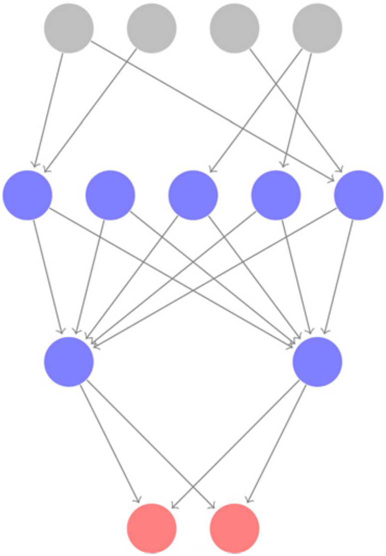
Towards Efficient Prune-Retrain Pipelines

Sparse Model Soups and Parameter-Efficient Retraining

Talk at RIKEN AIP, Tokyo, Japan

Max Zimmer

May 2025



Papers this talk builds upon:

- **Z.**, Spiegel, Pokutta. Sparse Model Soups: A Recipe for Improved Pruning via Model Averaging. ICLR 2024.
- **Z.**, Andoni, Spiegel, Pokutta. PERP: Rethinking the Prune-Retrain Paradigm in the Era of LLMs. arXiv preprint.

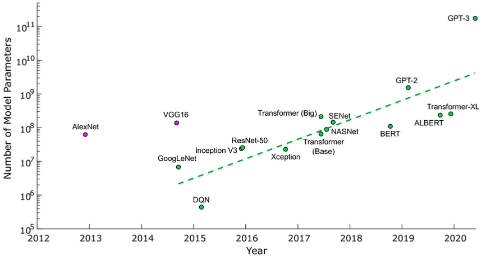
Christoph Spiegel
Zuse Institute Berlin

Sebastian Pokutta
Zuse Institute Berlin

1. Introduction

Why do we need sparsity?

- Neural Networks are exploding in size
- This yields several problems:
  - **Efficiency:** Slow training and inference
  - **Storage:** Not deployable on phones, ...
  - **Costs:** Costly energy demands
    - Training of LLMs: emits as much CO_2 as five cars in their lifetime (Strubell et al., 2019)
    - GPT-4 Training: more than $100M$ USD (according to Altman)



Source: Bernstein et al. (2021)

- **One solution:** *Pruning* - the removal of parameters from the network (LeCun et al., 1989; Han et al., 2015).

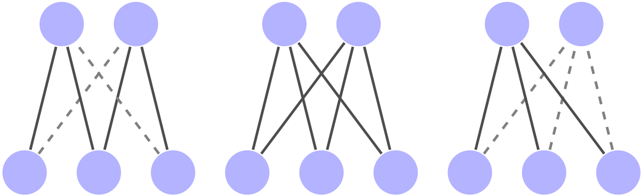
How to decide what to prune?

Idea: introduce *sparsity* into tensors to reduce storage- and compute-demands

Solve: $\min_{\mathcal{W}} \mathcal{L}(\mathcal{W}, \mathcal{D})$ s.t. $\|\mathcal{W}\|_0 \leq k$. $\rightarrow$ *Intractable!*

Better: Remove weights of pretrained model using heuristic, e.g., parameter *magnitude*.

unstructured **dense** **structured**



$$\begin{bmatrix} \times & \times & 0 \\ 0 & \times & \times \end{bmatrix} \quad \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \end{bmatrix} \quad \begin{bmatrix} \times & \times & \times \\ 0 & 0 & 0 \end{bmatrix}$$

A classical pruning approach

Iterative Magnitude Pruning (IMP, Han et al., 2015);
Input: A pretrained network θ.
repeat
 PRUNE a fraction of the lowest-magnitude weights;
 RETRAIN the non-pruned weights for a bit; $\leftarrow$ *costly*
until *the desired sparsity is reached*;

- Problem: Iterative pruning and retraining is *costly*!

