Exploiting low-rank geometry in deep learning

Francesco Tudisco

In this talk

Recent work on theory and algorithms for reducing model-order in deep learning via low-rank factorizations

Thanks to several collaborators:

- P. Deidda; E. Zangrando (GSSI, Italy IT)
- S. Schotthoefer (Oak Ridge National Lab, USA US)
- G. Ceruti (Univ of Innsbruck, Austria AT)
- J. Kusch (Univ of Oslo, Norway NO)
- S. Brugipaglia (Concordia Univ, Canada CA)

Outline

- Introduction/motivation
- Dynamical low-rank training (DLRT) algorithm and theory
- Experimental evaluation

Fast growth of model size

Model	# trainable parameters
ResNet-50	$O(10)$ million
BERT	$O(10^2)$ million
GPT-2	$O(10^3)$ million
DALL-E	$O(10^4)$ million
LLaMA	$O(10^5)$ million
Gemini	$O(10^6)$ million

Prohibitive memory, inference, training, fine-tuning cost

Storage, inference, training costs

- Large language models typically require hundreds of gigabytes to load and expensive GPUs to be used, which places them outside the range of most consumer electronics [Wikipedia]
- *Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos* — 30 epochs training took 9 days on 720 V100 GPUs
  AWS price/h smallest V100 (p3) = $3 → 3*9*24*720~$470K
  [Baker et al., NeurIPS 2022]
- In the USA market alone, AI energy demand is expected to reach 35GW by 2030 (up from 17 GW in 2022), the equivalent of powering 26M homes [McKinsey]

Parameter reduction

reduce memory and computation footprints while retaining performance

A variety of approaches in DL literature

- Neural Architecture Search (NAS)
- Distillation
- Quantization
- Graph sparsification
- Weight pruning
- Layer factorization

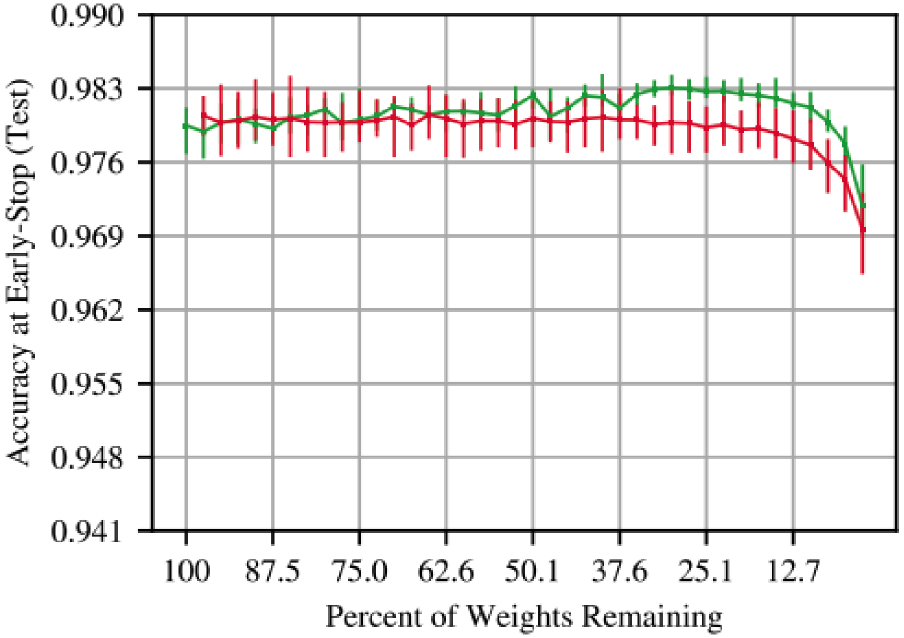
Compress after training

Train	Train the full network
Compress	Choose how to reduce model size
Adjust	Fine-tune the compressed model

Lottery ticket hypothesis

The lottery ticket hypothesis: finding sparse, trainable neural networks. J Frankle, M Carbin, ICLR 2019

A randomly-initialized dense NN contains a subnetwork that, when trained in isolation and for the same number of iterations, can match the accuracy of the original full network

