Sparsity for efficient LLM inference

Kai Sheng Tai

CIS 7000: Large Language Models (Fall 2024)

Today's lecture

- What we won't cover:
  - Sparsity in ML and statistics more broadly
  - Long history:

    Lasso (Tibshirani, 1996),
    Compressed sensing (Candes, Romberg, Tao, 2006; Donoho 2006),
    etc.

- What we will cover:
  - Sparsity as applied to LLMs
    (in particular, LLM inference)

What do we mean by “sparsity”?

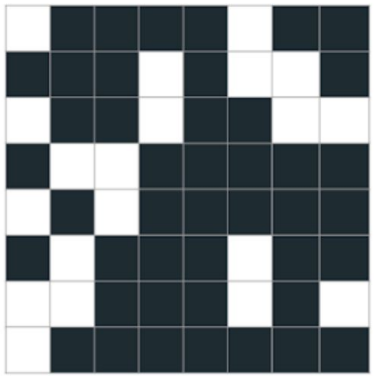
- The *sparsity* of a vector/matrix/tensor is the fraction of its entries that are exactly zero.

- A vector/matrix/tensor with a “large” fraction of zero entries is said to be *sparse*.

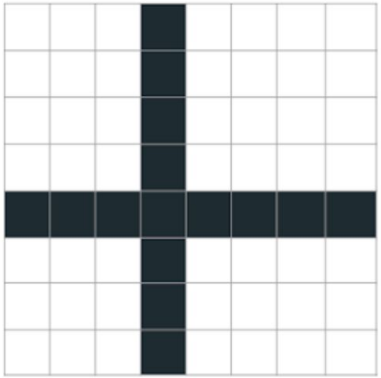
  Otherwise, it is said to be *dense*.

Sparsity patterns

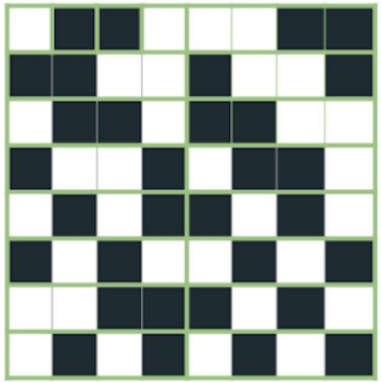
- *Sparsity pattern*: the arrangement of zeros in a sparse matrix



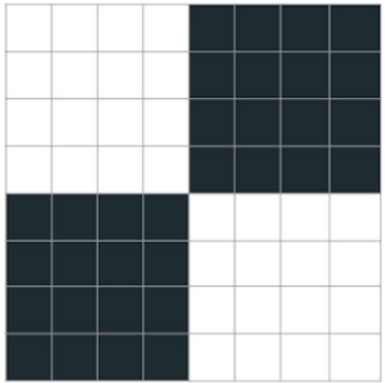
unstructured



row/column/
”channelwise”



N:M/
“semi-structured”



block

Fig: Elhoushi et al. (2024). Speeding up ViTs using Block Sparsity. https://pytorch.org/blog/speeding-up-vits/

How does sparsity help?

- Compute cost
  - sparse matrix multiplication requires fewer FLOPs

- Storage cost
  - sparse matrices can be encoded more compactly, e.g., tuple of (nonzero vals, bitmap of nonzero locs)
  - can improve computational performance if bounded by accelerator memory bandwidth

- Sparsity realizes a quality-performance tradeoff:
  - we typically get reduced quality at higher sparsities

Practical speedups depend on hardware & kernel support

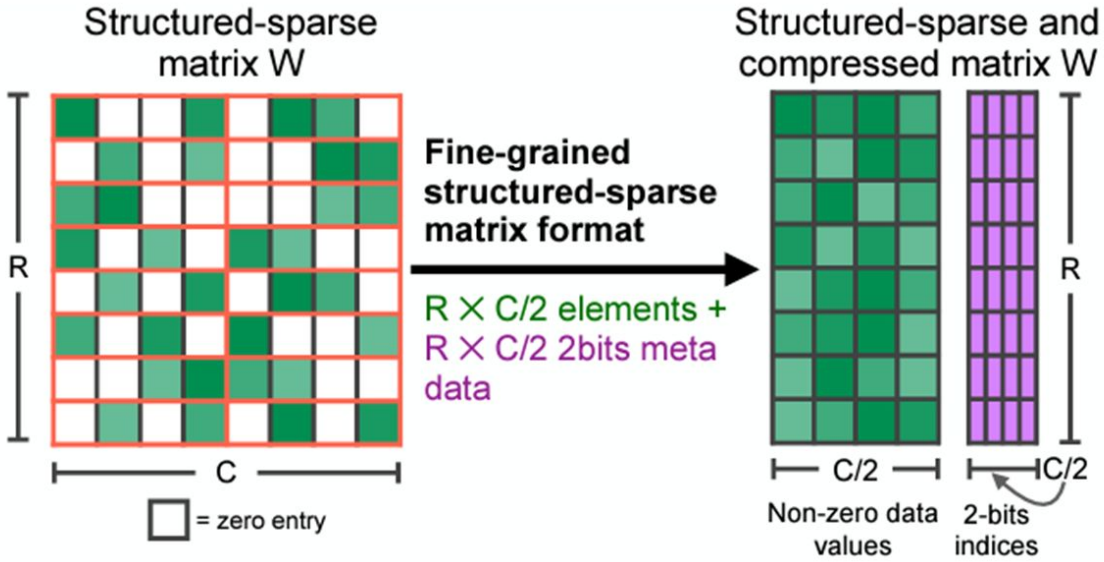
- We use hardware-specific compute kernels to run matmuls efficiently

- Some structured sparsity patterns are compatible with standard dense kernels.
  Examples:
  - remove all-zero rows from a row-sparse matrix
  - remove subnetworks with all-zero parameters from model

- Other sparsity patterns require specialized kernels.
  Examples (hardware → supported sparsity patterns):
  - x86 CPU (Intel MKL) unstructured
  - GPU (NVIDIA cuSPARSE) block, 2:4
  - Cerebras CS-2 unstructured

Practical speedups depend on hardware & kernel support

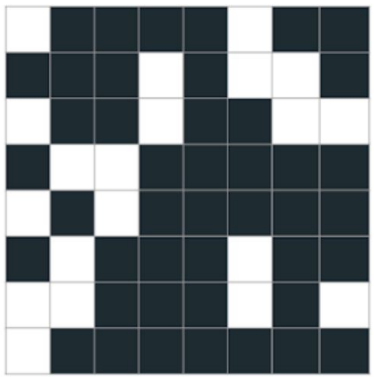
- Example of a specialized kernel: cuSPARSELt 2:4 sparsity



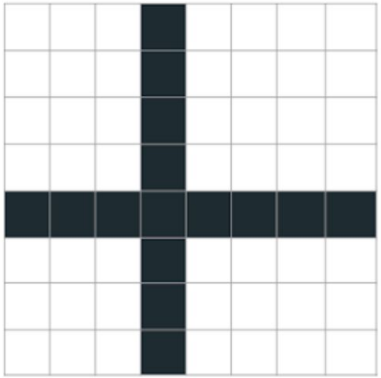
Bai & Li (2023). Structured Sparsity in the NVIDIA Ampere Architecture and Applications in Search Engines. https://developer.nvidia.com/blog/structured-sparsity-in-the-nvidia-ampere-architecture-and-applications-in-search-engines/

Sparsity patterns

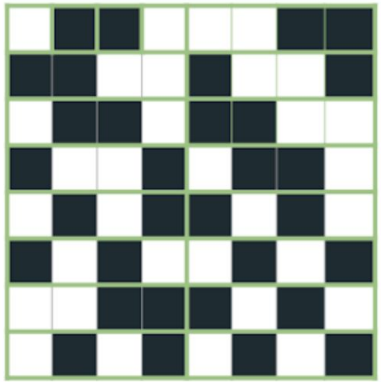
- *Sparsity pattern*: the arrangement of zeros in a sparse matrix



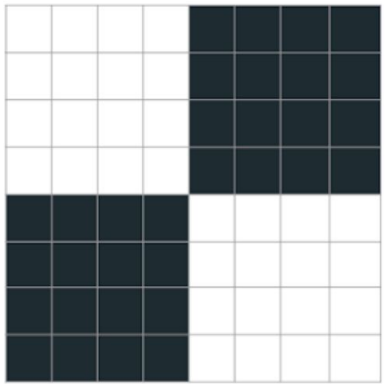
unstructured



row/column/
”channelwise”



N:M/
“semi-structured”



block

Fig: Elhoushi et al. (2024). Speeding up ViTs using Block Sparsity. https://pytorch.org/blog/speeding-up-vits/

Sparsity patterns can be static or dynamic

- *Static* sparsity is fixed for all inputs.
  Dynamic sparsity changes for each input.