$\rightarrow$ Our approach and topic of this talk: *Efficient Prune-Retrain Pipelines*
 - $\rightarrow$ *Sparse Model Soups (SMS)*: Parallelize retraining
 - $\rightarrow$ *Parameter-Efficient Retraining (PERP)*: Make retraining of LLMs feasible

2. Sparse Model Soups (SMS)

Leveraging multiple models for better performance

Given m models $\theta_1, \ldots, \theta_m$, can we construct a better model θ?

Ensembles: *Average the outputs* of m models

$\rightarrow$ Drastically improves generalization performance

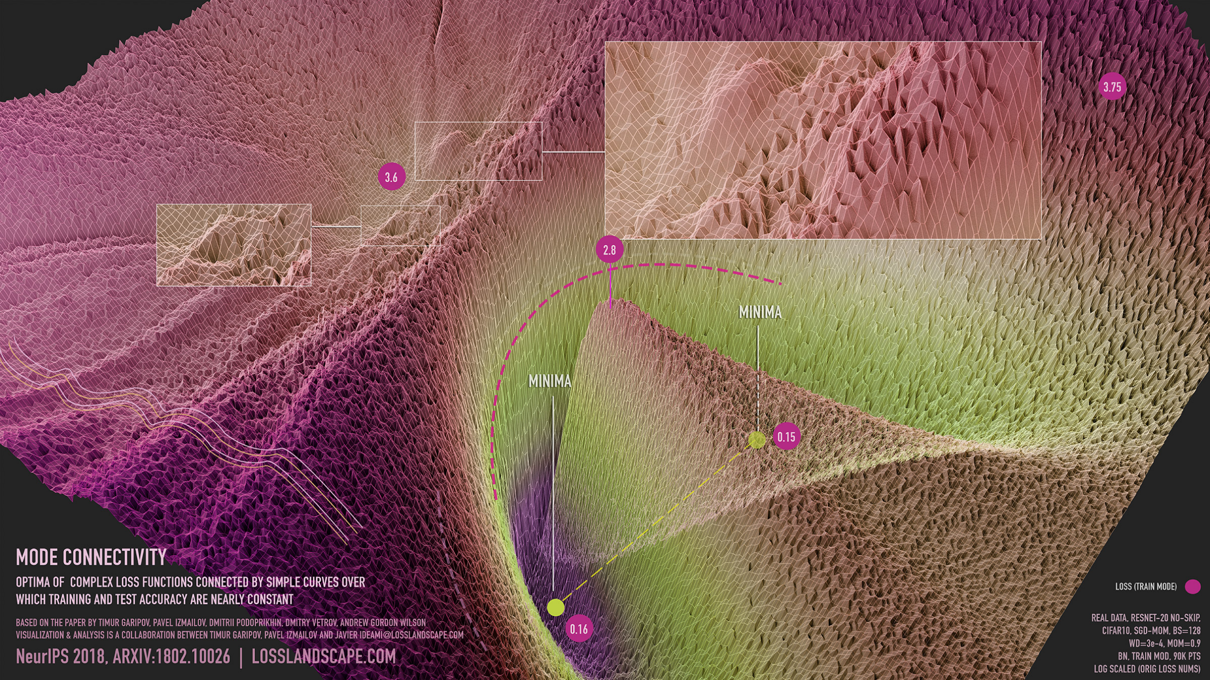
$\rightarrow$ *Problem:* Increases the inference time by a factor of m

Parameter Averaging or Model Soups: *Average the parameters* of m models

$\rightarrow$ New model $\bar{\theta} = \sum_{i \in [m]} \lambda_i \theta_i$ is efficient to use

$\rightarrow$ *Difficulty:*

- Models θ_i must reside in a linearly connected loss basin.
- Even averaging models trained with identical initialization but different seeds can degrade performance compared to individual models (Neyshabur et al., 2020).