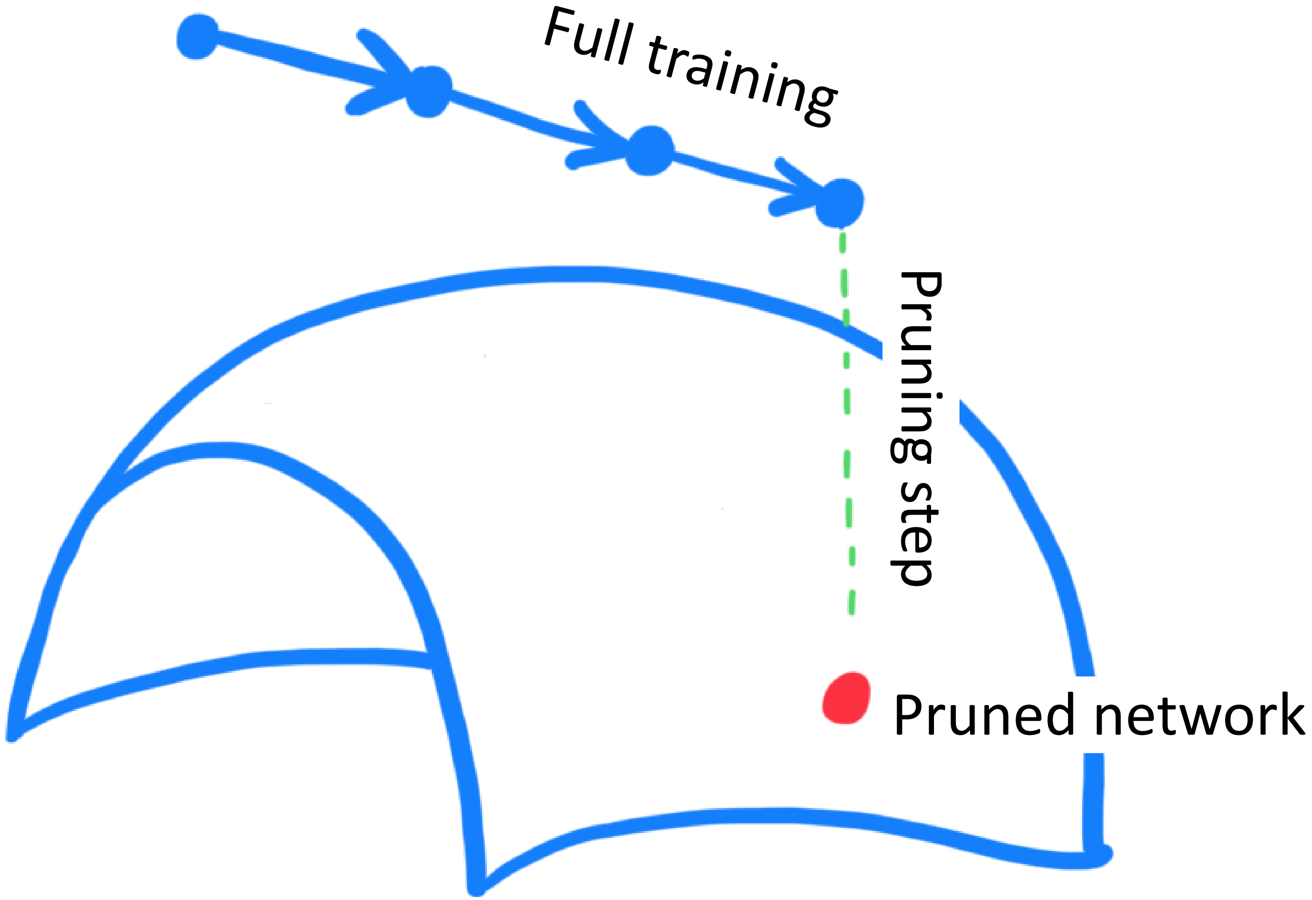


Full network vs pruned network as level of pruning increases on LeNet5

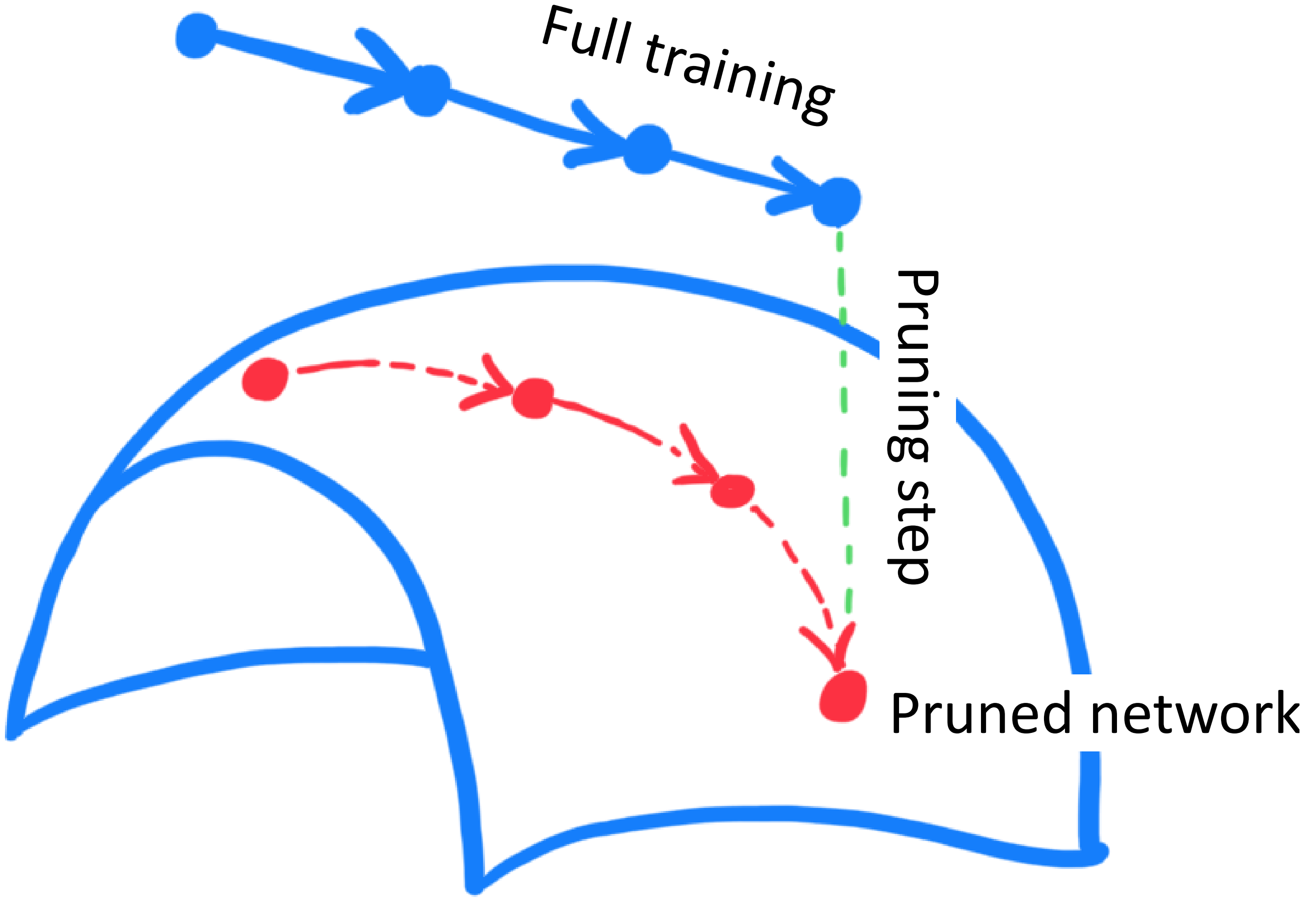
Bottlenecks

- **Speed-up**: Sparsification requires specialized hardware capable of leveraging sparse layers in order to produce a real speedup
- **Training memory:** In this approach, finding the winning tickets requires training the full network