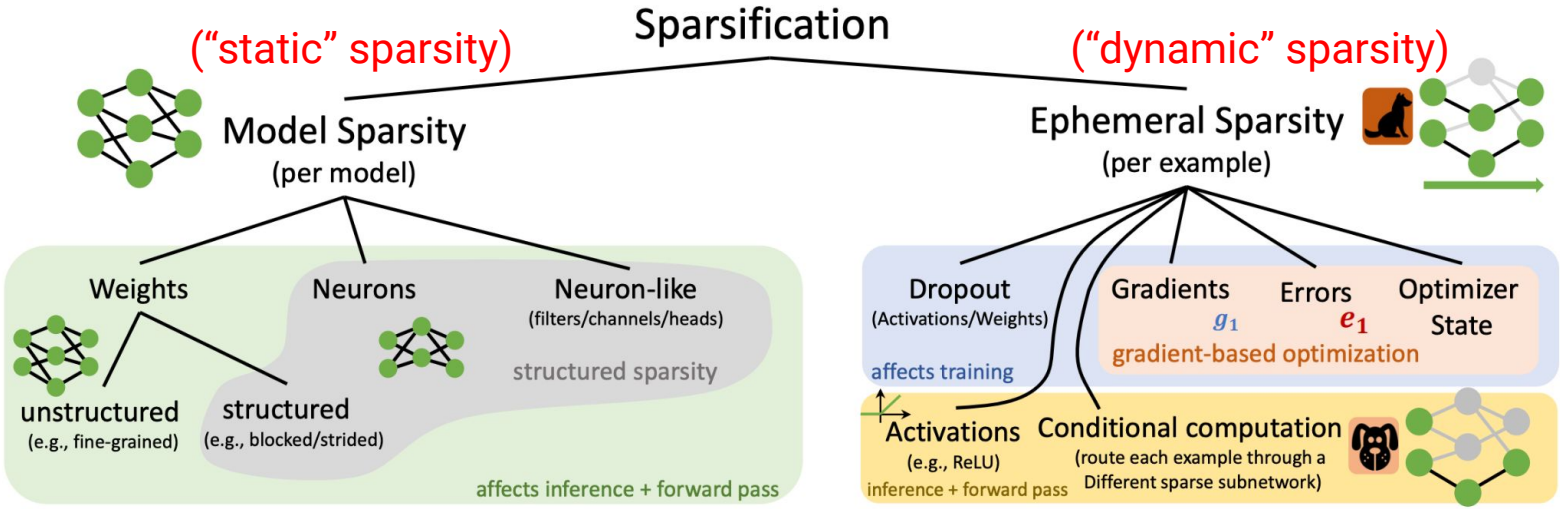


Fig: Hoefler et al. (2021). Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. JMLR.

Today’s topics

- Static sparsity
  - weight pruning:
    cut down a large LLM to a smaller LLM

- Dynamic sparsity
  - mixture-of-experts models:
    activate subnetworks in an input-dependent way

  - KV cache sparsification:
    keep only a subset of past K/V activations

Today’s topics

- Static sparsity
  - weight pruning:
    cut down a large LLM to a smaller LLM

- Dynamic sparsity
  - mixture-of-experts models:
    activate subnetworks in an input-dependent way

  - KV cache sparsification:
    keep only a subset of past K/V activations

A formalization of the weight pruning problem

Given:

- parameters θ
- loss function $L(\theta)$

A formalization of the weight pruning problem

Given:

- parameters θ
- loss function $L(\theta)$
- locally optimal solution θ^*

A formalization of the weight pruning problem

Given:

- parameters θ
- loss function $L(\theta)$
- locally optimal solution θ^*

"post-training sparsification"

A formalization of the weight pruning problem

Given:

- parameters θ
- loss function $L(\theta)$
- locally optimal solution θ^*

"post-training sparsification"

Optimize:

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \text{ s.t. } (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

A formalization of the weight pruning problem

Given:

- parameters θ
- loss function $L(\theta)$
- locally optimal solution θ^*

"post-training sparsification"

Optimize:

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \text{ s.t. } (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

perturbed parameters

A formalization of the weight pruning problem

Given:

- parameters θ
- loss function $L(\theta)$
- locally optimal solution θ^*

"post-training sparsification"

Optimize:

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \text{ s.t. } (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

perturbed parameters

Since solving this combinatorial optimization problem is NP-hard, we turn to tractable approximations

A simple heuristic: magnitude pruning

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \;\; \text{s.t.} \;\; (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

Procedure:

1. Identify weights with the smallest magnitude
2. Set these weights to zero
3. Retrain model, keeping pruned weights at zero
4. (Optional) Repeat until desired sparsity reached

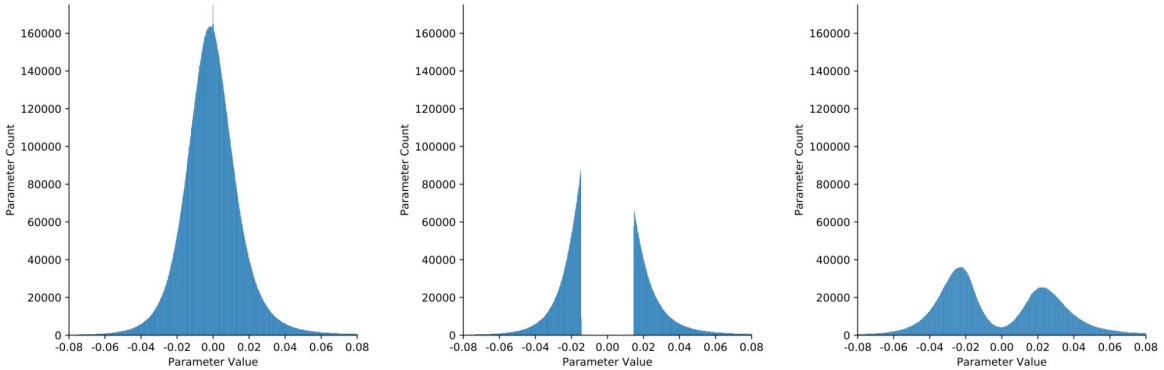


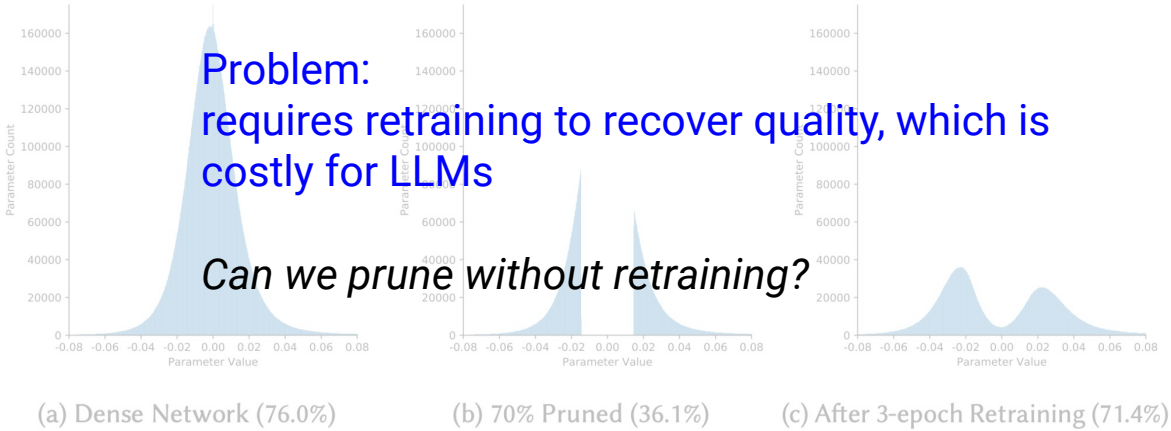
Fig: Hoefler et al. (2021). Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. JMLR.

