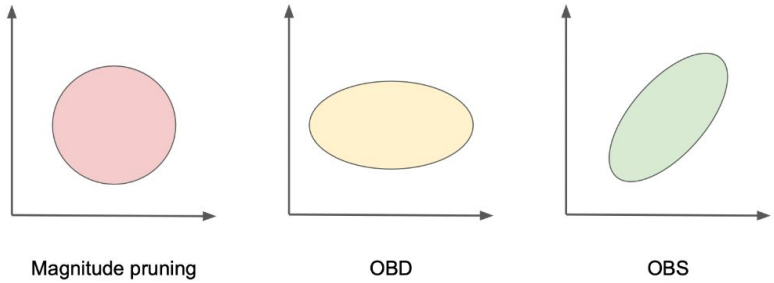
$$\delta W = -\frac{W_i}{[\mathcal{H}^{-1}]_{ii}} \mathcal{H}^{-1} \cdot e_i$$

$$\mathcal{E} = \frac{1}{2} \frac{W_i^2}{[\mathcal{H}^{-1}]_{ii}}$$

Optimal Brain Surgeon - Algorithm:

- Compute H^{-1}
- Fin weights that has the smallest importance $\mathcal{E} = \frac{1}{2} \frac{W_i^2}{[\mathcal{H}^{-1}]_{ii}}$
- Remove or quantize these weights
- Update remaining weights $\delta W = -\frac{W_i}{[\mathcal{H}^{-1}]_{ii}} \mathcal{H}^{-1} \cdot e_i$
- Repeat


OBS pruning
$L(0,0)=0$ (optimal)
magnitude pruning
$L(0,-0.3)=0.045$
no pruning
$L(0.1,-0.3)=0.065$
OBD pruning
$L(0.1,0)=0.02$
$L(x_1, x_2)$
x_1
x_2


Magnitude pruning
OBD
OBS

OBS & GPTQ For LLM

Now we need to adapt this solution to LLM

Problems:

- Computing H for all weighs is expensive;
- There are many parameters to be updated;
- We need to update H after each deletion ;

OBS & GPTQ For LLM

Now we need to adapt this solution to LLM

Problems:

- Computing H for all weighs is expensive;
- There are many parameters to be updated;
- We need to update H after each deletion ;

Algorithm 1 Prune $k \leq d_{\text{col}}$ weights from row $\mathbf{w}$ with inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^\top)^{-1}$ according to OBS in $O(k \cdot d_{\text{col}}^2)$ time.

$M = \{1, \ldots, d_{\text{col}}\}$
for $i = 1, \ldots, k$ **do**
  $p \leftarrow \text{argmin}_{p \in M} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p^2$
  $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}_{:,p} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p$
  $\mathbf{H}^{-1} \leftarrow \mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:}$
  $M \leftarrow M - \{p\}$
end for

Solutions:

- Consider a layerwise problem and minimize MSE between compressed and not compressed activations.
- Perform updates row-wise;

$$\mathcal{L} = ||(WX^T - \hat{W}X^T)||^2$$

$$\mathcal{H} = \frac{\partial^2 \mathcal{L}}{\partial^2 W} = \frac{\partial^2}{\partial^2 W} ||(WX^T - \hat{W}X^T)||^2 = XX^T$$

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning

OBS & GPTQ For LLM

Method	ResNet50 – 76.13			YOLOv5l – 66.97			BERT – 88.53		
	2×	3×	4×	2×	3×	4×	2×	3×	4×
GMP	74.86	71.44	64.84	65.83	62.30	55.09	65.64	12.52	09.23
L-OBS	75.48	73.73	71.24	**66.21**	64.47	61.15	77.67	3.62	6.63
AdaPrune	75.53	74.47	72.39	66.00	64.88	62.71	87.12	70.32	18.75
ExactOBS	**75.64**	**75.01**	**74.05**	66.14	**65.35**	**64.05**	**87.81**	**85.87**	**82.10**

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning

SparseGPT

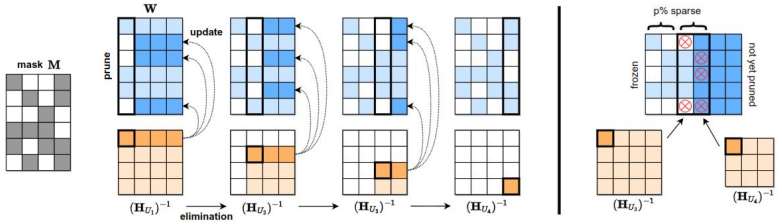


Figure 4. [Left] Visualization of the `SparseGPT` reconstruction algorithm. Given a fixed pruning mask $\mathbf{M}$, we incrementally prune weights in each column of the weight matrix $\mathbf{W}$, using a sequence of Hessian inverses $(\mathbf{H}_{U_j})^{-1}$, and updating the remainder of the weights in those rows, located to the "right" of the column being processed. Specifically, the weights to the "right" of a pruned weight (dark blue) will be updated to compensate for the pruning error, whereas the unpruned weights do not generate updates (light blue). [Right] Illustration of the adaptive mask selection via iterative blocking.

Algorithm 1 The `SparseGPT` algorithm. We prune the layer matrix $\mathbf{W}$ to $p\%$ unstructured sparsity given inverse Hessian $\mathbf{H}^{-1} = (\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}$, lazy batch-update blocksize B and adaptive mask selection blocksize B_s; each B_s consecutive columns will be $p\%$ sparse.