Train and compress

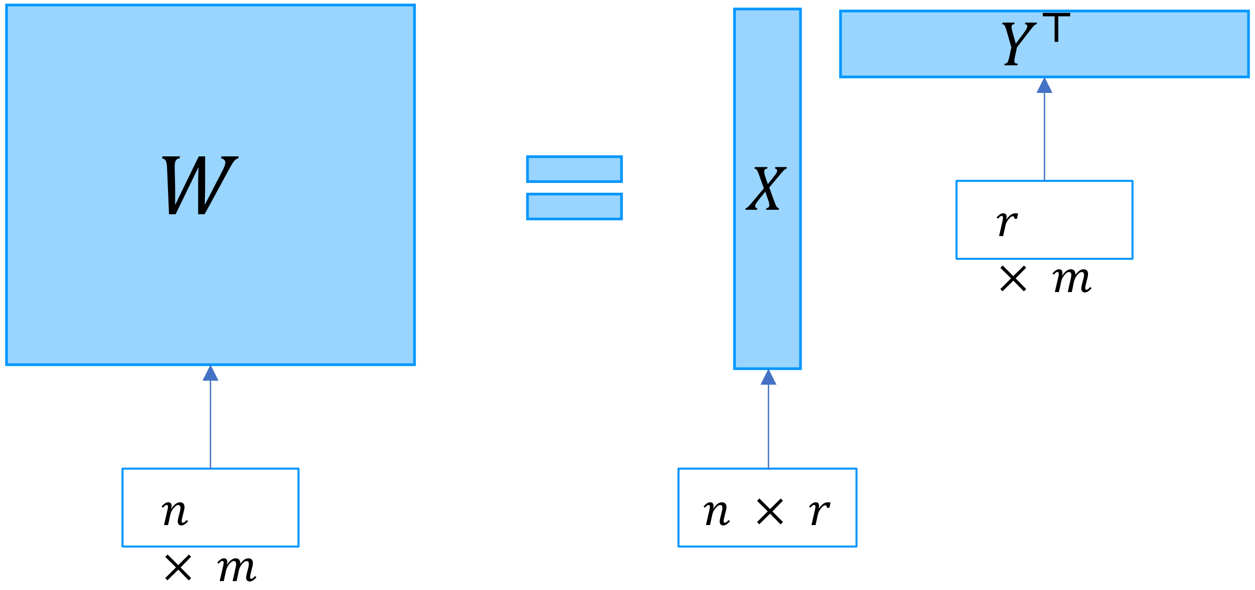


Train and compress / train while compressing



Layer factorization

NN: $$f(x) = h_N \quad h_{\ell+1} = f_\ell(h_\ell; W_\ell, b_\ell) \quad h_0 = x$$



Storage and operations cost: $O(nm)$ $O(nr + mr)$

Training factorized layers

Direct training of low-rank factorization

Fix the rank of the layers written as $W = XY^\top$ and interpret the loss L as a function of the factors:

$$\min_{W\sim n\times n} L(x;W) \longrightarrow \min_{X,Y\sim n\times r} L(x;X,Y)$$

Then train in parallel with respect to the two small variables

$$X \leftarrow \mathrm{sgd}[\nabla_X L(x;X,Y)]; \qquad Y \leftarrow \mathrm{sgd}[\nabla_Y L(x;X,Y)]$$

Fine-tuning LLMs: $f_\ell(x; W_\ell + A_\ell B_\ell)$

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* Yelong Shen* Phillip Wallis Zeyuan Allen-Zhu
Yuanzhi Li Shean Wang Lu Wang Weizhu Chen

QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers* Artidoro Pagnoni* Ari Holtzman

Luke Zettlemoyer

…and also (pre)training

ReLoRA: High-Rank Training Through Low-Rank Updates

Vladislav Lialin[†,‡*] Sherin Muckatira[†], Namrata Shivagunde[†], and Anna Rumshisky[†,§]

GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

Jiawei Zhao[1] Zhenyu Zhang[3] Beidi Chen[2 4] Zhangyang Wang[3] Anima Anandkumar[* 1] Yuandong Tian[* 2]

Drawbacks

- It is highly sensitive to initialization
  - Typically requires a full warm-up pass: train the full model for some epochs at the beginning and then compress with a (regularized) SVD
- It is highly sensitive to small singular values at initialization and during training
- Convergence can be slow
- It requires to manually adjust the ranks
  - Typically requires leaving k initial layers untouched; test for various choices of r

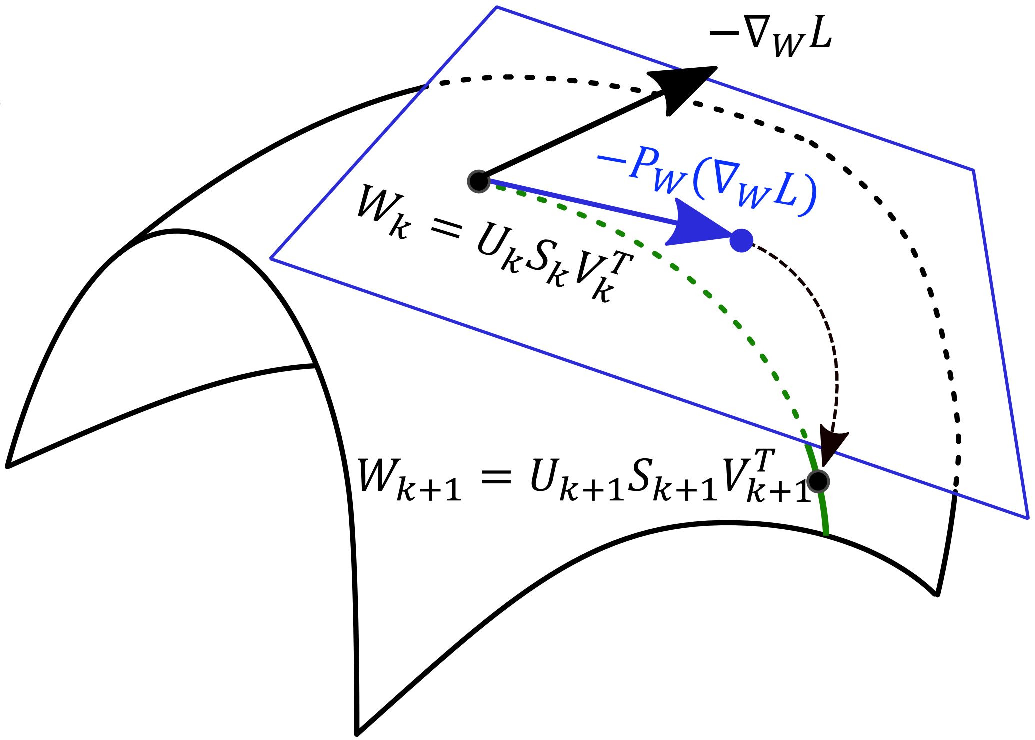
Training exploiting the Riemannian geometry

- When the factorization model forms a smooth manifold $\mathcal{M}$ one can "do better"

- Our approach:
  Project the gradient flow

$$\dot{W}(t) = -\nabla L\big(x; W(t)\big)$$

  onto the tangent plane at each point and then *retract* to $\mathcal{M}$ when integrating



USV parametrization

- $\mathcal{M} = \{USV^\top : U, V \sim n \times r \text{ orthonormal}, S \sim r \times r \text{ invertible}\}$
  Smooth Riemannian manifold

- P_X = projection on the tangent plane of $\mathcal{M}$ at the point $X \in \mathcal{M}$

- Projected gradient flow:

$$\dot{W}(t) = -P_{W(t)} \nabla L\big(x; W(t)\big)$$

System of ODEs for the factors

- When $W = USV^\top \in \mathcal{M}$, the projected gradient flow coincides with

$$\begin{cases} \dot{S} = -U^\top \nabla_W L(W) V \\ \dot{U} = -(I - UU^\top)\nabla_W L(W) V S^{-1} \\ \dot{V} = -(I - VV^\top)\nabla_W L(W)^T U S^{-\top} \end{cases}$$

However:

- Equation is stiff when the singular values of S are small
- It requires the gradient with respect to the full weight W

Change of parametrization

An unconventional DLR integrator, Ceruti, Lubich, BIT 2022

KLS trick:

Change variables: $K(t) = U(t)S(t), L(t) = V(t)S(t)^\top$

$$\begin{cases} \dot{S} = -\nabla_S L(USV^\top) \\ \dot{K} = -\nabla_K L(KV^\top) \\ \dot{L} = -\nabla_L L(UL)^\top \end{cases}$$

Note that K, L have the same thin shape as U, V.