A simple heuristic: magnitude pruning

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \;\; \text{s.t.} \;\; (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

Procedure:

1. Identify weights with the smallest magnitude
2. Set these weights to zero
3. Retrain model, keeping pruned weights at zero
4. (Optional) Repeat until desired sparsity reached



Problem:
requires retraining to recover quality, which is costly for LLMs

Can we prune without retraining?

Fig: Hoefler et al. (2021). Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. JMLR.

A greedy heuristic: prune weights one at a time

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \;\; \text{s.t.} \;\; (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

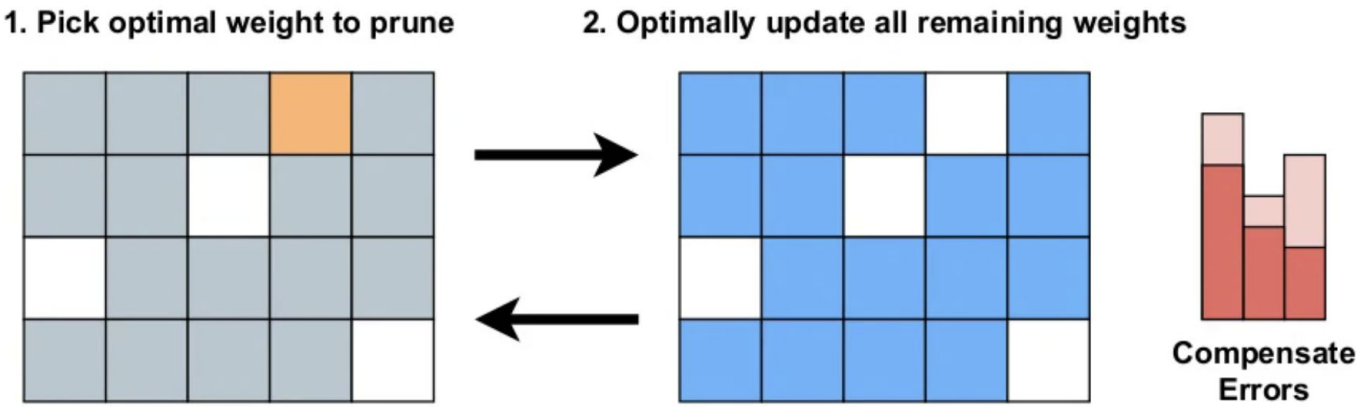


Fig: Frantar et al. (2022). Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning. ICML.

A greedy heuristic: prune weights one at a time

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \;\text{ s.t. }\; (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

Taylor expand to second order:

$$L(\theta^* + \delta\theta) - L(\theta^*) \approx g^T \delta\theta + \frac{1}{2}\delta\theta^T H\, \delta\theta$$

= 0 by assumption that θ* is local minimum

A greedy heuristic: prune weights one at a time

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \;\text{ s.t. }\; (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

Taylor expand to second order:

$$L(\theta^* + \delta\theta) - L(\theta^*) \approx g^T \delta\theta + \frac{1}{2}\delta\theta^T H\, \delta\theta$$

= 0 by assumption that θ* is local minimum

Solve to estimate perturbation that sets *i*th weight to zero:

$$\delta\theta^{(i)} = -\frac{\theta_i}{[H^{-1}]_{ii}}[H^{-1}]_{\cdot i}$$

perturbation to *all* parameters that zeroes out *i*th weight

$$\delta L^{(i)} = \frac{\theta_i^2}{[H^{-1}]_{ii}}$$

"Saliency": Use to select which weight to prune

increase in loss from zeroing out *i*th weight + adjusting remaining parameters

Second order derivatives for network pruning: Optimal Brain Surgeon

Babak Hassibi* and **David G. Stork**
Ricoh California Research Center
2882 Sand Hill Road, Suite 115
Menlo Park, CA 94025-7022
stork@crc.ricoh.com
and
* Department of Electrical Engineering
Stanford University
Stanford, CA 94305

Abstract

We investigate the use of information from *all* second order derivatives of the error function to perform network pruning (i.e., removing unimportant weights from a trained network) in order to improve generalization, simplify networks, reduce hardware or storage requirements, increase the speed of further training, and in some cases enable rule extraction. Our method, Optimal Brain Surgeon (OBS), is significantly better than magnitude-based methods and Optimal Brain Damage [Le Cun, Denker and Solla, 1990], which often remove the wrong weights. OBS permits the pruning of more weights than other methods (for the same error on the training set), and thus yields better generalization on test data. Crucial to OBS is a recursion relation for calculating the inverse Hessian matrix $\mathbf{H}^{-1}$ from training data and structural information of the net. OBS permits a 90%, a 76%, and a 62% reduction in weights over backpropagation with weight decay on three benchmark MONK's problems [Thrun et al., 1991]. Of OBS, Optimal Brain Damage, and magnitude-based methods, only OBS deletes the correct weights from a trained XOR network in every case. Finally, whereas Sejnowski and Rosenberg [1987] used 18,000 weights in their NETtalk network, we used OBS to prune a network to just 1560 weights, yielding better generalization.

NeurIPS
(then NIPS)
1992

Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

ABSTRACT

We have used information-theoretic ideas to derive a class of practical and nearly optimal schemes for adapting the size of a neural network. By removing unimportant weights from a network, several improvements can be expected: better generalization, fewer training examples required, and improved speed of learning and/or classification. The basic idea is to use second-derivative information to make a tradeoff between network complexity and training set error. Experiments confirm the usefulness of the methods on a real-world application.

Predecessor to Optimal Brain Surgeon: assumes Hessian is diagonal

NeurIPS
(then NIPS)
1989

A greedy heuristic: prune weights one at a time

$$\min_{\delta\theta} L(\theta^* + \delta\theta) \;\text{ s.t. }\; (\theta^* + \delta\theta) \text{ is } k\text{-sparse}$$

Taylor expand to second order:

$$L(\theta^* + \delta\theta) - L(\theta^*) \approx g^T \delta\theta + \frac{1}{2}\delta\theta^T H\, \delta\theta$$

= 0 by assumption that θ* is local minimum

Solve to estimate perturbation that sets *i*th weight to zero:

$$\delta\theta^{(i)} = -\frac{\theta_i}{[H^{-1}]_{ii}}[H^{-1}]_{\cdot i}$$

perturbation to *all* parameters that zeroes out *i*th weight

$$\delta L^{(i)} = \frac{\theta_i^2}{[H^{-1}]_{ii}}$$

"Saliency": Use to select which weight to prune

increase in loss from zeroing out *i*th weight + adjusting remaining parameters

A greedy heuristic: prune weights one at a time

Problems:

- Hessian inverses are expensive to compute (O(d^3))
- Need to update H^{-1} after each step to account for pruned weight

How do we scale Optimal Brain Surgeon (OBS) to LLM-size models?

Solve to estimate perturbation that sets *i*th weight to zero:

$$\delta\theta^{(i)} = -\frac{\theta_i}{[H^{-1}]_{ii}}[H^{-1}]_{\cdot i}$$

perturbation to *all* parameters that zeroes out *i*th weight

$$\delta L^{(i)} = \frac{\theta_i^2}{[H^{-1}]_{ii}}$$

increase in loss from zeroing out *i*th weight + adjusting remaining parameters

SparseGPT: scaling up OBS-based pruning

SparseGPT: scaling up OBS-based pruning

Main tricks for scaling up OBS:

- Prune *layer-by-layer* instead of entire model, minimizing the Euclidean loss

$$\frac{1}{2}\mathbb{E}_x\|W^*x - (W^* + \delta W)x\|_2^2 = \frac{1}{2}\mathbb{E}_x\|\delta W x\|_2^2$$