$\mathbf{M} \leftarrow \mathbf{1}_{d_{\text{row}} \times d_{\text{col}}}$ *// binary pruning mask*
$\mathbf{E} \leftarrow \mathbf{0}_{d_{\text{row}} \times B}$ *// block quantization errors*
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})^\top$ *// Hessian inverse information*
for $i = 0, B, 2B, \ldots$ **do**
  for $j = i, \ldots, i + B - 1$ **do**
    if $j \bmod B_s = 0$ **then**
      $\mathbf{M}_{:,j:(j+B_s)} \leftarrow$ mask of $(1-p)\%$ weights $w_c \in \mathbf{W}_{:,j:(j+B_s)}$ with largest $w_c^2/[\mathbf{H}^{-1}]_{cc}^2$
    end if
    $\mathbf{E}_{:,j-i} \leftarrow \mathbf{W}_{:,j} \,/\, [\mathbf{H}^{-1}]_{jj}$ *// pruning error*
    $\mathbf{E}_{:,j-i} \leftarrow (\mathbf{1} - \mathbf{M}_{:,j}) \cdot \mathbf{E}_{:,j-i}$ *// freeze weights*
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ *// update*
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ *// update*
end for
$\mathbf{W} \leftarrow \mathbf{W} \cdot \mathbf{M}$ *// set pruned weights to 0*

SparseGPT

SparseGPT

Table 1. OPT perplexity results on raw-WikiText2.

OPT - 50%	125M	350M	1.3B
Dense	27.66	22.00	14.62
Magnitude	193.	97.80	1.7e4
AdaPrune	58.66	48.46	32.52
SparseGPT	**36.85**	**31.58**	**17.46**

OPT	Sparsity	2.7B	6.7B	13B	30B	66B	175B
Dense	0%	12.47	10.86	10.13	9.56	9.34	8.35
Magnitude	50%	265.	969.	1.2e4	168.	4.2e3	4.3e4
SparseGPT	50%	**13.48**	**11.55**	**11.17**	**9.79**	**9.32**	**8.21**
SparseGPT	4:8	14.98	12.56	11.77	10.30	9.65	8.45
SparseGPT	2:4	17.18	14.20	12.96	10.90	10.09	8.74

Table 2. ZeroShot results on several datasets for sparsified variants of OPT-175B.

Method	Spars.	Lamb.	PIQA	ARC-e	ARC-c	Story.	**Avg.**
Dense	0%	75.59	81.07	71.04	43.94	79.82	**70.29**
Magnitude	50%	00.02	54.73	28.03	25.60	47.10	**31.10**
SparseGPT	50%	78.47	80.63	70.45	43.94	79.12	**70.52**
SparseGPT	4:8	80.30	79.54	68.85	41.30	78.10	**69.62**
SparseGPT	2:4	80.92	79.54	68.77	39.25	77.08	**69.11**

SparseGPT

WANDA

Method	Weight Update	Calibration Data	Pruning Metric $\mathbf{S}_{ij}$	Complexity
Magnitude	✗	✗	$\lvert\mathbf{W}_{ij}\rvert$	$O(1)$
SparseGPT	✓	✓	$\left[\lvert\mathbf{W}\rvert^2/\mathrm{diag}\left[(\mathbf{X}\mathbf{X}^T+\lambda\mathbf{I})^{-1}\right]\right]_{ij}$	$O(d^3_{\texttt{hidden}})$
Wanda	✗	✓	$\lvert\mathbf{W}_{ij}\rvert \cdot \lVert\mathbf{X}_j\rVert_2$	$O(d^2_{\texttt{hidden}})$

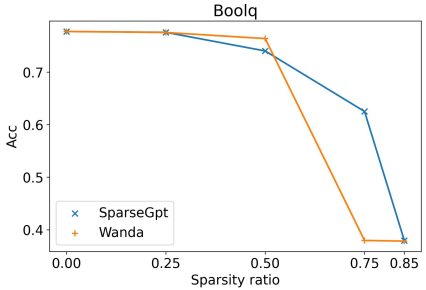
Wanda

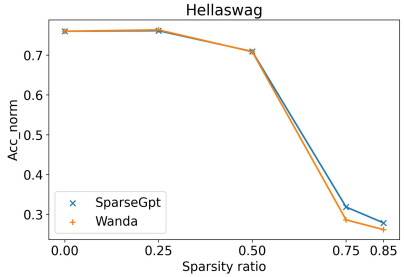
WANDA

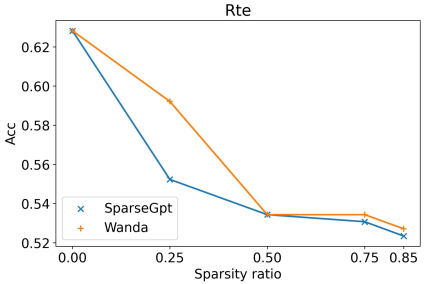
Params	Method	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
7B	Dense	75.05	66.43	56.92	69.93	75.34	41.89	34.40	59.99
	Magnitude	54.59	54.51	45.49	59.19	58.84	33.53	22.40	46.94
	SparseGPT	**72.05**	54.15	51.43	**67.88**	**71.38**	**37.71**	**30.00**	**54.94**
	Wanda	71.22	**55.60**	**51.85**	66.06	69.11	36.86	28.80	54.21
13B	Dense	77.89	70.4	59.94	72.77	77.40	46.50	33.20	62.59
	Magnitude	54.89	51.26	44.16	63.14	58.80	33.79	27.20	47.61
	SparseGPT	**76.97**	61.01	54.95	71.67	72.47	41.98	31.20	58.61
	Wanda	75.90	**62.82**	**55.71**	**71.98**	**73.19**	**43.52**	**32.20**	**59.33**
30B	Dense	82.69	66.79	63.35	75.69	80.30	52.82	36.00	65.38
	Magnitude	64.34	50.18	50.59	66.54	72.39	43.77	29.00	53.83
	SparseGPT	**82.32**	62.45	59.15	**75.22**	78.96	48.56	**35.00**	63.09
	Wanda	81.90	**65.34**	**60.93**	73.48	**79.29**	**49.66**	34.60	**63.60**
65B	Dense	84.83	69.68	64.54	77.27	81.40	52.90	38.20	66.97
	Magnitude	79.15	62.45	61.90	74.74	76.40	49.57	35.00	62.74
	SparseGPT	84.60	70.76	63.90	**77.43**	79.35	**50.85**	37.20	66.30
	Wanda	**84.70**	**71.48**	**64.55**	76.87	**79.75**	50.51	**38.80**	**66.67**