Drawback: there is an implicit dependence on U and V → it is not obvious how one can run the updates in parallel

Rank-adaptive DLRT algorithm

Based on this system of ODEs, we propose a rank-adaptive training algorithm that simultaneously updates U, S, V and has several theoretical guarantees

Rank-adaptive DLRT scheme

GeoLoRA: Geometric integration for parameter-efficient ft, arXiv, 2024

1. In parallel:
   - $K \leftarrow \mathbf{sgd}[\nabla_U L(x_\mathrm{b}; USV^\top)]$; $L \leftarrow \mathbf{sgd}[\nabla_V L(x_\mathrm{b}; USV^\top)]$; $S \leftarrow \mathbf{sgd}[\nabla_S L(x_\mathrm{b}; USV^\top)]$
2. In parallel: $\widetilde{U} \leftarrow \mathbf{basis\_aug}(U, K)$ $\widetilde{V} \leftarrow \mathbf{basis\_aug}(V, L)$
3. Form the augmented $\widetilde{S} \leftarrow \begin{bmatrix} S & L^\top \widetilde{V} \\ \widetilde{U}^\top K & 0 \end{bmatrix} \sim 2r \times 2r$
4. Compress $\widetilde{S}$ to its best τ low-rank approximation

$$S \leftarrow \text{matrix } Z \text{ with smallest rank such that } \|\widetilde{S} - Z\| \leq \tau \, \|\widetilde{S}\|$$

   and form the new U, V accordingly

Training cost (per iteration)

- When we do the basis augmentation we need to:
  - Perform a QR decomposition of $n \times r$ matrix – cost: $O(nr^2)$
- For the best low-rank approximation up to error τ, we need to:
  - Perform an SVD decomposition of $r \times r$ matrix – cost: $O(r^3)$

$W \sim n \times n$	Cost per iteration	Memory storage
Direct	$O(2nr)$	$O(2nr)$
DLRT	$O(nr^2) + O(r^3) + O(2nr)$	$O(2nr + r^2)$
Full	$O(n^2)$	$O(n^2)$

Analysis

Theorem (Descent and convergence using SGD)

Assume the loss L is smooth and bounded. Let $W_k = U_k S_k V_k^T$ be the weight matrices computed after k training iterations using SGD with learning rate λ_k and truncation parameter τ_k.

Then:

$$\mathbb{E}_{k+1}[L(W_{k+1})] \leq L(W_k) - \lambda_k \left(1 - \frac{L\lambda_k}{2}\right) \mathbb{E}_k\left[\left\|P_{W_k} \nabla_{\xi_k} L(W_k)\right\|^2\right] + L\tau_k$$

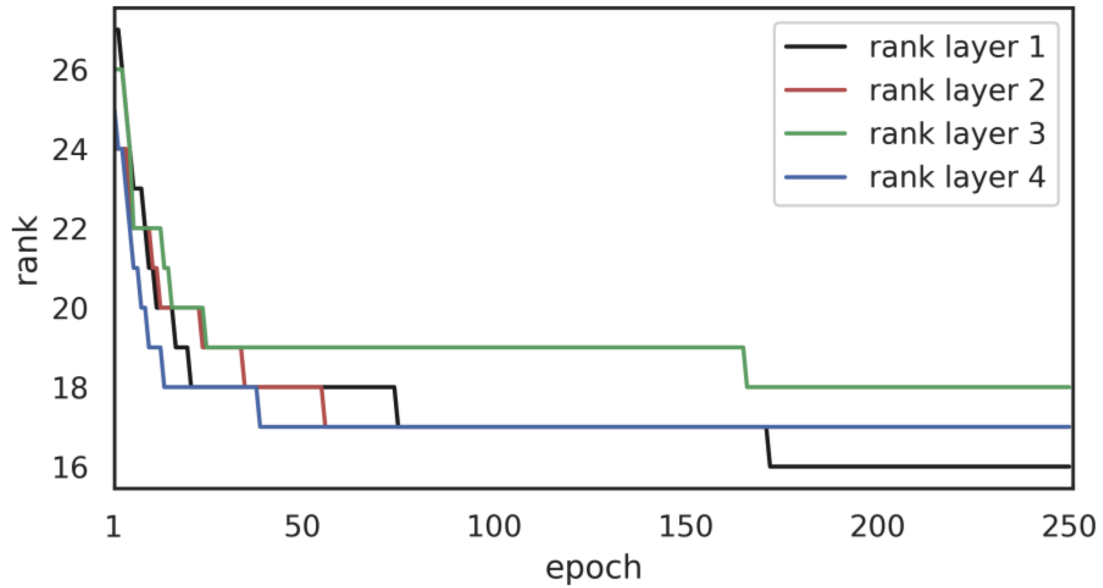
Theorem (Descent and convergence using SGD)

Moreover, assume:

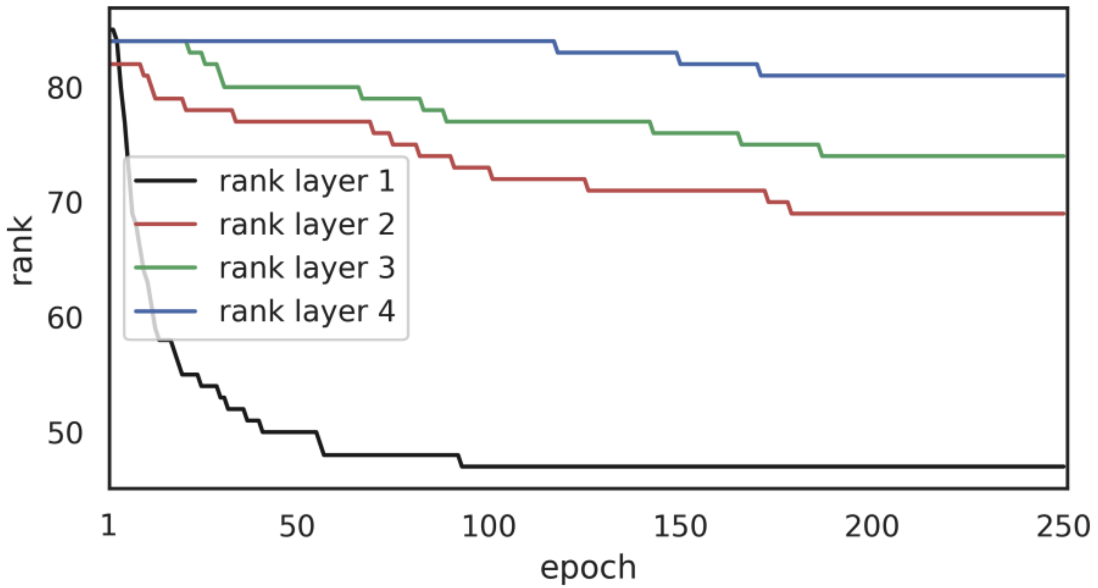
1. the learning rate sequence satisfies the Robin-Monro conditions
$$\textstyle\sum_k \lambda_k = +\infty \qquad \sum_k \lambda_k^2 < +\infty$$
2. the rank distribution stabilizes fast enough, namely
$$\textstyle\sum_k \left\| W_k - \widetilde{W}_k \right\|^2 < +\infty$$
where $\widetilde{W}_k$ is the matrix before rank adjustment.