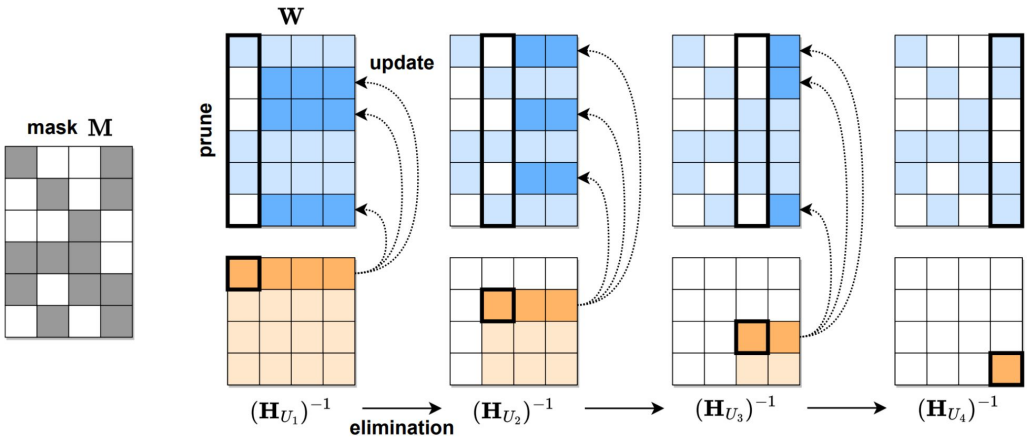
SparseGPT: scaling up OBS-based pruning

Main tricks for scaling up OBS:

- Prune *layer-by-layer* instead of entire model, minimizing the Euclidean loss

$$\frac{1}{2}\mathbb{E}_x\|W^*x - (W^* + \delta W)x\|_2^2 = \frac{1}{2}\mathbb{E}_x\|\delta W x\|_2^2$$

- Pay the $O(d^3)$ cost of Hessian inversion *only once per weight matrix*, sharing the Hessian for all rows by restricting the set of "loss compensating" weights



Frantar & Alistarh (2023). SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. ICML.

SparseGPT: scaling up OBS-based pruning

- *Scalability*: pruned OPT-175B in 4 hours on single A100-80GB
- *Calibration data*: 128 2048-token sequences from C4 dataset

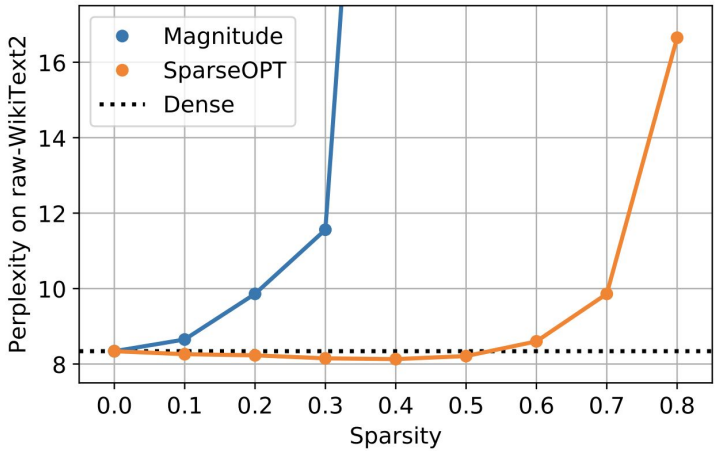


Figure 1. Sparsity-vs-perplexity comparison of `SparseGPT` against magnitude pruning on OPT-175B, when pruning to different *uniform* per-layer sparsities.

Frantar & Alistarh (2023). SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. ICML.

SparseGPT: scaling up OBS-based pruning

- Applied to unstructured and semi-structured sparsity patterns

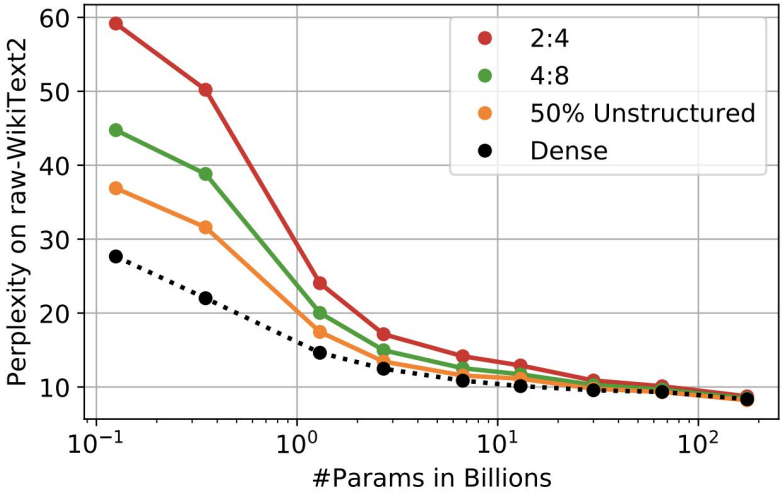


Figure 2. Perplexity vs. model and sparsity type when compressing the entire OPT model family (135M, 350M, ..., 66B, 175B) to different sparsity patterns using `SparseGPT`.

Frantar & Alistarh (2023). SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. ICML.

Recap of post-training sparsification so far

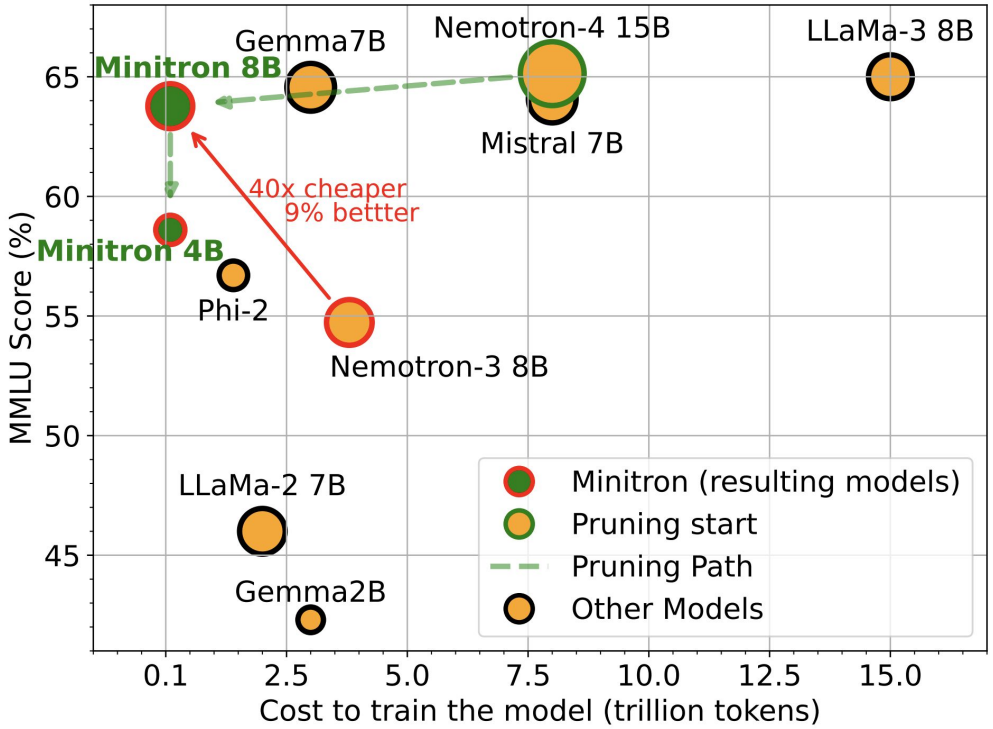
- Combinatorial optimization: intractable, so introduce greedy heuristics
- Magnitude pruning: simple but requires iterative retraining
- Optimal Brain Surgeon (OBS): optimal single-parameter updates under quadratic model of loss; requires expensive Hessian inversions
- SparseGPT: scalable approximation of OBS; local parameter updates requiring fewer Hessian inversions

Recap of post-training sparsification so far

- Combinatorial optimization: intractable, so introduce greedy heuristics
- Magnitude pruning: simple but requires iterative retraining
- Optimal Brain Surgeon (OBS): optimal single-parameter updates under quadratic model of loss; requires expensive Hessian inversions
- SparseGPT: scalable approximation of OBS; local parameter updates requiring fewer Hessian inversions

What about structured sparsity?