3.75
3.6
2.8
MINIMA
MINIMA
0.15
0.16
MODE CONNECTIVITY
OPTIMA OF COMPLEX LOSS FUNCTIONS CONNECTED BY SIMPLE CURVES OVER
WHICH TRAINING AND TEST ACCURACY ARE NEARLY CONSTANT
BASED ON THE PAPER BY TIMUR GARIPOV, PAVEL IZMAILOV, DMITRII PODOPRIKHIN, DMITRY VETROV, ANDREW GORDON WILSON
VISUALIZATION & ANALYSIS IS A COLLABORATION BETWEEN TIMUR GARIPOV, PAVEL IZMAILOV AND JAVIER IDEAMI@LOSSLANDSCAPE.COM
NeurIPS 2018, ARXIV:1802.10026 | LOSSLANDSCAPE.COM
LOSS (TRAIN MODE)
REAL DATA, RESNET-20 NO-SKIP,
CIFAR10, SGD-MOM, BS=128
WD=3e-4, MOM=0.9
BN, TRAIN MOD, 90K PTS
LOG SCALED (ORIG LOSS NUMS)

Motivation

Can we get the benefits of both *weight averaging* and *sparsity*?

- For ensembles: Easy! Just obtain multiple sparse models and average the outputs!

Problem: How to find models that are both sparse and weight-averageable?

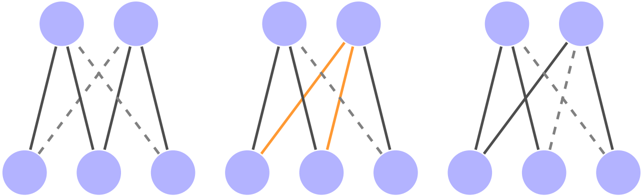
- Ensembles should be as diverse as possible, but what about model soups?

$\rightarrow$ Model Soup candidates should be diverse enough, but not too diverse?

Problem 1: Averaging destroys sparsity

Problem 1: Averaging sparse models may destroy the sparsity pattern!

θ_1 $\qquad$ $\frac{1}{2}(\theta_1 + \theta_2)$ $\qquad$ θ_2



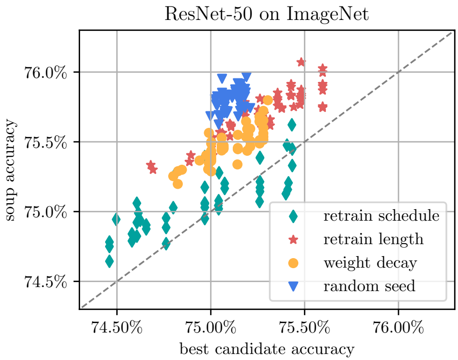
$$\begin{bmatrix} \times & \times & 0 \\ 0 & \times & \times \end{bmatrix} \quad \begin{bmatrix} \times & \times & 0 \\ \times & \times & \times \end{bmatrix} \quad \begin{bmatrix} \times & \times & 0 \\ \times & 0 & \times \end{bmatrix}$$

Problem 2: Finding averageable models

Problem 2: How to obtain models that are averageable?

Idea: Training from the same *pretrained* model keeps models close.

Observation: Prune a pretrained model, retrain copies with *different hyperparameters* $\rightarrow$ **averageable models**.



The recipe

We obtain:

- Sparse models with superior performance
- without destroying the sparsity

Idea: Average models after each prune-retrain cycle to maintain sparsity! $\rightarrow$ **SMS**

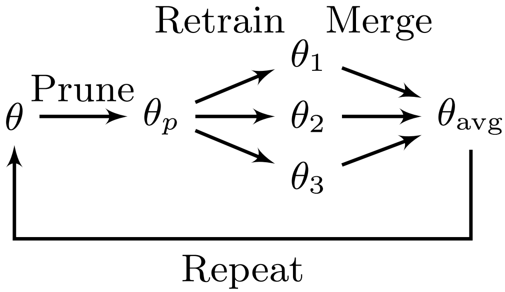


Figure: Sketch for a single phase, $m = 3$.

Comparing SMS against suitable baselines (1)

In each phase, SMS trains m models in parallel for k epochs each.

Suitable baselines:

- IMP: Regular IMP without averaging, i.e., $m = 1$.
- $\text{IMP}_{m\times}$: Extended IMP, where the IMP retraining duration is extended by a factor of m, resulting in $k \cdot m$ retraining epochs per prune-retrain cycle as as many overall epochs as SMS.
- Best candidate: Best accuracy among all averaging candidates.