$$\Longrightarrow \qquad \liminf_k \mathbb{E}_k \left[\left\| P_{W_k} \nabla_{\xi_k} L(W_k) \right\|^2 \right] = 0$$

Rank evolution (feed forward fully connected)



(a) Rank evolution for $\tau = 0.15$



(b) Rank evolution of $\tau = 0.05$

Theorem (Error-bound when using GD)

Assume:

- $W(t) \leftarrow$ solution of the full-model gradient flow at time t
- $W_k = U_k S_k V_k^\top \leftarrow$ computed after k training iterations of DLRT, with GD using learning rate λ and error tolerance τ

If the gradient $\nabla L(W(t))$ is ϵ-close to $\mathcal{M}_{r_k}$ at time $t = k\lambda$, then

$$\left|W(t) - U_k S_k V_k^T\right| \leq c_1\epsilon + c_2\lambda + c_3\tau/\lambda$$

Winning ticket interpretation

An *informal* way to interpret the previous theorem:

If there exists a low-rank winning ticket (a highly-performing low-rank subnetwork) then the proposed DLRT scheme well approximates it

Is there a good low-rank subnet?

Extensive recent work on low-rank implicit bias

- Arora, Cohen, Hu, Luo, Implicit regularization in deep matrix factorization, NeurIPS 2019
- Singh, Bachmann, Hofmann. Analytic insights into structure and rank of neural network Hessian maps, NeurIPS 2021
- Feng, Zheng, Huang, Zhao, Jordan, Zha, Rank Diminishing in Deep Neural Networks, NeurIPS 2022
- Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, The low-rank simplicity bias in deep networks, TMLR 2023
- Chou, Gieshoff, Maly, Rauhut, Gradient descent for deep matrix factorization: Dynamics and implicit bias towards low rank, Appl Harmonic Analysis, 2024
- Galanti, Siegel, Gupte, Poggio, SGD and Weight Decay Provably Induce a Low-Rank Bias in Deep Neural Networks, 2024

Majority of recent theory is for linear networks

[Bah, Rauhut, Terstiege, Westdickenberg, *Deep linear neural networks: Riemannian gradient flows and convergence to global minimizers*, 2022.]

Deep linear network with N layers: $f_\theta(x) = W_N W_{N-1} \cdots W_2 W_1 x$

Thm: (Training on low-rank manifold and full training coincide)

- L be the square loss $L(\theta) = ||f_\theta(x) - y||^2$
- $W^* = W_N^* W_{N-1}^* \cdots W_2^* W_1^*$ be s.t. $W_i^* = \operatorname{argmin} L(\theta)$

Then, if $\operatorname{rank} W^* = r$, we have $W_i^* = \operatorname{argmin}_{\operatorname{rank} W_i = r} L(\theta)$

Theorem (Neural Rank Collapse)

Given a dataset $\mathcal{X} \subseteq \mathbb{R}^d$ let

$$\mathrm{TCV}(r) = \min_{\substack{C_1,\dots,C_r \subseteq \mathcal{X} \\ \cup_i C_i = \mathcal{X}}} \sum_{i=1}^{r} \sum_{x \in C_i} \|x - \mu(C_i)\|^2$$

and let W_ℓ be a stationary point of **any** FFNN trained with $L + \lambda \|W\|^2$. Then,

$$\min_{\mathrm{rank}(Z) \leq r} \|W_\ell - Z\|^2 \leq C_\ell \min_{k<\ell} \frac{\mathrm{TCV}(\mathcal{X}_k, r)}{\lambda}$$

where $\mathcal{X}_k$ is the output of layer k. In other words: $W_\ell = R(r) + \text{small err}$