Minitron: Structured pruning to obtain smaller dense models

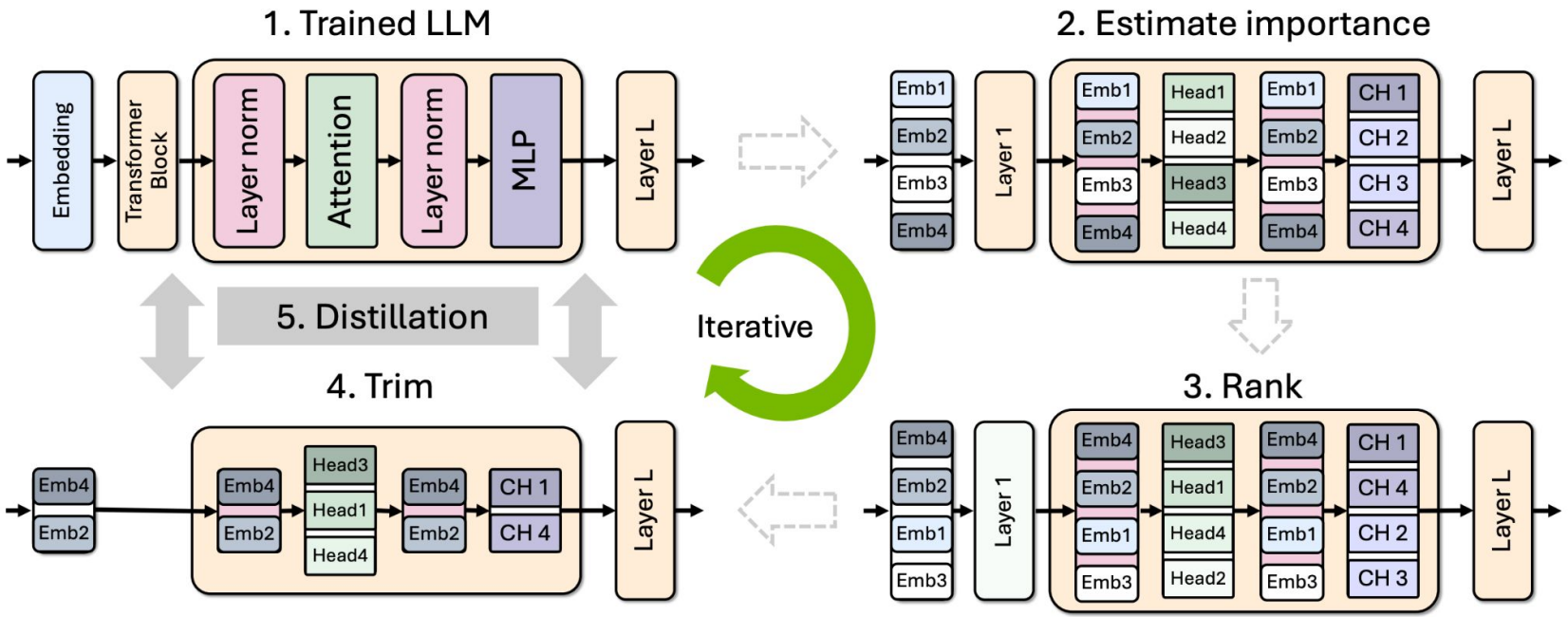


Minitron-8B and -4B: created by pruning *Nemotron-4 15B* and applying a lightweight finetuning process.

These models outperform *Nemotron-3 8B*, which was trained “from scratch”.

Muralidharan et al. (2024). Compact Language Models via Pruning and Knowledge Distillation. NeurIPS.

`Minitron`: Structured pruning to obtain smaller dense models



`Minitron`: Structured pruning to obtain smaller dense models

Procedure:

- Very simple heuristic for scoring neurons/attention heads/layers:
  Use a small calibration dataset to measure mean/average norm of output activations

- Remove lowest-scoring neurons/attention heads/layers and finetune network for a few hundred iterations

Minitron: Structured pruning to obtain smaller dense models

Experiments:

- Base model: 15B parameter LLM trained on 8T tokens
- Calibration data: 1024 sequences
- Retraining data: 1.8B tokens (400 iterations)

			Models						
	Benchmark	**Metric**	**Llama-3**	**Llama-2**	**Mistral**	**Gemma**	**Nemotron-4**	**Nemotron-3**	**MINITRON**
	# Parameters		8B	6.7B	7.3B	8.5B	15.6B	8.5B	8.3B
	# Non-Emb. Params		5.9B	6.4B	7B	7.7B	12.5B	6.4B	6.2B
	# Training Tokens		>15T	2T	8T	6T	8T	3.8T	**94B**
Knowledge,Logic	winogrande (5)	acc	78	74	78.5	78	83.6	75.9	**79.0**
	arc_challenge (25)	acc_norm	58	53	60.3	**61**	58.8	52.8	52.6
	MMLU(5)	acc	**65**	46	64.1	64	66.6	54.7	63.8
	hellaswag(10)	acc_norm	82	79	**83.2**	82	84.6	78.5	80.7
	gsm8k(5)	acc	50	14	37	50	48.5	24.0	**51.3**
	truthfulqa(0)	mc2	44	39	42.6	**45**	40.7	36.5	42.6
	XLSum en (20)(3)	rougeL	31	31	4.80	17	32	30.9	**31.2**
Coding	MBPP(0)	pass@1	**42**	20	38.8	39	38	27.04	35.2
	humaneval (n=20)(0)	pass@1	28	12	28.7	**32**	35.4	20.7	31.6

Muralidharan et al. (2024). Compact Language Models via Pruning and Knowledge Distillation. NeurIPS.

`Minitron`: Structured pruning to obtain smaller dense models

Model	Layers	Hidden Size	Att. Heads	Query Groups	MLP Hidden	Parameters
Nemotron-4 15B	32	6144	48	8	24576	15.6B
Nemotron-3 8B	32	4096	32	32	16384	8.5B
MINITRON 8B	32	4096	48	8	16384	8.27B
MINITRON 4B	32	3072	24	8	9216	4.19B

Empirically, pruning network width outperforms pruning depth

Muralidharan et al. (2024). Compact Language Models via Pruning and Knowledge Distillation. NeurIPS.

Today’s topics

- Static sparsity
  - weight pruning:
    cut down a large LLM to a smaller LLM

- Dynamic sparsity
  - mixture-of-experts models:
    activate subnetworks in an input-dependent way

  - KV cache sparsification:
    keep only a subset of past K/V activations

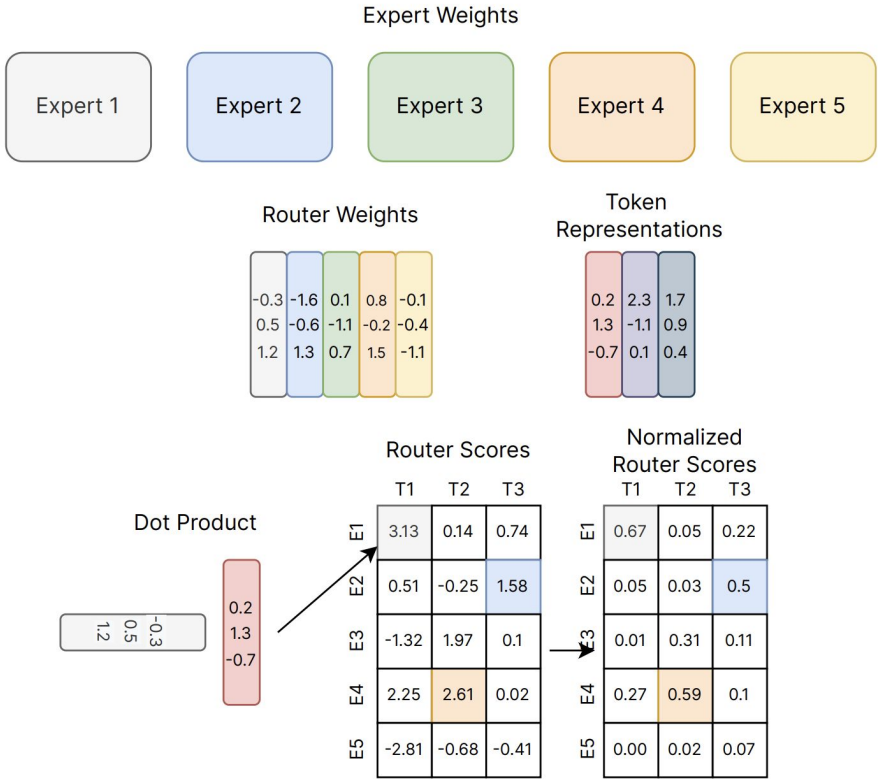
Mixture-of-Experts models

- Idea: Not every part of the network needs to used for every input
  - the model may comprise several “specialist” or “domain-specific” subnetworks
  - we call each such subnetwork an “expert”

- A form of *structured sparsity* where the sparsity pattern is *input-dependent*

- Network needs a mechanism to route computation to the right expert(s): *“expert routing”*