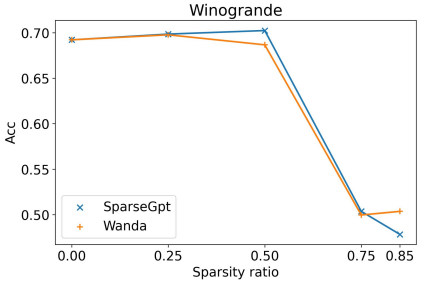
Wanda

WANDA vs SGPT









Wanda

Hessian Approximation

OBD - Optimal Brain Damage 1989 (LeCun)

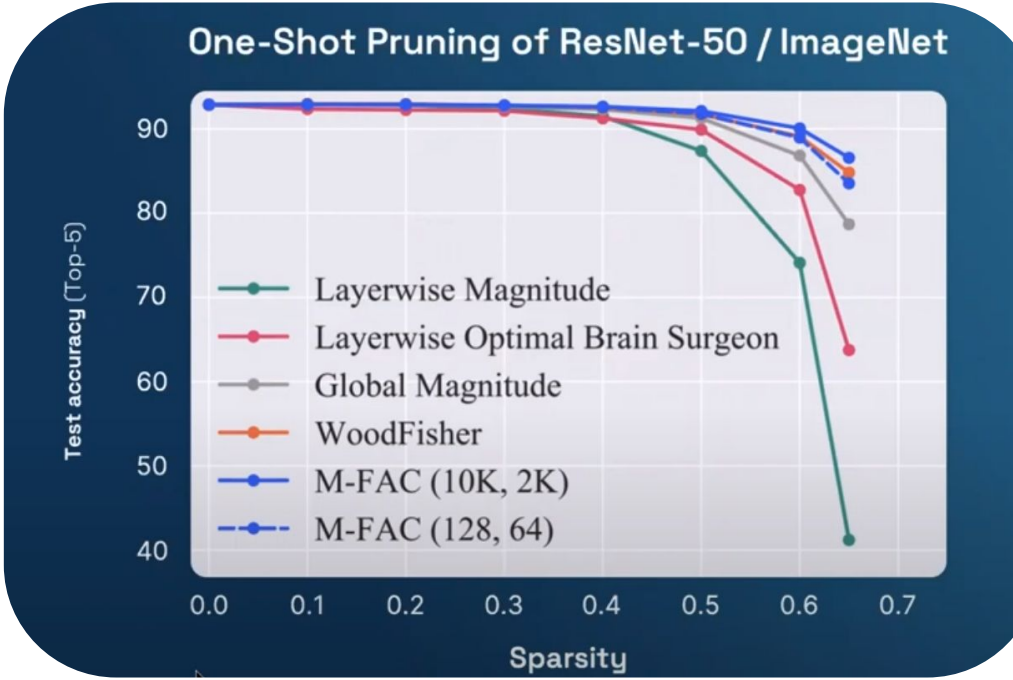
OBS - Optimal Brain Surgeon 1993

EBD - Early Bird Damage 1996

Wood - Fisher 2020 (on estimates of the inverse Hessian)

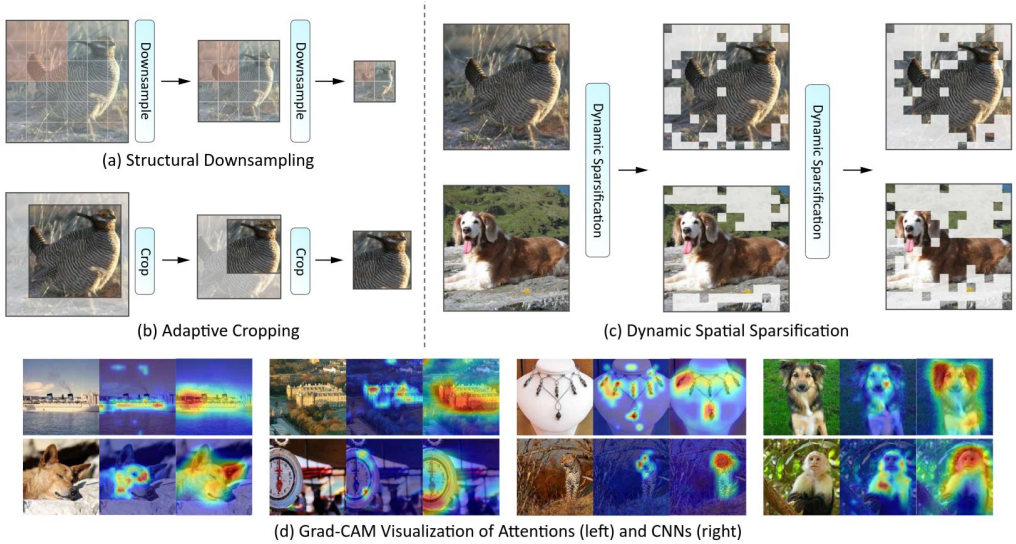
M - FAC 2021 (on estimates of the inverse Hessian)