4-layer auto encoder

$\mathcal{X} = 10$ Gaussian blobs with variance σ^2

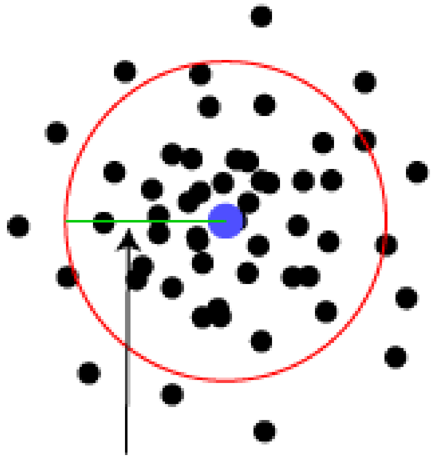
$$\Downarrow$$

$$\mathrm{TCV}(\mathcal{X}, 10) = \sigma^2$$

$$\Downarrow$$

$$W_\ell = rank10 + E$$

$$\|E\| = O(\sigma^2/\lambda)$$



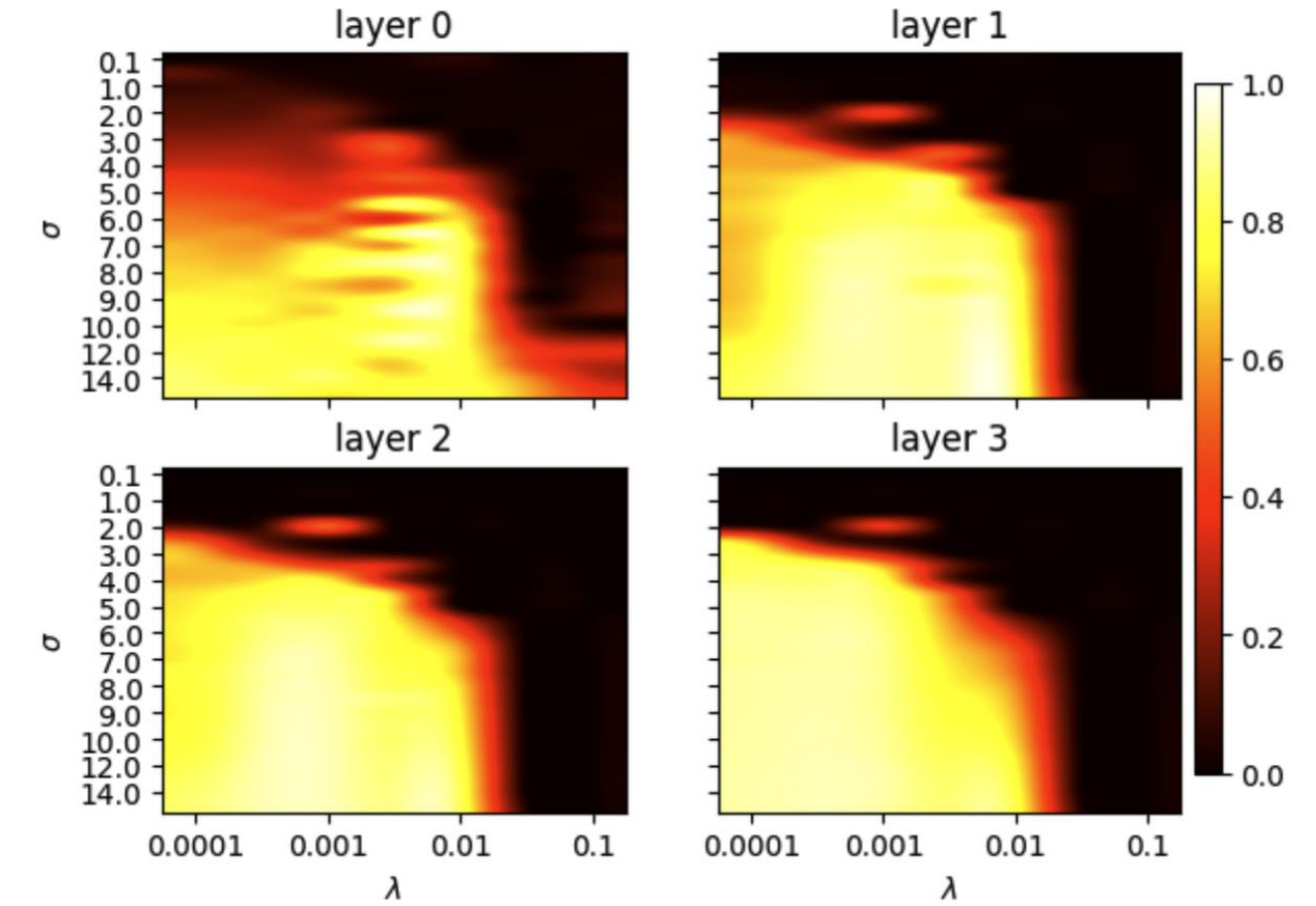


Fig. 6: Relative singular value tail error of each layer $\frac{\sum_{j>K}\sigma_j(W)^2}{\sum_j \sigma_j(W)^2}$.

Tensor kernels

Almost everything transfers to the tensor case.

For example, consider a conv kernel

$$(W * X)(i_1, i_2, i_3, i_4) = \sum_{j_2, j_3, j_4} W(i_2, j_2, j_3, j_4) X(i_1, j_2, i_3 - j_3, i_4 - j_4)$$

- Two possibilities: (a) matricization, (b) tensor factorization

CNN via tensor factorizations

It is known that tensor factorizations work well on CNNs

CP

$$T = \sum_i w_i \otimes x_i \otimes y_i \otimes z_i$$

- Lebedev,Ganin,Rakhuba,Oseledets,Lempitsky, ICLR 2015
- Astrid, Lee, BigComp 2017
- Phan, Sobolev, Sozykin, Ermilov, Gusak, Tichavský, Glukhov, et al, ECCV, 2020

Tucker

$$T = C \times_1 U_1 \times_2 U_2 \times_3 U_3 \times_4 U_4$$

- Kim, Park, Yoo, Choi, Yang, Shin, ICLR 2016
- Kossaifi, Bulat, Tzimiropoulos, Pantic, CVPR 2019
- Song, Zhang, Li, ICMLT 2020

TDLRT

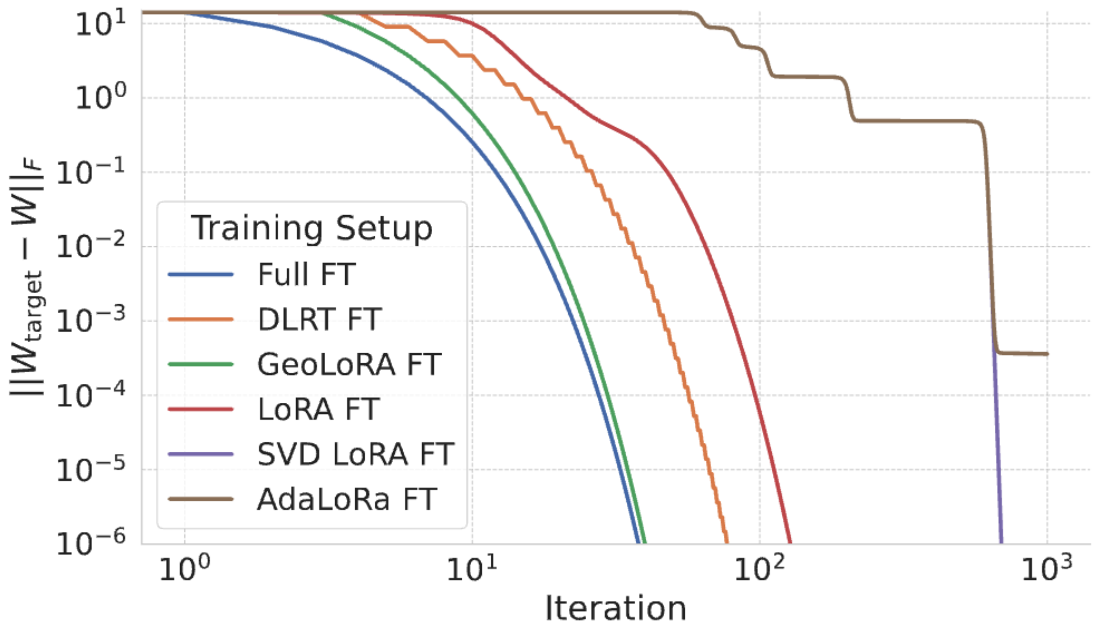
Tucker decomposition forms a smooth manifold, so we can define the tangent plane there and adapt the algorithm using HOSVD in place of the classical SVD