Expert routing via softmax weights



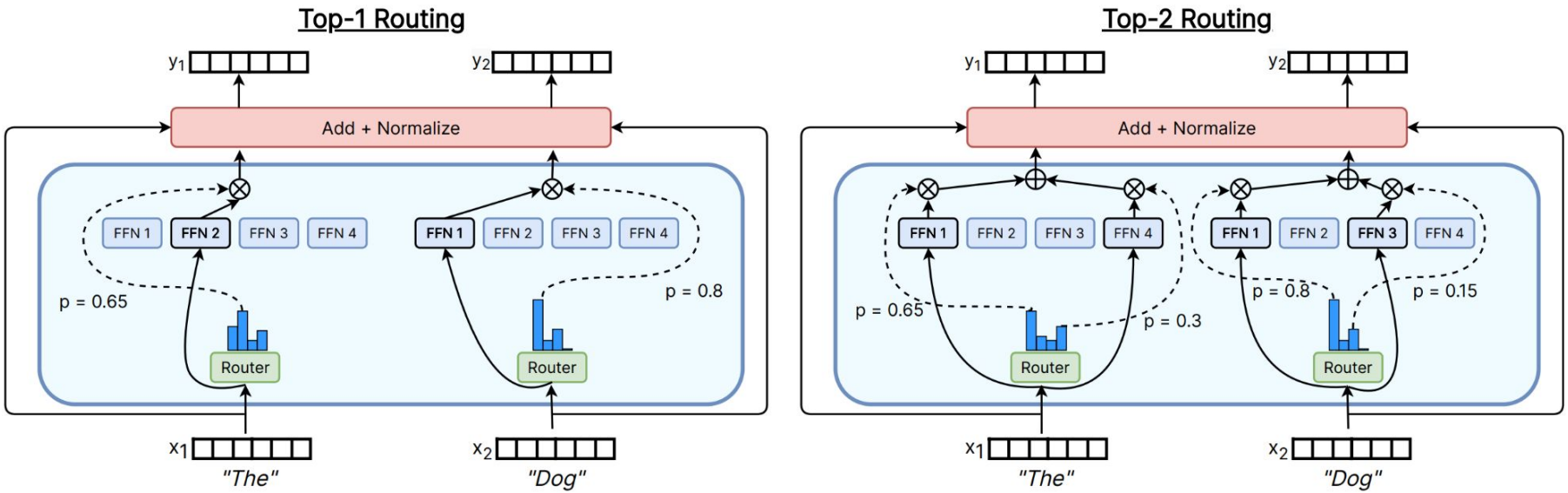
A popular parameterization of expert routing.

- *N* experts $\{E_i\}_{i=1}^{N}$
- Expert logits $h(x) = W_r \cdot x$
- Gating value $p_i(x) = \frac{e^{h(x)_i}}{\sum_j^N e^{h(x)_j}}$
- Output: weighted sum over top-*k* experts

$$y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x)$$

Fedus et al. (2022). A Review of Sparse Expert Models in Deep Learning. arXiv.

Top-*k* routing



Fedus et al. (2022). A Review of Sparse Expert Models in Deep Learning. arXiv.

Communicated by Jeffrey Elman

Adaptive Mixtures of Local Experts

Robert A. Jacobs
Michael I. Jordan
Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139 USA

Steven J. Nowlan
Geoffrey E. Hinton
Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4

We present a new supervised learning procedure for systems composed of many separate networks, each of which learns to handle a subset of the complete set of training cases. The new procedure can be viewed either as a modular version of a multilayer supervised network, or as an associative version of competitive learning. It therefore provides a new link between these two apparently different approaches. We demonstrate that the learning procedure divides up a vowel discrimination task into appropriate subtasks, each of which can be solved by a very simple expert network.

A new instantiation of an old idea…

Jacobs et al. (1991). Adaptive Mixtures of Local Experts. Neural Computation.

Proceedings of 1993 International Joint Conference on Neural Networks

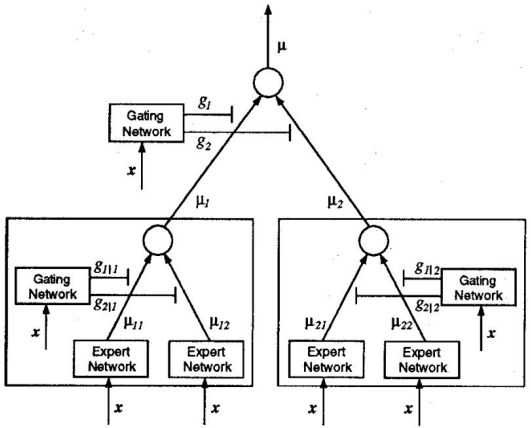
Hierarchical mixtures of experts and the EM algorithm

Michael I. Jordan
Department of Brain and Cognitive Sciences
MIT
Cambridge, MA 02139

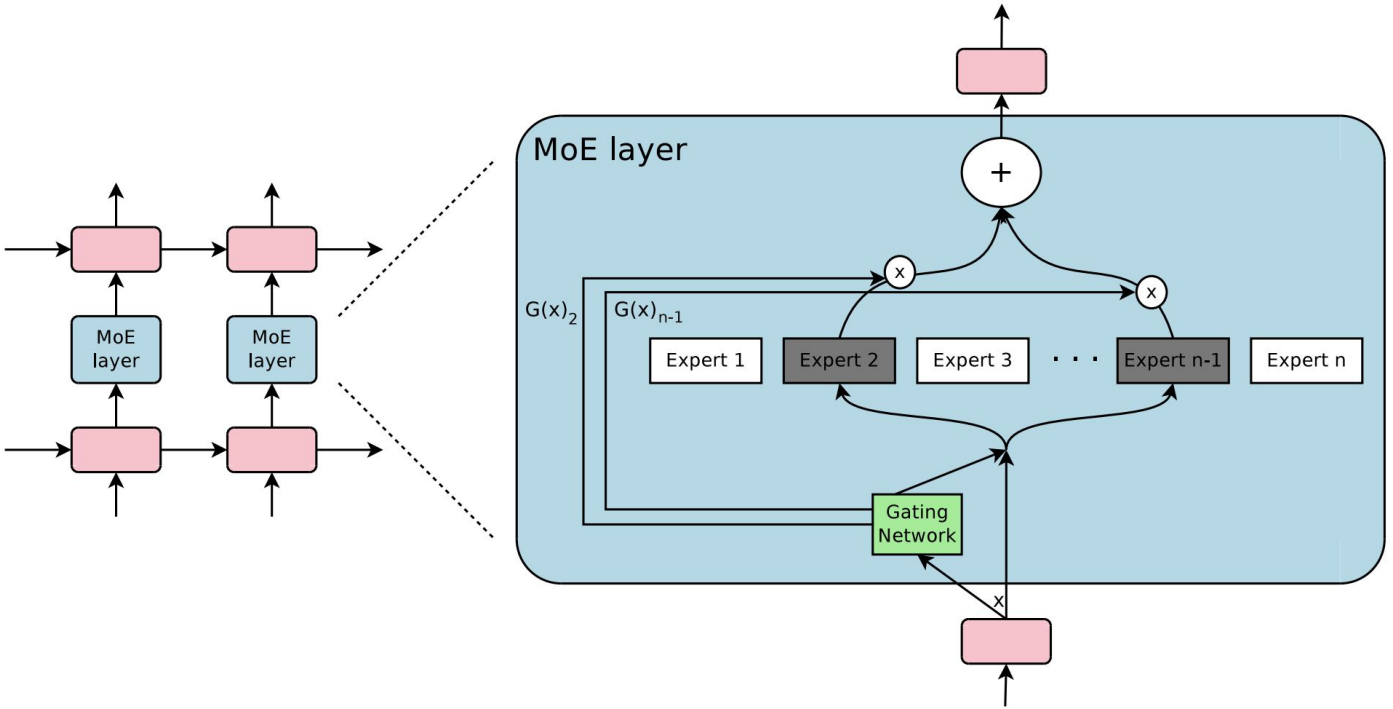
Robert A. Jacobs
Department of Psychology
University of Rochester
Rochester, NY 14627

Abstract

We present a tree-structured architecture for supervised learning. The statistical model underlying the architecture is a hierarchical mixture model in which both the mixture coefficients and the mixture components are generalized linear models (GLIM's). Learning is treated as a maximum likelihood problem; in particular, we present an Expectation-Maximization (EM) algorithm for adjusting the parameters of the architecture. We also develop an on-line learning algorithm in which the parameters are updated incrementally. Comparative simulation results are presented in the robot dynamics domain.