Comparing SMS against suitable baselines (2)

Table: ResNet-50 on ImageNet: Test accuracies for target sparsity 90% given three IMP cycles.

	Sparsity 53.6% (Phase 1)			Sparsity 78.5% (Phase 2)			Sparsity 90.0% (Phase 3)		
Accuracy of	$m=3$	$m=5$	$m=10$	$m=3$	$m=5$	$m=10$	$m=3$	$m=5$	$m=10$
SMS	**76.74 ±0.20**	**76.89 ±0.18**	**77.01 ±0.05**	**76.04 ±0.21**	**76.30 ±0.13**	**76.49 ±0.12**	**74.53 ±0.04**	**74.82 ±0.08**	**74.96 ±0.16**
best candidate	76.07 ±0.01	76.07 ±0.21	76.14 ±0.18	75.48 ±0.16	75.46 ±0.11	75.70 ±0.03	74.00 ±0.03	74.19 ±0.08	74.25 ±0.13
$\text{IMP}_{m\times}$	76.25 ±0.08	76.21 ±0.14	76.46 ±0.04	75.74 ±0.03	75.87 ±0.11	75.93 ±0.03	74.34 ±0.09	74.56 ±0.24	74.50 ±0.09
IMP		— 75.97 ±0.16 —			— 75.19 ±0.14 —			— 73.59 ±0.04 —	

Key insight: SMS outperforms all baselines.

Back to efficient Prune-Retrain Pipelines:

- SMS outperforms $\text{IMP}_{m\times}$.
- Both use *km* retraining epochs per phase, but SMS is fully parallelizable

$\rightarrow$ SMS needs less retraining wall-time than IMP to achieve the same performance.

Sparse Model Soups: Conclusion

- **Challenge:** Combining sparsity and model averaging
- **Insight:** Models retrained with different hyperparameters are averageable
- **Results:** Better performance than baselines (ID and OOD)

Takeaway: SMS creates a single, sparse, high-performing model in less time than IMP.

3. Parameter-Efficient Retraining after Pruning (PERP)

The Problem with Retraining LLMs

- Simple heuristics like magnitude pruning require *retraining* to recover performance.
- Retraining involves updating **all** remaining parameters.
- For Large Language Models (LLMs), full retraining is demanding:
  - **Memory:** Optimizers require storing parameters, gradients, and optimizer states.
  - **Compute:** Backpropagation through billions of parameters is slow and costly.

→ Narrative: Retraining is **infeasible** for LLMs

→ Growing interest in *retraining-free* methods

Can we make retraining feasible for LLMs?

PERP: Parameter-Efficient Retraining after Pruning

Hypothesis: Retraining *all* parameters is unnecessary.
Retraining a small subset of expressive parameters might suffice.

Idea:

- Update tiny parameter subsets, such as only biases or Layer Normalization (LN) parameters.
- Leverage techniques from *Parameter-Efficient Fine-Tuning (PEFT)* for retraining.

Retraining only

- Biases $\rightarrow \approx 0.03\%$ of parameters
- LN parameters $\rightarrow \approx 0.01\%$ of parameters

is surprisingly effective, even though pruning severely degrades model performance.

A naive idea: Retraining tiny parameter subsets

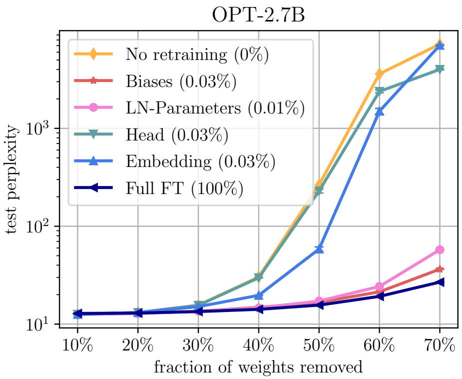


Figure: OPT-2.7B: Perplexity vs. Sparsity. Retraining biases or LN params nearly matches full fine-tuning (FT).