Experimental evaluation

$$\min_{W} \|W_{\text{target}} - W\|^2$$

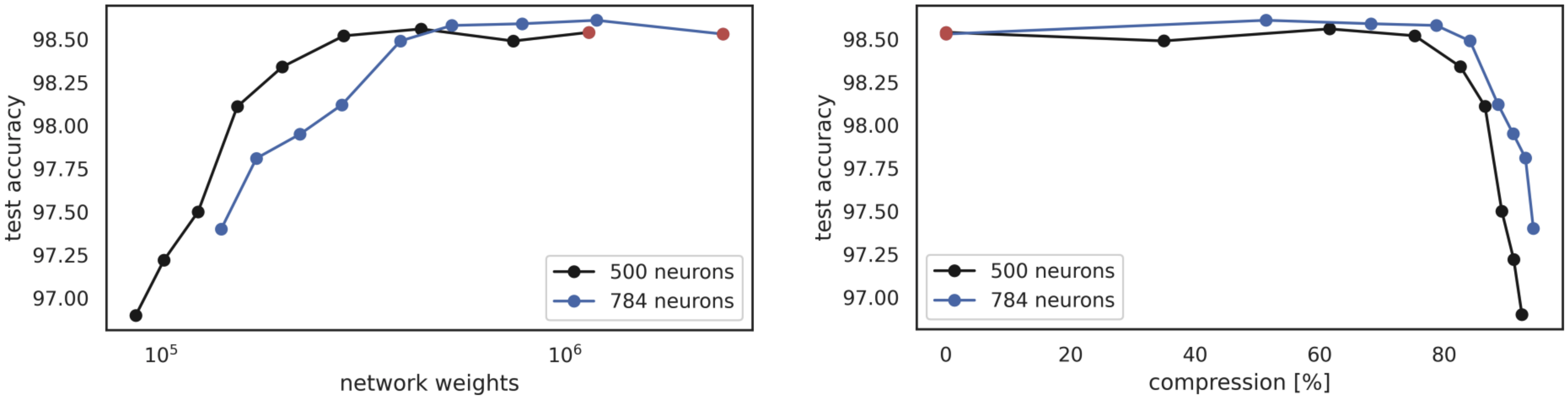
$W \sim 5000 \times 5000$, $\text{rank}(W) = 5$, $\lambda = 0.1$, Initial rank $= 50$

1. Full fine-tuning (FT) (blue),
2. DLRT from (Schotthöfer et al., 2022) (orange),
3. The proposed GeoLoRA method (green),
4. Fixed rank LoRA from Hu et al. (2021) (red),
5. AdaLoRA from (Zhang et al., 2023) (brown),
6. Fixed rank AdaLoRA (purple).



AdaLoRA imposes orthogonality on U, V adding penalty terms $\|U^\top U - I\|^2$

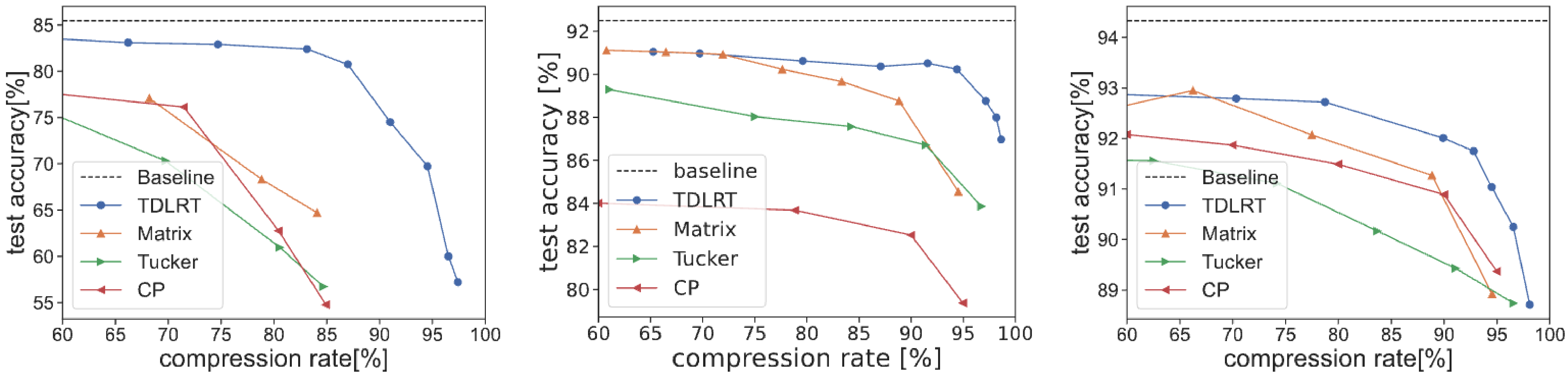
Compression rate behavior (FFFC)



LeNet

	method	NN metrics: test acc.	NN metrics: ranks	Evaluation: params	Evaluation: c.r.	Train: params	Train: c.r.
	LeNet5	**99.2**%	[20, 50, 500, 10]	430500	0%	430500	0%
DLRT	$\tau = 0.11$	98.0%	[15, 46, 13, 10]	47975	88.86%	50585	88.25%
DLRT	$\tau = 0.15$	97.8%	[13, 31, 9, 10]	34435	92.0%	35746	91.7%
DLRT	$\tau = 0.2$	97.2%	[10, 20, 7, 10]	25650	94.04%	26299	93.89%
DLRT	$\tau = 0.3$	95.3%	[6, 9, 4, 10]	15520	**96.4**%	15753	**96.34**%
	SSL [62] (ft)	99.18%		110000	74.4%		$< 0\%$
	NISP [58] (ft)	99.0%		100000	76.5%		$< 0\%$
	GAL [42]	98.97%		30000	93.0%		$< 0\%$
	LRNN [28]	98.67%	[3, 3, 9, 9]	18075	95.8%		$< 0\%$
	SVD prune [61]	94.0%	[2, 5, 89, 10]	123646	71.2%		$< 0\%$

Comparison with direct factorization descent



(a) Alexnet Cifar10 (b) VGG16 Cifar10 (c) ResNet18 Cifar10

CIFAR10

		VGG16		Alexnet		ResNet18	
		test acc. [%]	c.r. [%]	test acc. [%]	c.r. [%]	test acc. [%]	c.r. [%]
	Baseline	92.01	0.0	85.46	0.0	94.33	0.0
Factorization	**TDLRT** (ours)	**90.23**	**94.40**	**82.39**	83.12	92.72	78.73
	Matrix DLRT [68]	89.13	83.22	73.57	71.57	80.98	56.85
	Tucker-factorized [38]	86.71	91.4	70.30	69.74	91.11	74.19
	Matrix-factorized [79]	84.54	94.34	77.07	68.20	92.07	77.49
	CP-factorized [44]	82.53	89.98	76.14	71.46	91.87	69.95
	Tucker RGD [74]	81.48	84.26	73.88	74.01	**92.76**	74.18
	TT-factorized [62]	87.27	90.30	78.13	**88.14**	87.13	81.24
Pruning	SNIP [45]	89.58	56.23	–	–	89.50	78.50
	IMP [20]	87.21	58.54	–	–	90.50	**82.50**
	GraSP [78]	88.50	77.30	–	–	89.40	77.90

GLUE benchmark (fine-tuning)