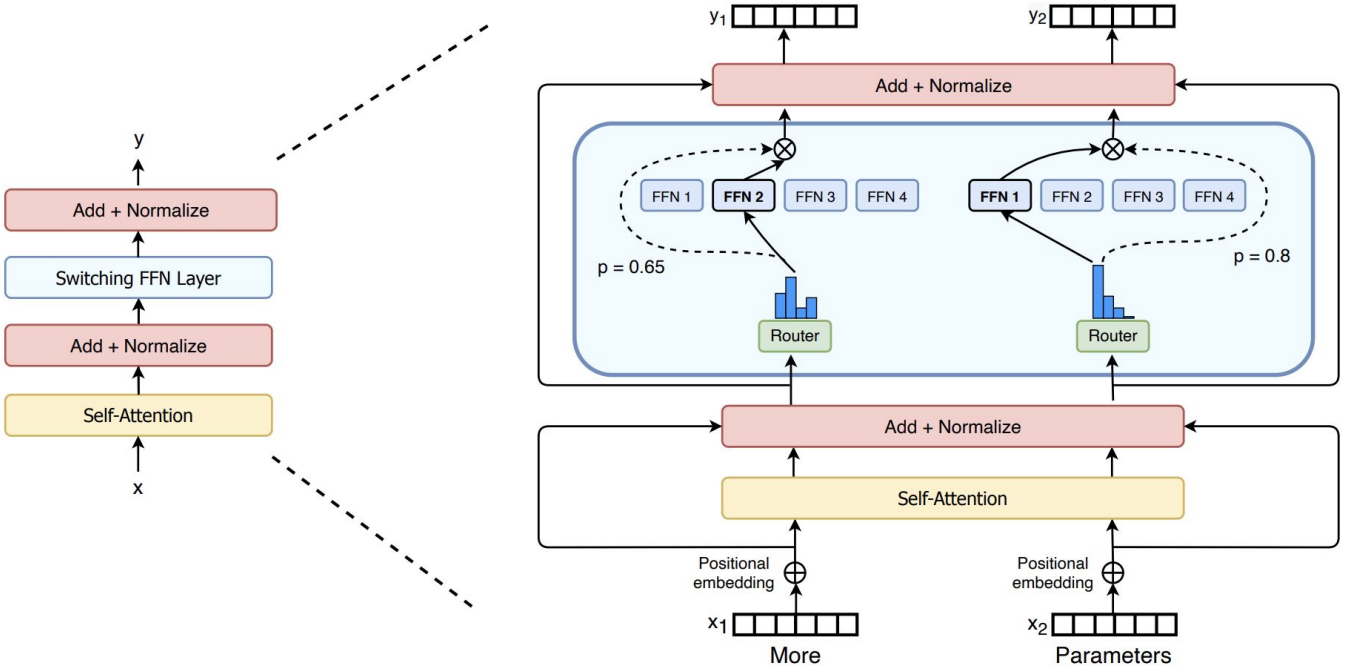


Mixture-of-Experts in RNNs



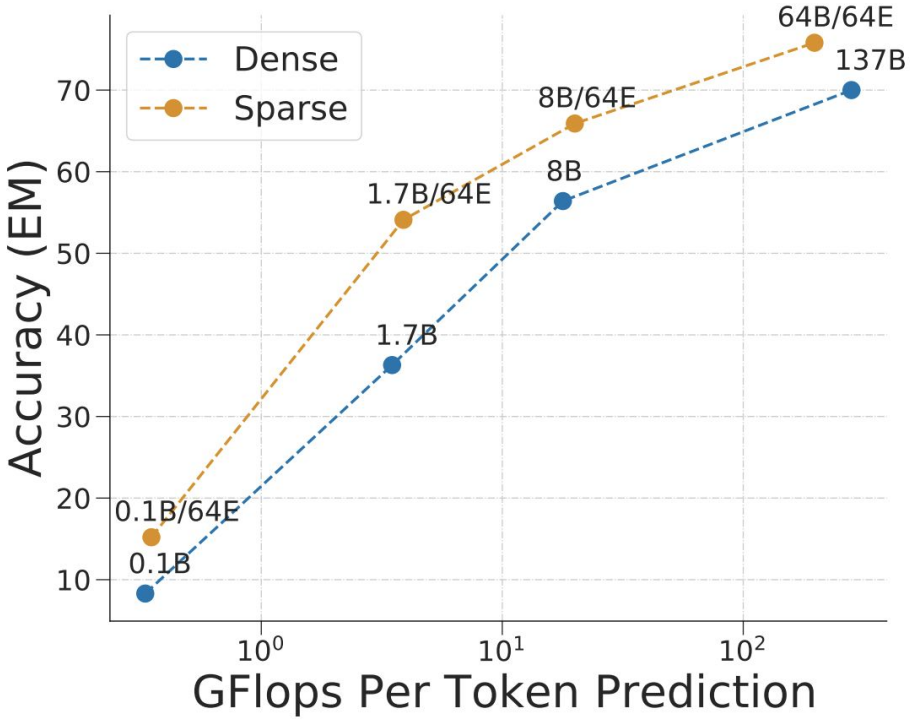
Shazeer et al. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. ICLR.

Mixture-of-Experts in Transformers



Fedus et al. (2022). Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. JMLR.

MoE decouples parameter count from compute cost



Du et al. (2022). GLaM: Efficient scaling of language models with mixture-of-experts. ICML

When should you consider using MoE?

- When there is *plentiful accelerator memory* to host model parameters, but we want to reduce compute/energy usage

- When *sharding model parameters* across several devices during inference
  - natural fit for expert routing
  - need ability to load balance requests to each device

MoE optimization

- Problem: how do we train the router?
  - top-*k* selection results in zero gradients for unselected experts
  - “soft selection” relaxations require running all experts during training, which is expensive

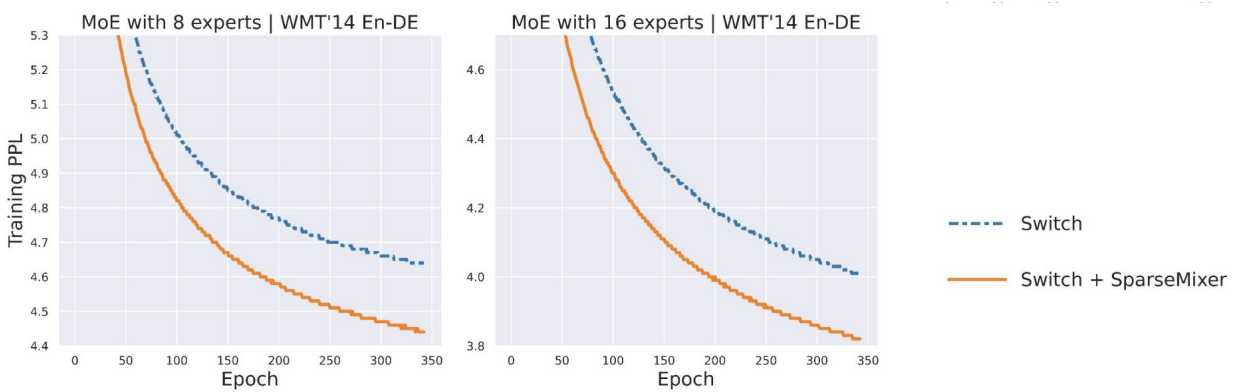
- Heuristic approaches are common in practice:
  - Add “jitter” noise to router logits during training (Fedus et al., 2022)
  - Add batch-wise “load balancing” auxiliary loss (Fedus et al., 2022)
  - “Router z-loss”: Penalize large router logit values (Zoph et al., 2022)

Fedus et al. (2022). Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. JMLR.
Zoph et al. (2022). ST-MoE: Designing Stable and Transferable Sparse Expert Models. arXiv.

MoE optimization: A more principled approach?

- Fundamental problem: gradient estimation
  - Backpropagation requires differentiable computation graphs
  - Sampling discrete variables (e.g., for expert routing) is non-differentiable
  - Need alternative gradient estimators for such situations

- SparseMixer: gradient estimation with sparsely activated experts



Liu et al. (2023). Sparse backpropagation for MoE training. NeurIPS.
Liu et al. (2023). Bridging Discrete and Backpropagation: Straight-Through and Beyond. NeurIPS.