- Retrain only a tiny parameter subset

$\rightarrow$ Reduces memory and compute overhead

$\rightarrow$ Enables retraining huge models (e.g., 30B) on a *single GPU* in minutes.

$\rightarrow$ **Problem:** Gap to full FT at high sparsities

Closing the Gap: Sparsity-Aware LoRA I

A popular PEFT method: *Low-Rank Adaptation (LoRA)*:

- Freeze $W \in \mathbb{R}^{n,m}$, train low-rank matrices $B \in \mathbb{R}^{n,r}, A \in \mathbb{R}^{r,m}, r \ll n, m$.
- New forward: $Wx \to (W + BA)x$
- After finetuning: Merge $W \leftarrow W + BA$

Problem for pruning:

- Merging BA (dense) into W (sparse) destroys sparsity!
- Keeping forward without merging increases inference cost.

Proposed solution:

- *MaskLoRA:* Apply pruning mask M during forward pass $(W + M \odot BA)x$.
  - W is the pruned matrix (i.e., $W = M \odot W$)
  - Keeps sparsity: $W \leftarrow W + M \odot BA$.
  - Integrates sparsity into training

Closing the Gap: Sparsity-Aware LoRA II

Table: OPT-2.7B Zero-Shot accuracy comparison (magnitude pruning). MaskLoRA closes the gap to Full FT, using less than 1% params.

OPT-2.7B (Base Accuracy: 47.81%)

Method	% trainable	30%	40%	50%	60%	70%
Full FT	100%	46.99%	46.20%	45.44%	44.53%	42.44%
MaskLoRA	0.882%	47.25%	46.29%	45.92%	43.92%	41.56%
Biases	0.034%	46.75%	45.66%	45.29%	42.75%	39.49%
LN-Params	0.013%	46.78%	45.48%	44.72%	41.37%	38.32%
No retraining	0.000%	44.99%	42.77%	40.01%	35.34%	32.38%

MaskLoRA largely closes the gap to Full FT, but...
Backpropagating a global loss still requires storing activations for the *entire* model!

Layer-wise Reconstruction I

Observation: Many state-of-the-art *retraining-free* methods (Wanda, SparseGPT) already operate *layer-by-layer*, using calibration data to prune locally.

Idea: Optimize a *local, layer-wise reconstruction error*:

$$\min_{\hat{W}_l} \| W_l X_l - (M_l \odot \hat{W}_l) X_l \|_2^2$$

Problem: For large models, optimizing just one layer can already be infeasible.

Can we apply MaskLoRA in the layer-wise reconstruction setting?

→ More memory efficient (requires activations/gradients for only one layer at a time)

→ Enhance existing retraining-free methods like Wanda and SparseGPT

Layer-wise Reconstruction II

Table: OPT Zero-Shot Accuracy using MaskLoRA-reconstruction, 50% unstructured sparsity.

		OPT			
Method	Reconstruction	2.7B	6.7B	13B	30B
Magnitude	✗	40.07%	35.54%	33.80%	36.39%
Magnitude	✓	**45.14%**	**48.99%**	**50.41%**	**51.81%**
Wanda	✗	42.63%	47.14%	50.34%	53.15%
Wanda	✓	**46.47%**	**49.81%**	**51.65%**	**54.00%**
SparseGPT	✗	46.53%	50.26%	**51.93%**	54.01%
SparseGPT	✓	**46.62%**	**50.42%**	51.92%	**54.33%**

MaskLoRA reconstruction

- significantly boosts existing pruning methods.
- makes simple magnitude pruning competitive.

PERP: Conclusion

- **Challenge:** Full retraining after pruning is infeasible for LLMs.
- **Insight:** Retraining a tiny fraction of parameters (e.g., biases) is highly effective and efficient.
- **Results:**
  - MaskLoRA closes the gap to full FT while preserving sparsity upon merging.
  - Layer-wise MaskLoRA reconstruction improves memory efficiency and enhances existing pruning methods.

Takeaway: PERP makes retraining of large models feasible again.

ZUSE INSTITUTE BERLIN

Thank you for your attention!