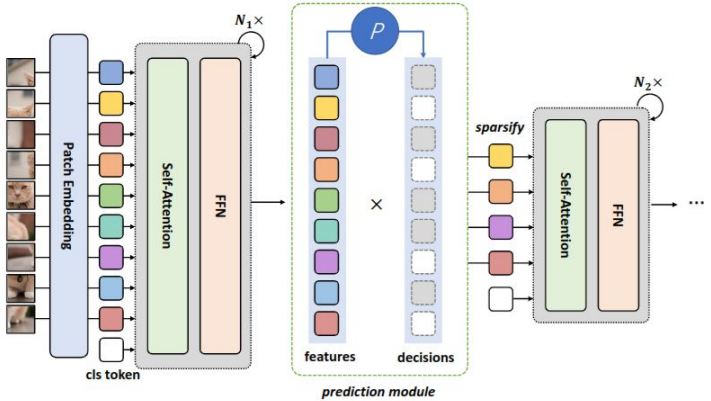
Optimal Bert Surgeon 2022



Source

Dynamic Sparsification





Dynamic Spatial Sparsification for Efficient Vision Transformers and Convolutional Neural Networks

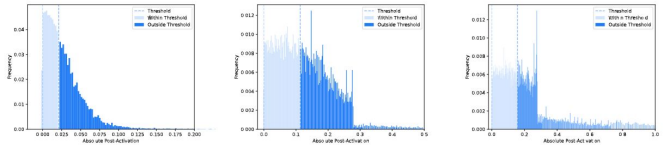
Dynamic Sparsification

$$\text{Gated-MLP}(x) := (\text{SiLU}(xW_{\text{gate}}) * (xW_{\text{up}}))W_{\text{down}}$$

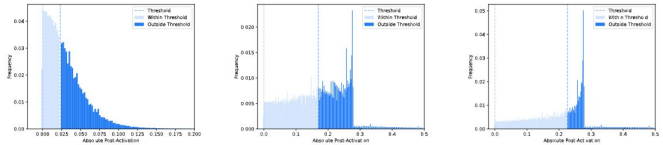
$$\text{CATS}_t(\text{SiLU}(xW_{\text{gate}})) = \begin{cases} \text{SiLU}(xW_{\text{gate}}) & |\text{SiLU}(xW_{\text{gate}})| \geq t \\ 0 & |\text{SiLU}(xW_{\text{gate}})| < t \end{cases}$$

Custom GPU Kernel 1 MLP using CATS

1: **Input:** threshold $t > 0$, hidden layer x, weights W_{gate}, W_{down}, and W_{up}
2: $v \leftarrow \text{SiLU}(xW_{gate})$
3: $\texttt{Mask} \leftarrow 1$ if $|v| \geq t$ else 0
4: $x_1 \leftarrow (xW_{\text{up}}[\texttt{Mask}] * v[\texttt{Mask}])$
5: $y \leftarrow x_1 W_{\text{down}}[\texttt{Mask}]$



(a) Llama2 Layer 0. (b) Llama2 Layer 15. (c) Llama2 Layer 31.



(d) Mistral7B Layer 0. (e) Mistral7B Layer 15. (f) Mistral7B Layer 31.

CATS: Contextually-Aware Thresholding for Sparsity in LargeLanguage Models

Dynamic Sparsification

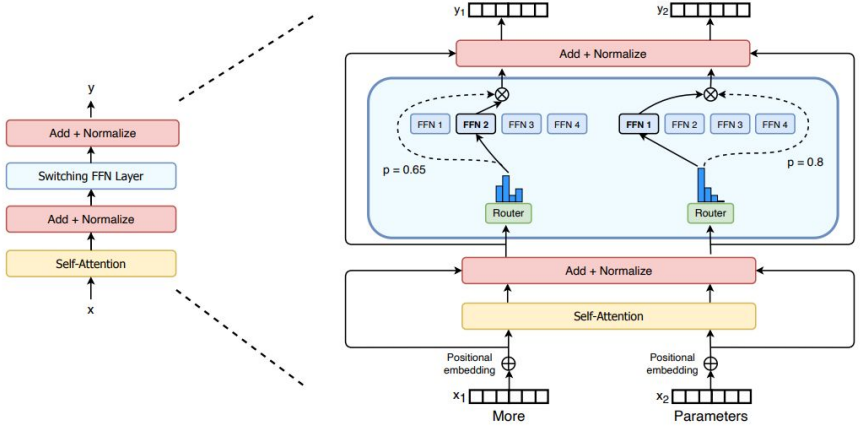
Model ╲ Dataset	WG acc↑	PIQA acc↑	SciQ acc↑	QA acc↑	HS acc↑	BoolQ acc↑	Arc-E acc↑	Arc-C acc↑	Avg acc↑
Mistral-7B	0.7419	0.8069	0.959	0.3260	0.6128	0.8370	0.8085	0.5034	0.6994
CATS 50%	0.7245	0.8009	0.948	0.3200	0.6097	0.8193	0.7849	0.5043	0.6890
CATS 70%	0.7190	0.8003	0.929	0.292	0.6057	0.8028	0.7492	0.4693	0.6709
CATS 90%	0.5627	0.6001	0.422	0.212	0.3359	0.7086	0.3754	0.2773	0.4368
ReLUfication	0.5043	0.5092	0.236	0.142	0.2580	0.4208	0.2723	0.2415	0.3230
Llama2-7B	0.6906	0.7807	0.94	0.314	0.5715	0.7774	0.7630	0.4343	0.6589
CATS 50%	0.6748	0.7693	0.927	0.322	0.5711	0.7263	0.7441	0.4121	0.6433
CATS 70%	0.6693	0.7584	0.902	0.294	0.5500	0.6590	0.7008	0.3805	0.6143
CATS 90%	0.5738	0.6627	0.611	0.212	0.3848	0.6284	0.4566	0.2816	0.4764
ReLUfication	0.4893	0.5408	0.2570	0.154	0.2586	0.6003	0.2795	0.2406	0.3525