Today's topics

- Static sparsity
  - weight pruning:
    cut down a large LLM to a smaller LLM

- Dynamic sparsity
  - mixture-of-experts models:
    activate subnetworks in an input-dependent way

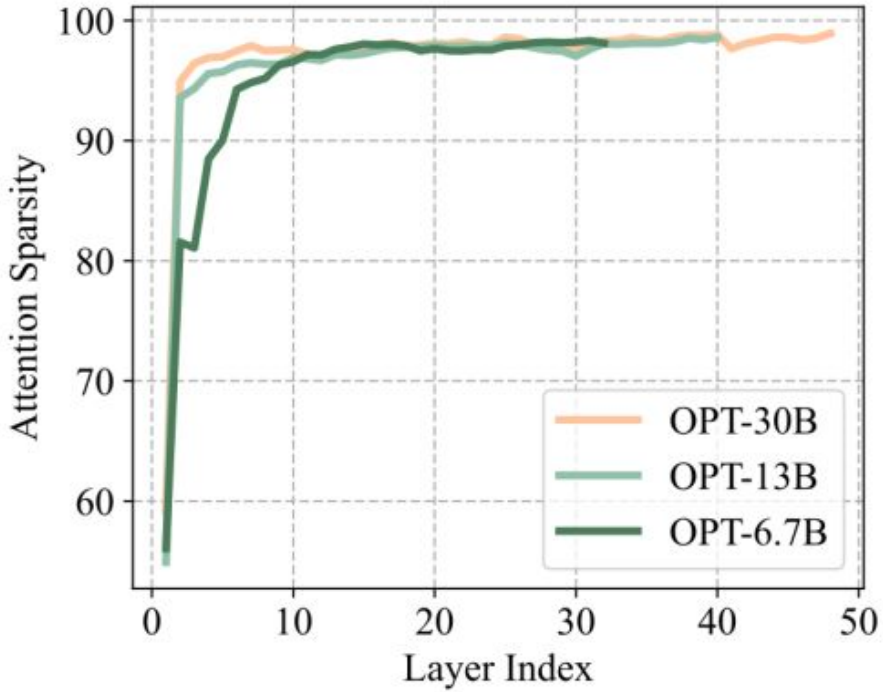
  - KV cache sparsification:
    keep only a subset of past K/V activations

Dynamic activation sparsity: KV Cache Pruning

- For autoregressive generation, we maintain a history of all previous key and value vectors (the “KV cache”) to avoid recomputation
- Memory usage scales linearly with the length of the generated sequence
- We can reduce both memory usage and computational cost by implementing a KV cache eviction policy

Zhang et al. (2023). H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. NeurIPS

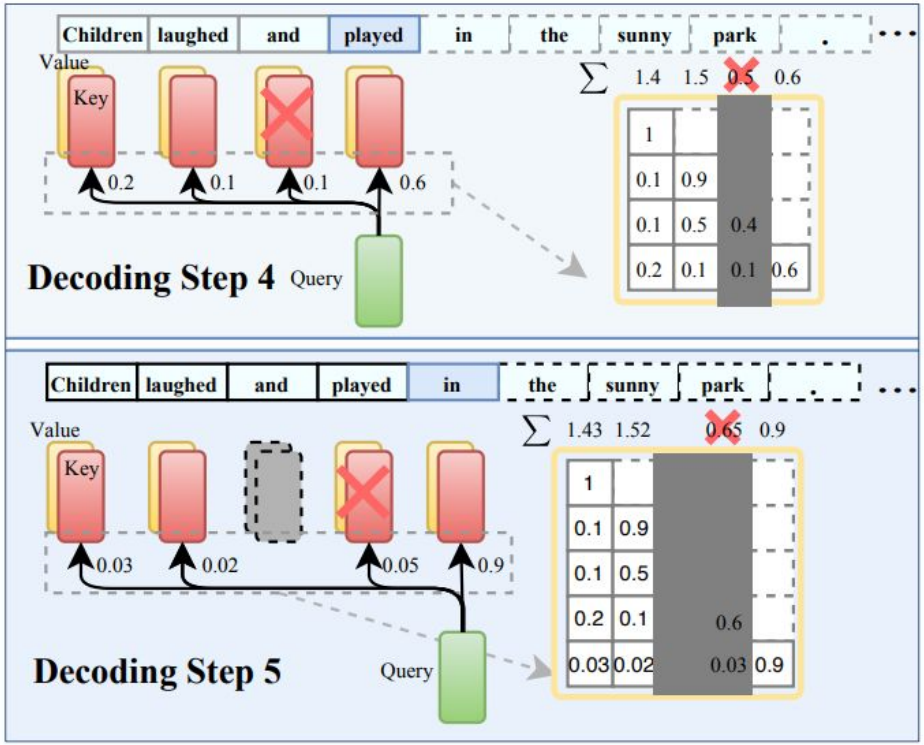
Observation: attention matrices are approximately sparse



- Additionally, the cumulative attention scores for each token follows a power law distribution.

Zhang et al. (2023). H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. NeurIPS

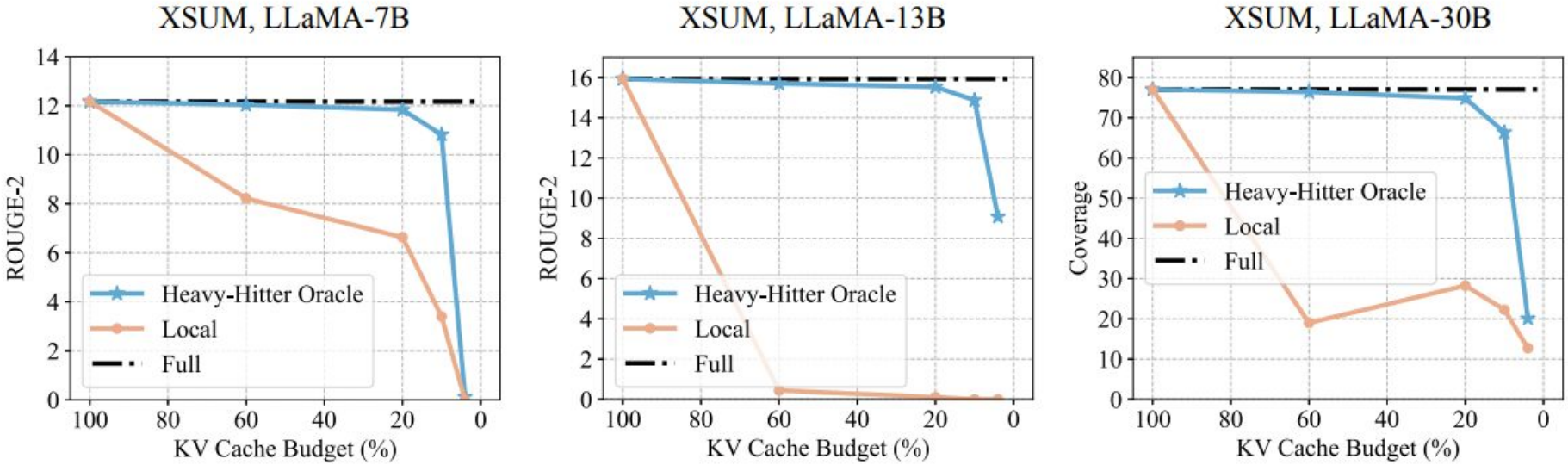
H_2O: Heavy Hitter Oracle



- Greedy heuristic: only keep *k* tokens with highest cumulative attention scores so far.
- Example figure: cache size *k = 3* “and” and “played” are evicted since they have the lowest cumulative scores

Zhang et al. (2023). H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. NeurIPS

H_2O: Heavy Hitter Oracle



- “local” baseline keeps only most recent tokens

Zhang et al. (2023). H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. NeurIPS

Recap

- Static sparsity
  - weight pruning:
    cut down a large LLM to a smaller LLM

- Optimal brain surgeon
- SparseGPT: Fast OBS for LLMs
- Minitron structured pruning

- Dynamic sparsity
  - mixture-of-experts models:
    activate subnetworks in an input-dependent way

- Top-*k* softmax routing
- Optimization issues

  - KV cache sparsification:
    keep only a subset of past K/V activations

- H_2O: evict tokens with lowest cumulative attention