General Language Understanding Evaluation

Method (# Params)	MNLI (Acc)	SST-2 (Acc)	CoLA (Mcc)	QQP (F1)	QNLI (Acc)	RTE (Acc)	MRPC (Acc)	STS-B (Corr)
Full FT (184M)	89.90	95.63	69.19	89.80	94.03	83.75	89.46	91.60
BitFit (0.1M)	89.37	94.84	66.96	84.95	92.24	78.70	87.75	91.35
HAdapter (1.22M)	90.13	95.53	68.64	89.27	94.11	84.48	89.95	91.48
PAdapter (1.18M)	90.33	95.61	68.77	89.40	**94.29**	85.20	89.46	91.54
LoRA r=8 (1.33M)	90.29	95.29	68.57	90.61	93.91	85.5	89.75	89.10
AdaLoRA (1.27M)	**90.44**	95.64	68.76	**90.65**	94.11	**86.00**	89.44	91.41
GeoLoRA (reported individually)	89.98 (0.7M)	**95.98** (1.17M)	**69.03** (0.98M)	90.53 (0.69M)	94.23 (0.7M)	85.93 (1.19M)	**90.10** (0.75M)	**91.58** (0.71M)
Evaluation and train time comparison								
AdaLoRA (eval/train) [it/sec]	12.4/4.3	17.6/6.7	24.6/8.1	9.2/3.2	4.9/1.6	10.3/3.2	9,9/3.1	21.1/**8.5**
GeoLoRA (eval/train) [it/sec]	**17.1/4.9**	**21.3/8.3**	**37.4/9.1**	**12.0/3.8**	**5.9/1.8**	**13.2/3.7**	**12.6/3.7**	**21.3**/8.3

ViT and Stable Diffusion

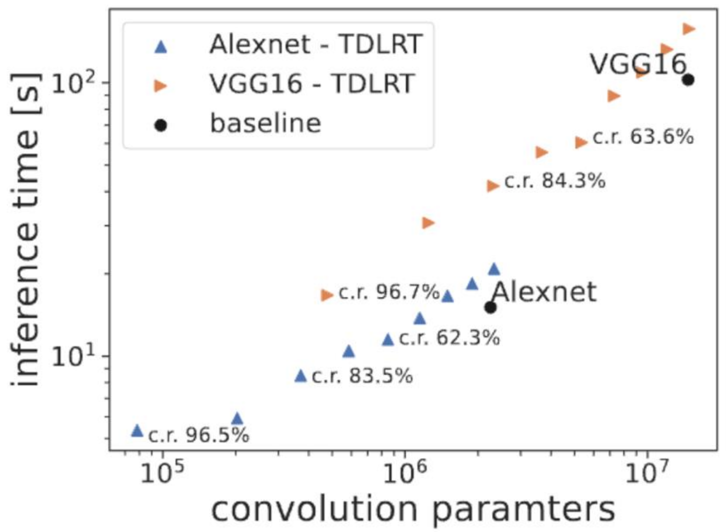
Table 3: Vit-base-patch16-224 fine-tuning on Cifar10, 100 and Tiny-Imagenet. We compare AdaLoRA to GeoLoRA with local and global budgeting reporting the median of 5 runs using different random seeds.

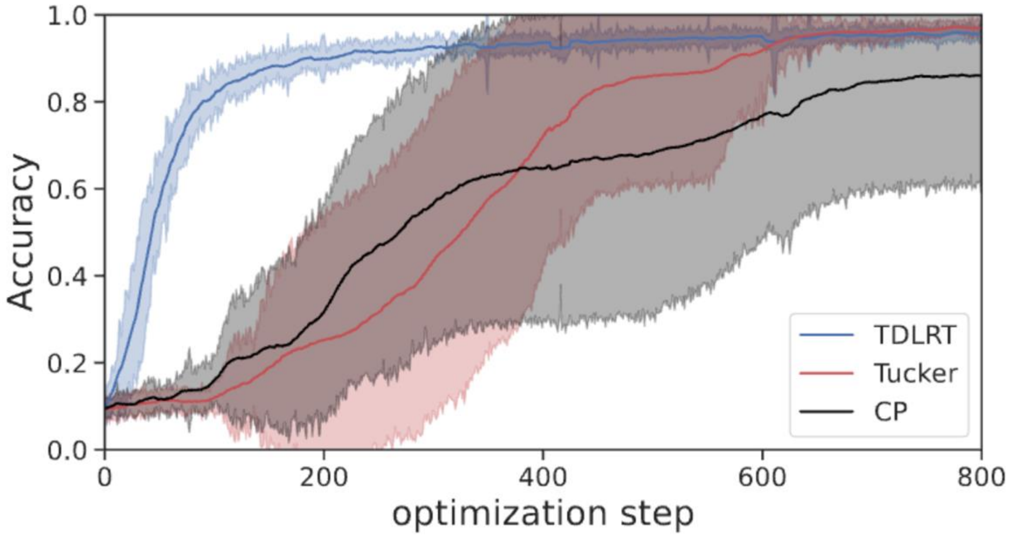
Method	Cifar 10 [%]		Cifar 100 [%]		Tiny-Imagenet [%]	
	# Params	Acc [%]	# Params	Acc [%]	# Params	Acc [%]
AdaLoRA	**0.47M**	98.51	0.45M	91.44	0.9M	87.2
GeoLoRA, local	**0.47M**	**98.55**	**0.35M**	**91.63**	0.92M	**88.09**
GeoLoRA, global	0.48M	98.51	0.47M	91.62	**0.75M**	88.07

Table 4: Stable Diffusion on Dreambooth benenchmark. We compare LoRA and GeoLoRA reporting the median of 5 runs.

Method	Val. Loss	# Params
LoRA ($r = 5$)	0.275	3.0 M
LoRA ($r = 3$)	0.281	1.8 M
GeoLoRA ($\tau = 0.02$)	**0.242**	**2.6M**
GeoLoRA ($\tau = 0.1$)	**0.257**	**1.4M**

Time and sensitivity to singular values





Method	time to 60% acc.
Tucker (best case)	2.35s
Tucker (worst case)	4.70s
TDLRT (fixed rank)	2.41s
TDLRT	4.16s

Conclusions

- Large/deep nets tend to be (approximately) low-rank
- Training low-rank factors is not necessarily straightforward
- We propose a particular Riemannian optimization strategy (DLRT) that has several theoretical guarantees
- DLRT outperforms default gradient descent on the factorization $XY^\top$ both in the martrix and the tensor Tucker case

Thank you!

- [NeurIPS22] Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations
- [NeurIPS23] Robust low-rank training via approximate orthonormal constraints
- [NeurIPS24] Geometry-aware training of factorized layers in tensor Tucker format
- [arXiv] Neural rank collapse in feed-forward networks
- [arXiv] GeoLoRA: Geometric integration for parameter-efficient fine-tuning