CATS: Contextually-Aware Thresholding for Sparsity in LargeLanguage Models

MoE: Mixture of Experts

Mixtral 8x7b

- 8 Experts in total
- Choose only two best performing

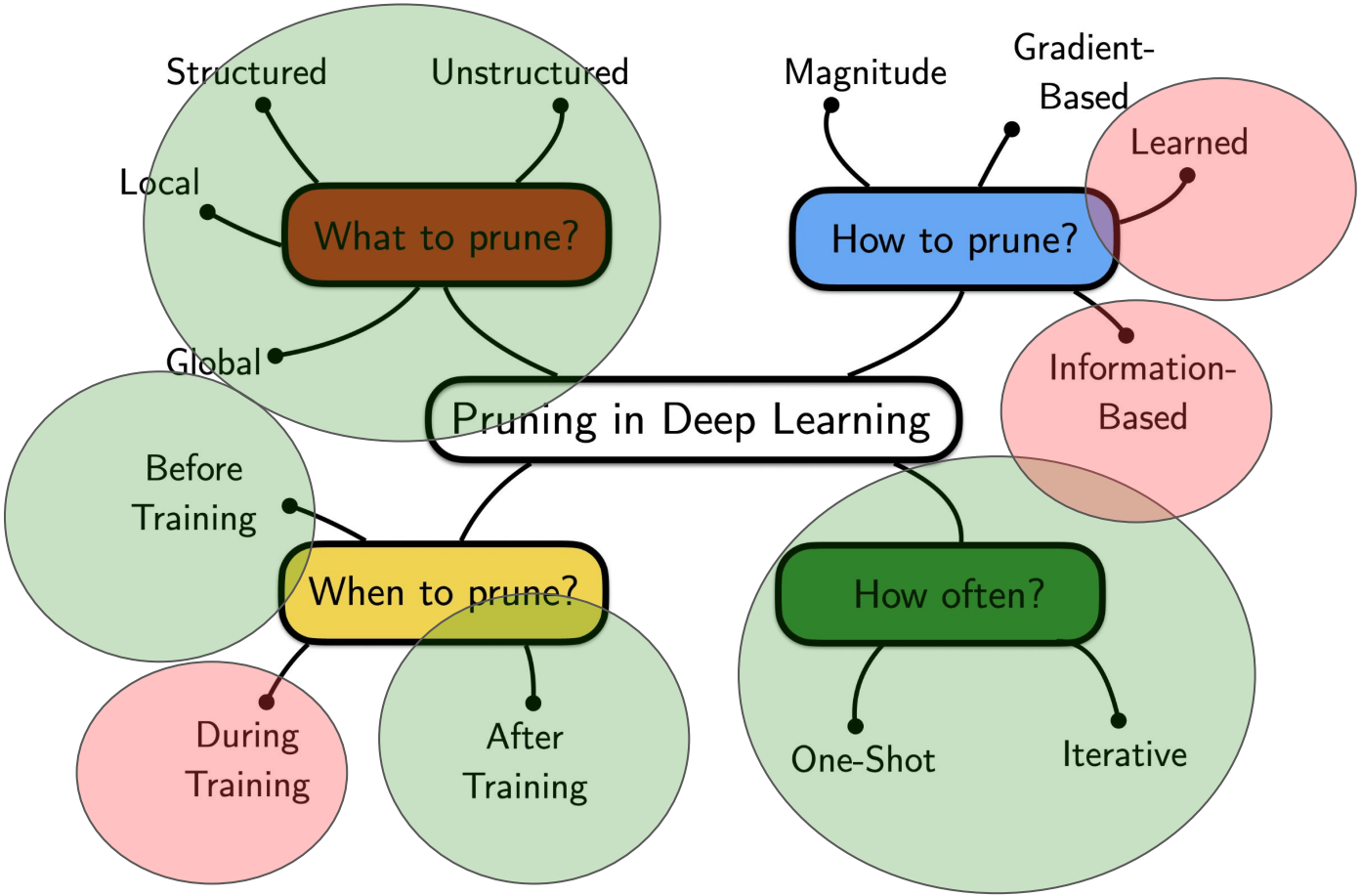


Обработка каждого токена выполняется 2-мя экспертами, которые выбираются Router слоем, в качестве которого используется линейный слой. К выходу каждого из отобранных экспертов применяется функция активации SwiGLU после чего результат складывается.

$$y = \sum_{i=0}^{n-1} \text{Softmax}(\text{Top2}(x \cdot W_g))_i \cdot \text{SwiGLU}_i(x)$$

Model	Active Params	MMLU	HellaS	WinoG	PIQA	Arc-e	Arc-c	NQ	TriQA	HumanE	MBPP	Math	GSM8K
LLaMA 2 7B	7B	44.4%	77.1%	69.5%	77.9%	68.7%	43.2%	17.5%	56.6%	11.6%	26.1%	3.9%	16.0%
LLaMA 2 13B	13B	55.6%	80.7%	72.9%	80.8%	75.2%	48.8%	16.7%	64.0%	18.9%	35.4%	6.0%	34.3%
LLaMA 1 33B	33B	56.8%	83.7%	76.2%	82.2%	79.6%	54.4%	24.1%	68.5%	25.0%	40.9%	8.4%	44.1%
LLaMA 2 70B	70B	69.9%	**85.4%**	**80.4%**	82.6%	79.9%	56.5%	25.4%	**73.0%**	29.3%	49.8%	13.8%	69.6%
Mistral 7B	7B	62.5%	81.0%	74.2%	82.2%	80.5%	54.9%	23.2%	62.5%	26.2%	50.2%	12.7%	50.0%
Mixtral 8x7B	13B	**70.6%**	84.4%	77.2%	**83.6%**	**83.1%**	**59.7%**	**30.6%**	71.5%	**40.2%**	**60.7%**	**28.4%**	**74.4%**

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., ... & Sayed, W. E. (2024). Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

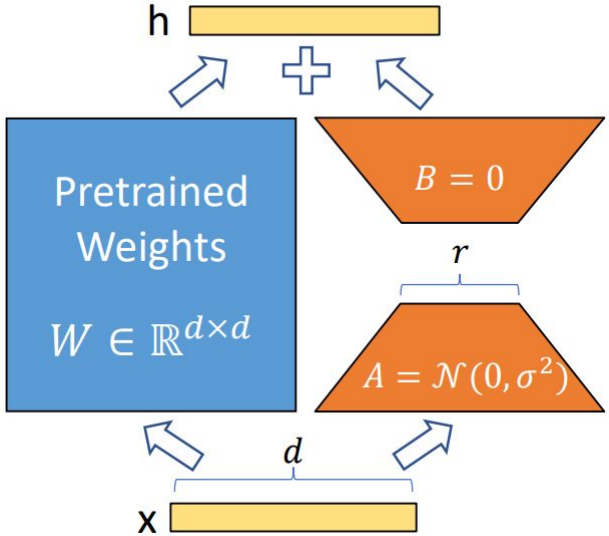

Pruning in Deep Learning
What to prune?
Structured
Unstructured
Local
Global
How to prune?
Magnitude
Gradient-Based
Learned
Information-Based
When to prune?
Before Training
During Training
After Training
How often?
One-Shot
Iterative


>>>
PEFT

LoRA

$\mathbf{W}_{orig} \in \mathbb{R}^{o \times h}$, $\mathbf{B} \in \mathbb{R}^{o \times r}$, $\mathbf{A} \in \mathbb{R}^{r \times h}$ $\quad \mathbf{W}_{train} = \mathbf{W}_{orig} + \mathbf{BA}$

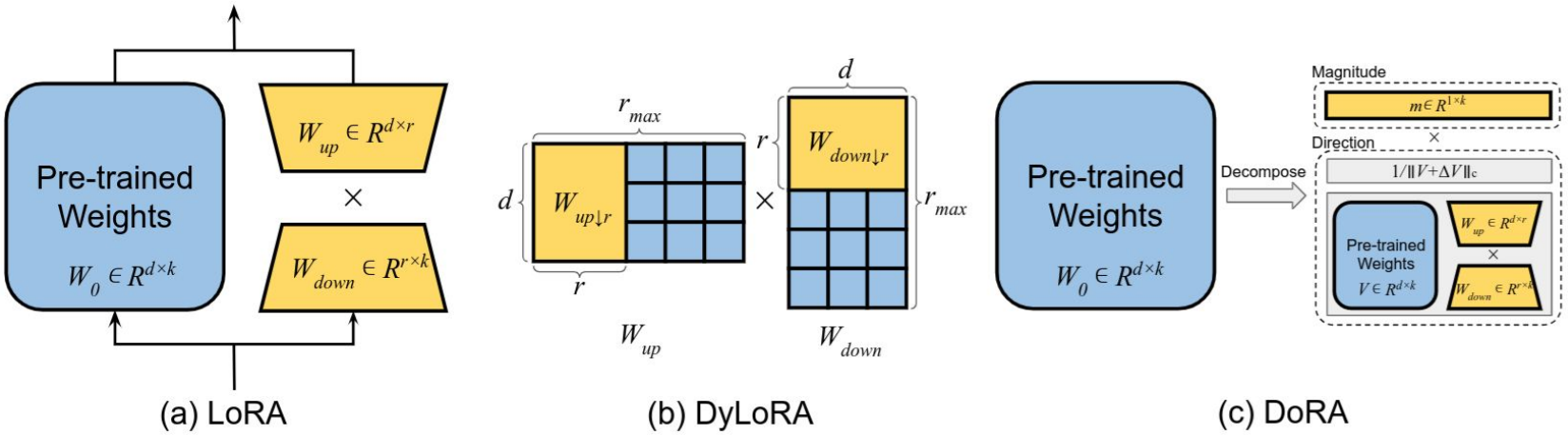
A - Initialize randomly
B - Initialize with zeros



Forward Pass $\quad \mathbf{Out} = \mathbf{X}\mathbf{W}_{train}^{T} = \mathbf{X}\mathbf{W}_{orig}^{T} + \mathbf{X}\mathbf{A}^{T}\mathbf{B}^{T}$

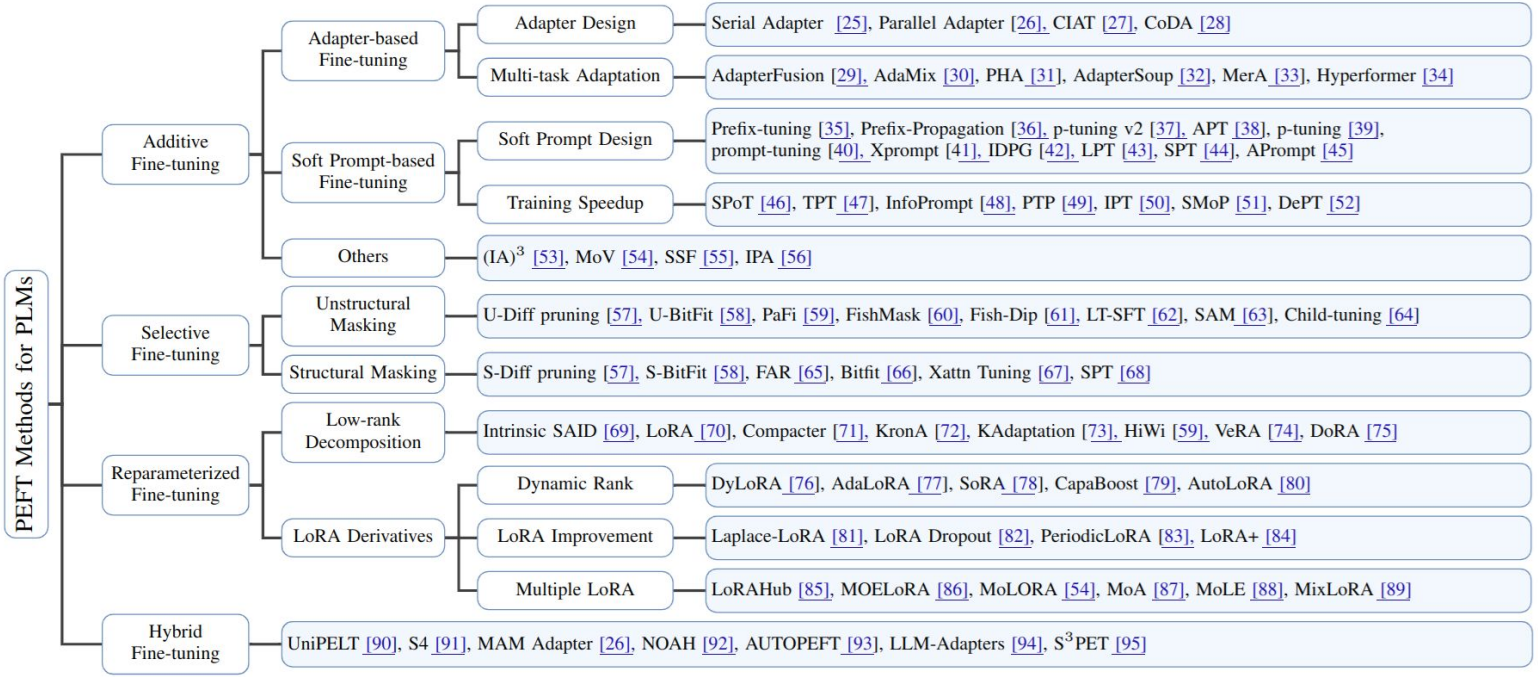
Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models.

PEFT



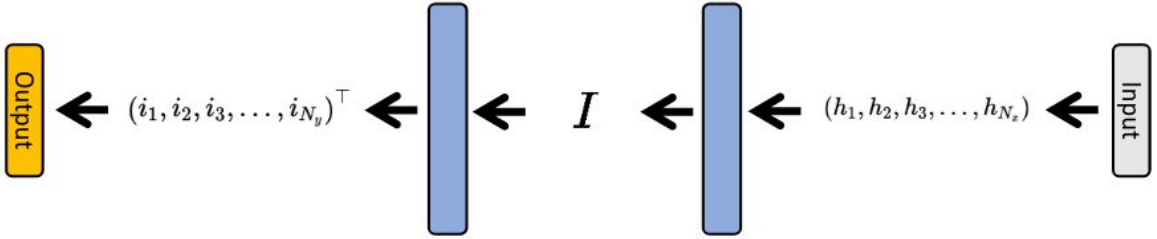
Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey

PEFT



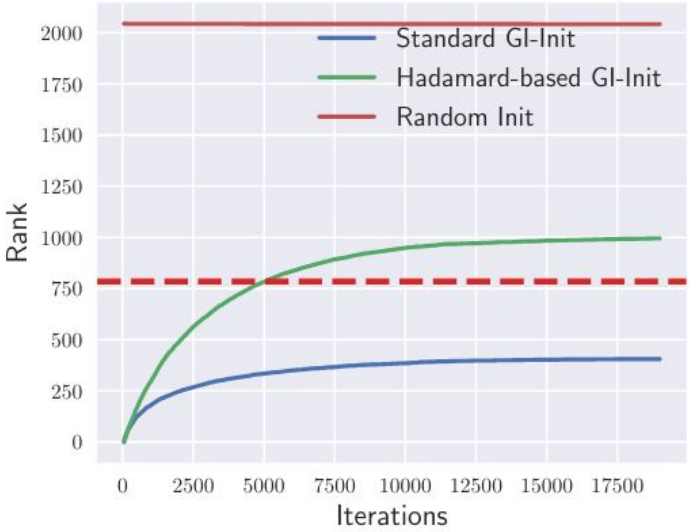
Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey

>>> Weight initialization

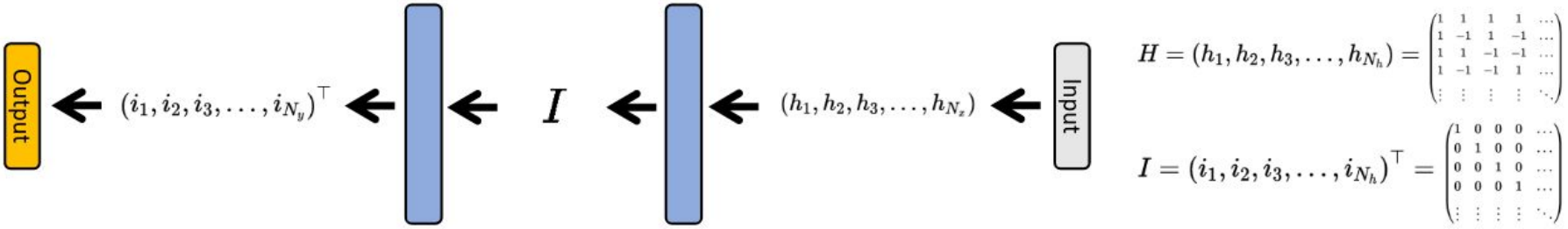


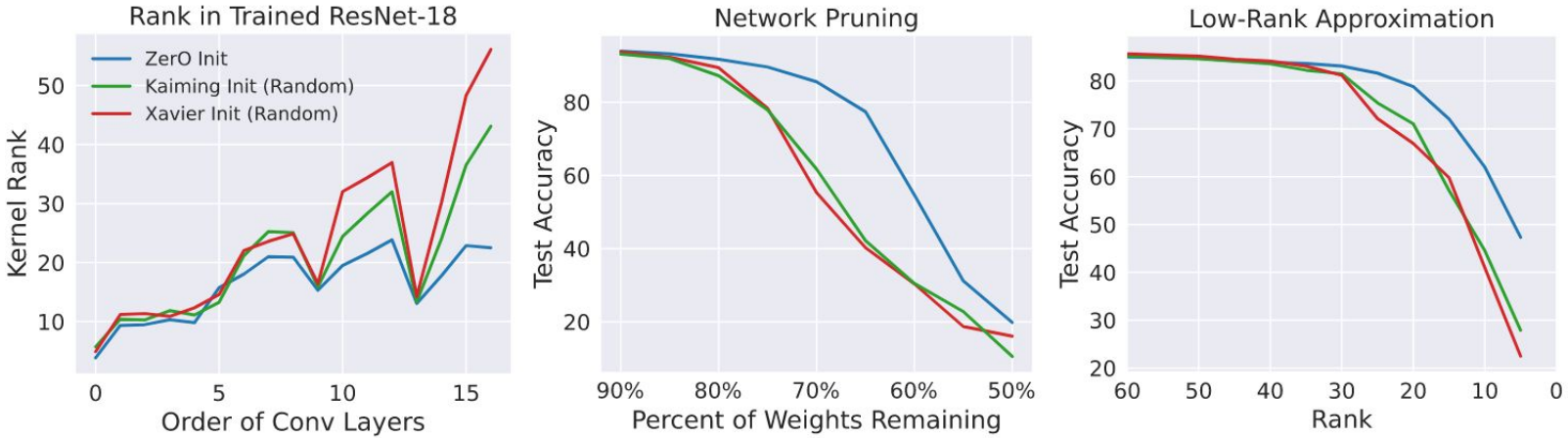
$$H = (h_1, h_2, h_3, \ldots, h_{N_h}) = \begin{pmatrix} 1 & 1 & 1 & 1 & \ldots \\ 1 & -1 & 1 & -1 & \ldots \\ 1 & 1 & -1 & -1 & \ldots \\ 1 & -1 & -1 & 1 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$I = (i_1, i_2, i_3, \ldots, i_{N_h})^\top = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots \\ 0 & 1 & 0 & 0 & \ldots \\ 0 & 0 & 1 & 0 & \ldots \\ 0 & 0 & 0 & 1 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$



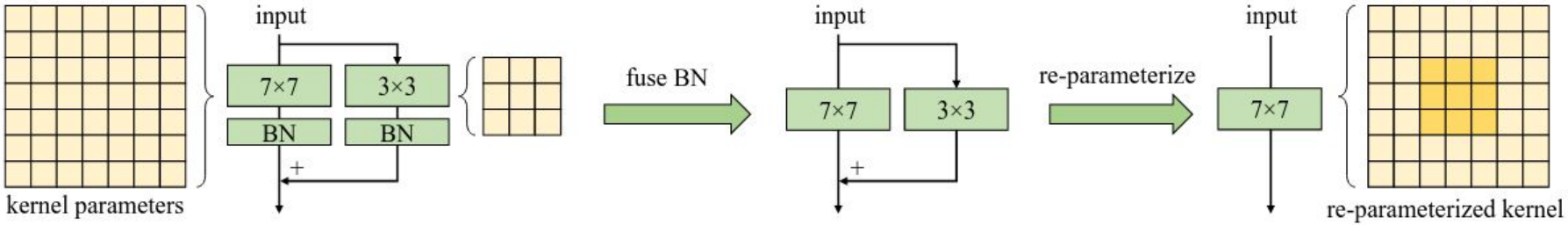
ZerO Initialization: Initializing Neural Networks with only Zeros and Ones





ZerO Initialization: Initializing Neural Networks with only Zeros and Ones

>>> Reparametrization



Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs