ToRA: Tensor Adapter for Parameter Efficient Finetuning

Anonymous ACL submission

Abstract

Recent studies show LoRA cannot reach the performance of full fine-tuning (FFT). This work shows weights and gradients during FT has a long-tail plus high-rank and LoRA's difficulties stem from its core low-rank matrix factoring assumption. ToRA is a LoRA style parallel adapter, using Tensor Train decomposition to efficiently represent the high-rank ΔW. ToRA consistently outperforms LoRA. For example, rank-8 ToRA beats LoRA for all ranks up to 128. Sometimes by more than 10 points - 80.32 vs. 69.56 for BoolQ and 48.82 vs. 34.20 on MMLU with Llama-3.2-3B. ToRA adapts all self-attention blocks for all layers using the same budget as LoRA - no tuning nor compromise is needed. It also pairs well with popular quantization methods like QLoRA. ToRA is a strong contender as a drop-in replacement for LoRA.

1 Introduction

Recent rapid progress in using generative models has relied on fine-tuning (FT) large pretrained models. A common pain point is fine-tuning these LLMs on commodity GPUs or even high-end GPUs. For example, LLama3-8B with fp16 needs 16Gb to load. FT usually requires at least $4 \sim 12$ times more memory to store activations, momentum etc. Practitioners resort to distilling, aggressive quantizing, dropping weights, or paging Alizadeh et al. (2023) which are time consuming and compromise quality and speed.

PEFT Houlsby et al. (2019) exemplified by Huggingface's adapters with a locked model, drastically reducing the required memory for FT. LoRA Hu et al. (2022) fine-tune with a set of low-rank matrix factors BA and is often the method of choice. It is most useful in the pure adapter form. The weights are not merged back into the LLM, thus avoiding catastrophic forgets and allowing many adapters to share one large model on the server. Additionally, adapter FT reduces the bandwidth needed in distributed setups and supports highly quantized models. Apple's Foundation Model and LoRA-Hub Huang et al. (2024) are good examples of such a system with a few highly tuned LLMs and many adapters nurturing an ecosystem. Instead of merging back, inference latency can be resolved by architectures like S-LoRA Sheng et al. (2024) and fLoRA Wen and Chaudhuri (2024).

Our contributions: (1) ToRA uses a tensor decomposition which is more powerful than the matrix factorization in LoRA. (2) ToRA seamlessly integrates with QLoRA. (3) ToRA uses a novel Kaiming style initialization critical for a tensor based adapter. (4) ToRA consistently outperforms LoRA by as much as 10 points on various datasets. (5) Our thorough empirical and theoretical analysis of the heavy-tail/high-rank nature of FT can help to guide future works on PEFT.

2 Preliminaries

First, the intrinsic dimension of FT is studied, followed by evidence of high-rank and heavy-tail gradients encountered in LLMs. Next drawing lessons from Yang et al. (2018), we study the *Softmax Bottleneck* in modern LLMs. Further evidence from FT a compressed or quantized model sharpens the focus on the high-rankness of gradient updates and problems encountered that we must address.

2.1 LoRA's Critical Weakness & Long-Tail

LoRA appeals to Aghajanyan et al. (2021) study of the intrinsic dimensions of FT. This approach was first proposed by Li et al. (2018) using a model with a random projection P and a frozen model parameterized as $\theta_0^{(D)}$:

$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)} \tag{1}$$

where $\theta^{(d)}$ is initialized to zero. The rank d is gradually increased until the augmented model reaches

90% of the performance of FFT. "Intrinsic dimension" is defined as the d needed to capture 90% of the performance of the FFT $\theta^{(D)}$. Aghajanyan et al. (2021) found the intrinsic dimension d to be in the hundreds to thousands and concluded $\theta^{(d)}$ is "low-rank". Putting things in perspective, a d in the hundreds might be low-rank relative to the number of weights in $\theta^{(D)}$, which is in the billions or greater. Their finding contradicts LoRA's notion of low-rank with $r = 2, 4, 8$.

LoRA use a low-rank matrix factoring for ΔW:

$$h = W_0 + \Delta W, \ \ \Delta W \approx BA^\top \tag{2}$$

W_0 is locked and $B \in \mathbb{R}^{d \times r}$ is set to 0. One can immediately see $\theta^{(D)} = W_0$, $\theta^{(d)} = BA^\top$ and $P = A$ which is random initialized. This "Noise & Zero" init is a characteristic of LoRA designed to match the gradients of FFT when no adapter is applied.

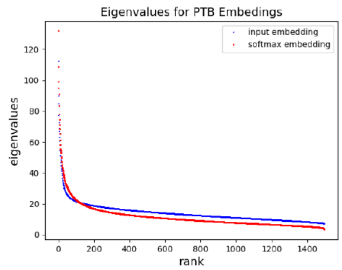
Recent studies by Xia et al. (2024); Lialin et al. (2024) show LoRA cannot reach performance of full FT, especially for tasks like complex reasoning and continual learning Liu et al. (2024). The limitations of LoRA and how they stem from low-rank factoring assumptions will be analyzed.

Softmax bottleneck Yang et al. (2018) consider language model as a finite set of pairs of context and its conditional next-token distribution $\mathcal{L} = \{(c_1, P^*(X \mid c_1)), \cdots, (c_1, P^*(X \mid c_N))\}$, where N is the number of possible contexts. The objective of a LM is to learn a distribution $P_\theta(x \mid c)$ parameterized by θ to match the true distribution $P^*(X|C)$. The contexts are encoded into vectors of dimension d, which is multiplied by the token embeddings (Inan et al., 2017) using dot-prod[1] to obtain the *logits* ($\mathbf{h}_c^T \mathbf{w}_x$) to feed the Softmax to produce a *categorical distribution over the next token* -

$$P_\theta(x \mid c) = \frac{exp\, \mathbf{h}_c^T \mathbf{w}_x}{\sum_{x'} exp\, \mathbf{h}_c^T \mathbf{w}_x} \tag{3}$$

$\mathbf{h}_c$ is the context vector/hidden state, and $\mathbf{w}_x$ is a token embedding - i.e. LLM is **a rank-d factoring** of $\mathbf{A}$ (the full autoregressive matrix of *logits*). For example, the hidden dimension d for LLama-3.2-3B is 4096 and 2048 for 1b. Yang et al. (2018) shows this d is the critical factor limiting the expressive power of LLMs and propose a *mixture-of-softmax*(MoS) to support much higher rank modeling. Their experiments show MoS reduced perplexity up to $d = 9981$ and beyond.

[1] context, token embedding have the same dimension d



(b) eigenvalues

Figure 1: Chen et al. (2018) Heavy-tail eigenvalues in embeddings. Eigenvalues at rank 200 and beyond are still significant (with permission).

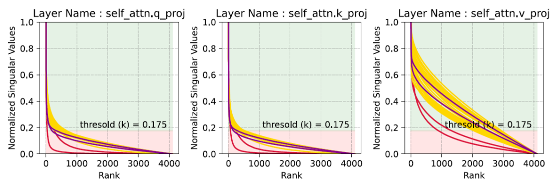


Figure 2: Jaiswal et al. (2024) Singular Values for different layers of LLama-7B. Yellow are the attention layers. Red and purple are the input and output layers (with permission).

Besides "Softmax Bottleneck" Fig. (1) is a plot of eigenvalues of embedding from Chen et al. (2018) with the "elbow" occuring around rank-200 and has a heavy-tail. Which suggests one cannot ignore higher eigenvalues without peril. Sainath et al. (2013) had to use rank 128 and above to compress the cross-entropy without compromising results. All these findings support our thesis that LLM matrices and gradients exhibit high ranks and long tails, posing significant challenges for LoRA but not for ToRA

2.2 Low-Rank Subspace and Gradients

Like ToRA, GaLore Zhao et al. (2024) and WeLore Jaiswal et al. (2024) also appeal to heavy-tail of singular values and found gradients have highly variable ranks. Fig (2) from WeLore show the heavy-tail distribution in various weight matrices and the "elbow" being > 200. Even more pronounced for `attn.v_proj` which is around 1000. We will come back to `attn.v_proj` later. Q-BERT Shen et al. (2020) found the top eigenvectors of the Hessian of gradients varies a lot during training. They all found the ranks to be at least in the few hundreds consistent with findings of Aghajanyan et al. (2021).

Gur-Ari et al. (2018) found gradient updates occur in a subspace matching the number of classes in

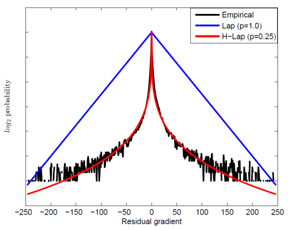


Figure 3: Badri and Shaji (2023) heavy-tail distribution of residual gradient error. A kurtotic distribution is needed - e.g. H-Laplacian (with permission)

a classifier and is used by LoRA to justify its factoring. However, the number of classes in LLM is very high, equal to the number of equivalence classes in the context window for an auto-regressive N-gram model Kneser (1995). "Softmax Bottleneck" dissected LLM itself as a low-rank factoring of the super high-rank auto-regressive matrix $\mathbf{A}$ of `logits` and is estimated using softmax to yield a *categorical distribution over semantic equivalence classes*. One must keep this firmly in mind when designing a PEFT method.

High-rank Gradients even if gradients are low-rank it does not mean their cumulative effect stays low-rank. Each update might and indeed can pull in highly variable directions. GaLore uses a projective gradient descent $\Delta W = P_t^T G_t Q_t$ to squeeze high-rank updates in a small core G_t. They use $r = 128..512$ even with periodic merge backs. WeLore attempts to improve GaLore by classifying layers into low-rank (LRC) and non-LRC and sidestep the problem by not adapting the NLRCs. MoRA Jiang et al. (2024) employs a rank-256 square matrix with input and output projector to achieve high-rank updating. ReLoRA Lialin et al. (2024) found gradient updates are often high rank and propose to use several LoRA adapters to approximate it. All four require periodic merge backs into W_0 negating savings in GPU memory and destroying any ability to share a model. RoSA Nikdan et al. (2024) uses the L+S decomposition $\Delta W = \Delta^L + \Delta^S$ from sparse coding. Adapters are stored in CSR format with a custom kernel plus capturing and storing full gradients etc. All of them recognize gradients are high-rank but their solution to squeeze them into a low-rank representation is not satisfactory because their core representation is still a low-rank *matrix*.

GaLore, WeLore, Fira Chen et al. (2024), MoRA, ReLoRA, PLoRA Meng et al. (2024b) and RoSA all understand the difficulties of high-rank gradients. The later four in the context of LoRA-like adapter FT. None of them consistently beat LoRA. Moreover, their solutions leave something to be desired. Most of them require merge-back into W_0. ToRA is an efficient approximation that can handle such updates.

Slow Convergence of LoRA LoRA+ Hayou et al. (2024) and PiSSA Meng et al. (2024a) study the slow convergence of LoRA stemming from its BA initialization to "Noise & Zero" which led to small gradients and slow convergence. Instead PiSSA *initializes the adapter matrices A and B with the top r principal components of* W_0, and put the remaining components into a residual matrix $W^{res} = (W_0 - \eta B_{init} A_{init})$ which is locked during training. W^{res} takes the place of W_0 during FT. $B_{init} A_{init}$ contains the low-rank factoring of W^{pri} instead of zero which solves the slow convergence of LoRA. However, in the experiments PiSSA does not consistently beat LoRA.

HQQ Badri and Shaji (2023) utilize a hyper-Laplacian (fig. 3) distribution to compress Llama-70B to 1 or 2-bits thereby heightens the urgency of facing long-tail squarely.

LoRA's main weakness is its low-rank matrix assumption in the face of strong evidence the singular values of weights and gradients have a heavy-tail distribution. Since gradients are persistently high rank, one cannot capture them faithfully if they are forced into a low-rank subspace. This work will provide a solution for these problems using a carefully chosen efficient tensor decomposition.

3 ToRA & Tensor Train Decomposition

The main contribution of ToRA is overcoming LoRA's low-rank limitation to approach full-fine tuning (FFT) performance with a smaller memory budget while preserving the pure adapter form. From above, it is evident one needs a tool that can accumulate gradients with a heavy-tail and highly variable ranks into an efficient representation with minimal loss. Among the possible forms of tensor decomposition, Tensor Train (TT) Oseledets (2011) is a balanced product factoring into d terms for a d-dim tensor:

$$\mathcal{A}(j_1, \ldots, j_d) = \underbrace{\mathbf{G}_1[(j_1)]}_{1 \times r_1} \underbrace{\mathbf{G}_2[(j_2)]}_{r_1 \times r_2} \ldots \underbrace{\mathbf{G}_d[(j_d)]}_{r_{d-1} \times 1} \tag{4}$$

Each $\mathbf{G}$ (`TT-core`) is a 3D array $r_{n-1} \times I_n \times r_n$ except for the first and last. $\mathbf{G}_1$ and $\mathbf{G}_d$ are always rank-1 modes for interfacing with vectors

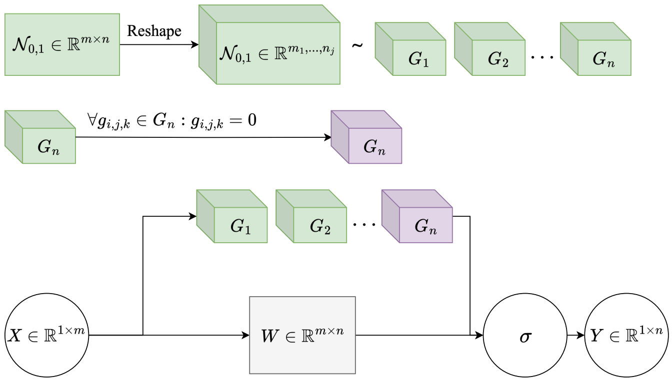


Figure 4: ToRA-Adapter: a `nn.linear` is initialized with Kaiming, then approximated with `tt-svd` into tensor-train form and zero out last core then discarded. The adapter is applied to W (K,Q,V) for all layers using LoRA-16 budget. ToRA naturally supports heavy tailed updates in a parameter efficient way.

and matrices. TT is a "chain" of smaller factors connected by `tensor-contraction` operator Kossaifi et al. (2017) for adjacent pairs of cores along a shared dimesion (e.g. r_1 for $\mathbf{G}_1[(j_1)]$ $\mathbf{G}_2[(j_2)]$). The `tensor-contraction` computes a tensor Kronecker-product. It is efficiently implemented in GPU friendly manner as a `reshape + matmat-mul`. It is supported in cuBLAS and TensorFlow etc. The number of terms in eq. 4 (3..5 for ToRA) is controlled by tensorization. TT bring multi-linear algebra to this problem and has many attractive numerical properties. TT has linear scaling with respect to dimension using $O(dnr^2)$ to store $O(n^d)$ elements. This critical property is not shared by common tensor forms like Tucker.

TT as Generalized Low-Rank Decomposition Rewriting eq. 4 as a TT factoring of dense ΔW, $\underbrace{\mathbf{G}_1[(j_1)]}_{1\times r_1} \mapsto \mathbf{U}$, $\underbrace{\mathbf{G}_d[(j_d)]}_{r_{d-1}\times 1} \mapsto \mathbf{V}^\top$ -

$$\Delta W \approx \mathbf{U} \prod_{j=2}^{d-1} \mathbf{\Sigma}_i \mathbf{V}^\top \tag{5}$$

This aligns TT decomposition into a SVD-like form used in LoRA. For LoRA, eq (5) has one $\mathbf{\Sigma}_i$ term and is a diagonal matrix becoming a standard low-rank matrix factoring. In ToRA, $\mathbf{\Sigma}_i$ is not limited to be a diagonal matrix but a set of 3D core-tensors in a product factoring. This is the source of the greater representation power of ToRA. In both LoRA and ToRA, $\mathbf{U}$ and $\mathbf{V}$ is a linear down and up projector into a lower-rank subspace. In LoRA, this subspace is the LoRA rank (e.g. 16) while for ToRA it is much much higher, corresponding to the unfolded dimension of $\mathbf{G}_2[(j_2)]$. One example is `qkv_proj` from OpenELM, ToRA use a 5 term decomposition:

tt-svd$(\Delta W \in \mathbb{R}^{1152\times 1280}) =$ $\{\mathbf{G_1} : (1,4,4,16); \mathbf{G_2} : (16,4,4,23);$ $\mathbf{G_3} : (23,4,4,29); \mathbf{G_4} : (29,6,4,15);$ $\mathbf{G_5} : (15,3,5,1)\}$ last dimension of each term is the same as the 1st dimension of the next term - those are the connections for tt-contraction.

Previous Works on TT for Deep Networks FacT Jie and Deng (2023), LoRTA Hounie et al. (2024) and LoTR Bershatsky et al. (2024) are the closest to ToRA in applying tensor decomposition to PEFT. All of them represent all the ΔWs as a single tensor and use weight sharing across layers. Their focus is maximum model compression and not FT. A single tensor with weight sharing might entail more tuning and difficult analysis as to which layers can profit from sharing etc. All of them focus on reducing the memory lower bound of LoRA by tensor decomposition together with weight sharing while ToRA aims for best-in-class FT performance. Moreover, they either use the simplest form of TT, with one 3D core without tensorization or use Tucker form which does not have linear scaling. Its storage grows exponentially and is limited by curse-of-dimensionality Kolda and Bader (2009). LoRTA and Hutchinson et al. (2011) both use CPD form (Kolda and Bader, 2009), which factor a dense tensor into a sum of rank-1 factors. However, CPD approximation with a fixed canonical rank is ill-posed (de Silva and Lim, 2008). Critically, Tucker and CPD are not endowed with an equivalent `TT-svd` algorithm which is needed to perform accurate construction of a tensor approximation and proper adapter initialization. The importance of good TT initialization will be comprehensively studied.

FacT trained for 100 epochs while ToRA use 5. It use much smaller TT ranks (1,4,8,16) than current best practice. `FacT-TT` do not possess ToRA's novel `tt-svd` based adatper init which is shown to be critical for good TT performance. Both FacT and LoRTA cannot reach LoRA's performance while ToRA beat LoRA consistently and by large margins. LoRTA achieved 61.8 on CoLA despite using a 7B base model. ToRA obtained $68.20, 81.23, 81.60$ using OpenELM (270m) and 1B, 3B Llama-3.2.

Novikov et al. (2015) introduced tensor-train to neural networks and compressed the MLP in VGG by $200000\times$. Hrinchuk et al. (2019) use TT to compress the high rank *softmax* in a wide range of NLP models. Ma et al. (2019); Ben Noach and Goldberg (2020) tensorized the attention blocks.

CoShNet Ko et al. (2022) uses a `TT-linear` to represent a 1250x500 dense FC-layer factored into a TT that is $455\times$ smaller with little lost in performance. Similarly Wang et al. (2022) applied TT to compress LLM's attention and embedding layers. Success of Hrinchuk et al. (2019) and Wang et al. (2022) are notable given the difficulties outlined above when using low-rank matrix factoring for softmax and attention. TT's incredible ability to compress and preserve high-rank linear operators inspired us to use it in a parallel adapter in the manner of LoRA.

TT is great for naturally high-dimensional data. However, the matrices in a LLM are all 2D (embedding, K,Q,V, FFN, MLP). Keep in mind the number of terms in a TT factoring is determined by the input dimension d. To more fully exploit the strength of TT, ToRA use a `tensorization` process Garipov et al. (2016) to reshape the arrays into higher dimension tensors using `folding`. This enables ToRA to exploit higher *tt-rank* by feeding tensorized matrices to `tt-svd` producing a compact yet powerful TT layer. This gives ToRA greater expressiveness for the same budget.

`TT-svd` (Algorithm 1) is a stable algorithm to convert any dense tensor into a TT form. It is a sequence of QR factoring. QR factors a matrix into an orthonormal (Q) and upper-triangular (R). Each step keeps a truncated Q and folds the residual R into the next core. It produces a set of orthonormal subspaces. If the singular values are truncated at δ, the error of approximation will be $\sqrt{d-1}\delta$.

3.1 ToRA Adapter

This work set out to apply the power of tensor-train to LoRA fine-tuning and arrived at ToRA. ToRA is a LoRA style parallel adapter using TT to efficiently represent ΔW in a "no compromise" manner. Since TT can represent high rank subspaces with small number of weights, it can be used everywhere without searching for which layers to apply and without tuning their sizes. ToRA use LoRA-16 size as its parameter budget. Even better results are possible with some tuning of the `tt-rank`. See ranks ablation in Table (5).

Algorithm 1 TT-SVD

Initialization: Compute truncation parameter
$$\delta = \frac{\varepsilon}{\sqrt{d-1}}\|\mathbf{A}\|_F.$$
Temporary tensor: $\mathbf{C} = \mathbf{A}$, $r_0 = 1$.
for $k = 1$ to $d-1$ **do**
  $\mathbf{C} := \text{reshape}(\mathbf{C}, [r_{k-1}n_k, \frac{\text{numel}(\mathbf{C})}{r_{k-1}n_k}])$.
  Compute δ-truncated SVD: $\mathbf{C} = USV^\top + E$, where $\|E\|_F \le \delta$, $r_k = \text{rank}_\delta(\mathbf{C})$.
  Core: $G_k := \text{reshape}(U, [r_{k-1}, n_k, r_k])$.
  $\mathbf{C} := SV^\top$.
end for
$G_d = \mathbf{C}$.

Adapting W_V: ToRA systematically adapt K, Q, V for all attention blocks using the same `tt-config` with a tiny percentage of the full model's weights - e.g. 0.2% corresponding to LoRA-16. ToRA being able to adapt W_V and get good results can be easily missed. LoRA's original experiments tested several combinations of q, k, v, o while WeLore classified `attn.v_proj` as a Non-Low-Rank and do not attempt to adapt. One clue is the rightmost subfigure in Fig. (2) from WeLore. It shows `attn.v_proj` has even higher-rank than K,Q which present problems for all works that use the $BA^\top$ factoring. ToRA do not suffer from this limitation because TT can handle high-rank updates. ToRA also do not need to tune training epochs, batch size, learning rate etc. 1-5 epochs is used for all the tests.

3.2 Novel Adapter-`tt-svd` Initialization

Most deep networks that use TT all treat each `tt-core` as a regular matrix and initialize them separately using Xavier/Kaiming or random init including Jie and Deng (2023) and (Hrinchuk et al., 2019; Ma et al., 2019; Wang et al., 2022) etc. This might seems like a natural thing to do but this work present evidence that it is a critical error and limited their final performance even when TT was applied.

Bear in mind the Xavier analysis is based on fan-in/out for layers in a deep network in order to preserve the variance of their gradients. The cores in a TT are not operating on the fan-in/out connections except for the 1st and last core. Similar to CoShNet Ko et al. (2022), a scratch `nn.linear` is initialized with Kaiming. The `tt-svd` factors that into small `tt-cores` whose combined operations will be a Kaiming inited block. This is sufficient when TT is used as a linear/fully-connected layer. However using TT in an adapter has extra considerations. This bring us to a novel contribution for ToRA.

Adapter-`tt-svd` init LoRA's factors $BA^{\top}$ are initialized with "Noise and Zero" to match the gradients of FFT. ToRA developed a novel TT adapter initialization scheme that stay within the LoRA's BA framework by using `tt-svd` to initialize all but the last core which is zeroed out. Thus ToRA also produces the same gradient as the original pretained model and simulates FFT faithfully. Table (3) shows how critical Adapter-`tt-svd` is to ToRA's performance. For example, Kaiming vs. `tt-svd-only` vs. Adapter-`tt-svd` is $(69.4, 62.07, \mathbf{81.2})$ for CoLA and $(58.07, 56, \mathbf{75.06})$ for BoolQ on Llama-1B.

Survey conducted by the author[2] suggest engineers use $r \in [16..64]$ in modern applications. In this range, ToRA with same or smaller memory budget as LoRA consistently outperform it across all ranks, with and without quantization for all three models. Some by more than 10 percentage points (Table 1). ToRA also beat PiSSA by a large margin and sometimes even Dense. PiSSA relies on LoRA's low-rank factoring to capture the top-r principal components of W_0. While being faithful to the original model it lacks the random initialized A needed for an adapter to explore outside of the original distribution. Dense under-performing in many case can be explained as over-fitting due to the datasets for FT are often much smaller than the training corpus for LLMs. From Table 1, It is the top performer in only 2 cases that used OpenELM which has $270m$ parameters only. Other authors have noted the regularization effects of adapter FT especially using LoRA. ToRA strengthens the regularization of LoRA by representing a high-rank subspace in smaller number of weights.

QLoRA & QToRA: modern FT workflow invariably use QLoRA on a LLM to bring down the GPU memory required. QLoRA is a quantile-based quantizer that support high quality 4-bit FT and inference. QToRA replaced the LoRA adapter in QLoRA with ToRA. in Table (2) contains the superb results of QToRA on 2 datasets and 2 models.

[2]Thanks to [Redacted] of [Redacted]

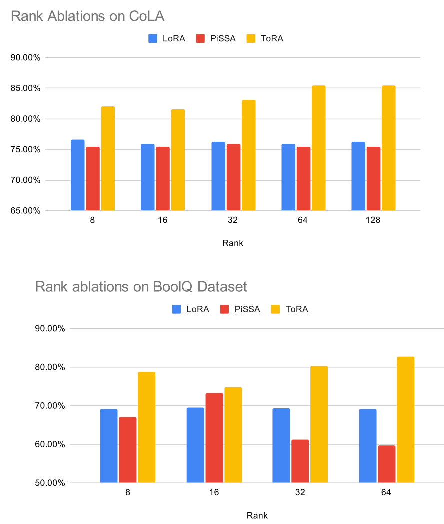


Figure 5: Rank ablations $8, 16, 32, 64, 128$ on BoolQ, CoLA using different adapters. ToRA is superior across the full range of ranks. ToRA-8 beats LoRA for all ranks even up to 128. BoolQ, rank 128 is missing because of GPU memory constraints.

4 Experiments

Table (1) summarized the results for 4 adapters and 3 models for 5 data sets. ToRA is compared next to LoRA and PiSSA using the same memory budget as well Dense ΔW using ADAM with lr $= 1e-5$, batch size 4, $r = 16$ and $\alpha = 16$ for 5 epochs unless otherwise stated. Dense is an adapter with full rank ΔW. Even though it is not a practical adapter it serves as the baseline for an adapter-based full FT whenever the GPUs allow. All experiments were carried out on 2 servers with 4 RTX 3090 24GB GPUs.

ToRA consistently outperforms LoRA and PiSSA with the same budget. In (Table 1) ToRA is compared against LoRA using Llama-3.2-3B e.g. $\mathbf{80.32}$ vs. 69.56 on BoolQ, $\mathbf{48.82}$ vs. 34.20 on MMLU, $\mathbf{81.60}$ vs. 75.86 for CoLA and $\mathbf{86.23}$ vs. 74.9 for MNLI.

For models, OpenELM 270M Mehta et al. (2024), instruction tuned LLama-3.1-1B, and LLama-3.2-3B (bf16) Dubey et al. (2024) on five diverse NLP datasets. They are chosen to test adapters across task types, difficulty levels and

Model	Method	%W	BoolQ	MMLU	CoLA	MNLI	GSM8k
OpenELM	Dense	10.24%	**63.81**	**28.98**	67.82	X	X
	PiSSA	0.26%	61.98	27.41	67.81	**80.40**	0.00
	LoRA-16	0.26%	62.71	24.28	**68.20**	79.30	0.00
	ToRA	0.20%	62.47	25.06	**68.20**	80.20	0.00
Llama-3.2-1B	Dense	7.53%	43.03	30.55	80.08	X	X
	PiSSA	0.19%	73.35	45.43	81.22	86.72	3.54
	LoRA-16	0.19%	72.86	**49.60**	78.93	**86.84**	5.31
	ToRA	0.19%	**74.81**	47.26	**81.23**	86.76	**18.58**
Llama-3.2-3B	Dense	12.06%	24.57	32.38	80.07	X	X
	PiSSA	0.20%	62.96	27.41	76.25	58.23	37.17
	LoRA-16	0.20%	69.56	34.20	75.86	74.90	37.17
	ToRA	0.20%	**80.32**	**48.82**	**81.60**	**86.23**	**38.94**

Table 1: ToRA vs. Dense ΔW, LoRA-16 and PiSSA for 3 different sized models. On OpenELM, ToRA is slightly better or matches LoRA. The gap become bigger as the base model is more powerful. For Llama-3.2-3B, ToRA is more than 10 points better for 3 datasets. The same trend is found in Table 3 for quantized base models. $\%W$ is the percentage of finetuned parameters. Dense uses batch size 1, Llama-3.2-3B is `bf16`. MMLU except for Dense use batch of 2 due to GPU memory constraints.

dataset sizes. Models are trained on the training set, and accuracy reported on the validation set to ensures consistency across different datasets (since many do not have a public test-set). ToRA does not require validation set based hyperparameter tuning.

Datasets: (1) **BoolQ** Clark et al. (2019) **B**inary (yes/no) QA dataset with 9.4k training and 3.2k validation samples. (2) **CoLA** Warstadt et al. (2019) **L**inguistic **A**cceptability dataset from published literature, part of GLUE Wang (2018), with 8.5k training and 516 validation samples. (3) **MMLU** Hendrycks et al. (2020) MCQ test across diverse subjects to assess world knowledge, with 99.8k training and 1.5k validation samples. (4) **MNLI** Williams et al. (2017) Multi-Genre NLI dataset covering 10 genres, with 393k training and 9.8k validation samples. (5) **GSM8k** Cobbe et al. (2021) **G**rade **S**chool **M**ath problem dataset for problem-solving evaluation, with 7.4k training and 1.3k test set samples.

4.1 Ablations

Base Models and Adapter Ablation with 3 pretrained base models (OpenELM, LLama-3.2 1B,3B) and 4 adapters (Dense, LoRA, PiSSA, ToRA) in Table (1) contain our main results. We consistently beat LoRA and PiSSA, sometimes by large margins on larger models (1B, 3B), while being neck-to-neck on OpenELM. We hypothesize the benefit of a good adapter is more pronounced for larger model because large model also has a large hidden-state. Unlike others that cites published results all results ran on the same platform with the same pipeline and adapted in the same manner.

Ranks $(8, 16, 32, 64, 128)$ for 3 adapters using Llama-3.2-3B in Figure (5) on CoLA and BoolQ are compared. LoRA shows limited ability to take advantage of higher ranks, ToRA displays a steady ability to improve. Remarkably, rank-8 ToRA beats LoRA and PiSSA for all ranks $8 - 128$.

Quantization ablation in Table (2). Three adapters are applied to two quantized models (Llama-3.2-1B and 3B) using two popular quantization methods QLoRA (Dettmers et al., 2024) and HQQ (Badri and Shaji, 2023), akin to many production deployments. In 3 out of 4 cases QToRA has the best results, the lone case of HQQ (4b) on CoLA goes to QPiSSA with QToRA a close second. Compare the corresponding rows for Table (1) and (2), one can see ToRA+LLama3.2-3B with QLoRA beats non-quantized for CoLA $(85.06$ vs. $81.60)$ and $(81.90$ vs $80.32)$ for BoolQ. The opposite is true for LLama3.2-1B. These unexpected results will be covered in a follow-up work.

Novel Adapter TT-SVD init: A novel TT initialization scheme "Adapter-`tt-svd`" is introduced above to initialize ToRA. Table (3) compares the performance of ToRA when initialized with standard Kaiming-Normal vs. `tt-svd`-only (without zero-last-core) and Adapter-`tt-svd`. For CoLA they are $(69.4, 62.07, \mathbf{81.2})$ and $(58.07, 56, \mathbf{75.06})$ for BoolQ on Llama-1B.

Model	Quantization	Adapter	CoLA	BoolQ
Llama-3.2-3B	QLoRA (4 bit)	QPiSSA	85.05	81.78
		QLoRA	83.91	81.54
		QToRA	**85.06**	**81.90**
Llama-3.2-1B	HQQ (4 bit)	QPiSSA	**81.23**	71.64
		QLoRA	78.54	72.37
		QToRA	80.84	**73.35**

Table 2: QLoRA or HQQ is applied to 2 models and 3 adapters. In 3 out of 4 case ToRA has the best performance and is always better than LoRA

Initialization	Model	CoLA	BoolQ
Kaiming		69.35	58.07
`tt-svd-only`	Llama-1B	62.07	56
`Adapter-tt-svd`		**81.23**	**75.06**

Table 3: TT init ablation: Kaiming, `tt-svd-only` and Adapter-`tt-svd` initialization for CoLA and BoolQ.

5 Conclusion

ToRA demonstrates greatly enhanced performance adapting attention blocks in a "no compromise" manner - i.e. adapting all attention blocks for all layers and for all (K,Q,V) matrices (Koohpayegani et al., 2023). ToRA performs better with larger models. While being competitive at the very low end, with or without quantization and use all ranks well. Thus ToRA can be applied easily in the field because it hardly needs any tuning. This gives practitioners a lot of freedom to choose what their budget permits. ToRA is a drop-in replacement for LoRA.

ToRA can contribute to adapters for ViT He et al. (2023) and diffusion models. Its benefits might even be more pronounced due to the inherit higher complexity/rank in these problems. This is left for future work.

6 Limitations

The experiments are limited to models with 3-billion parameters or less due to modest hardware (RTX3090) at our disposal.

Hyperparameter tuning the TT recipe could potentially result in better performance, we did not conduct it due to limited hardware budget. Instead we focus on using a configuration that matches LoRA's budget.

This work did not fully exploit the potential of ToRA applied to other parts of LLMs - e.g. `attn.proj_o`, `ffn`, MLP and embedding etc. Adapting attention blocks only follow LoRA's approach for ease of comparison. This is the current best practice. However, doubts linger in the community which ToRA helps to resolve.

While introducing ToRA to improve Parameter-Efficient Fine-Tuning (PEFT) can enhance model adaptability and efficiency, there are potential risks associated with publishing such methods. One major concern is the potential for adversarial exploitation, where malicious actors could use ToRA to create highly optimized yet harmful models at a lower computational cost. Additionally, making fine-tuning more efficient could lower the barrier for misuse, enabling the rapid adaptation of powerful models for unethical applications, such as misinformation generation or privacy-invasive surveillance. There is also a risk of unintentional bias amplification, as ToRA may introduce complex parameter interactions that are not well understood in terms of fairness and robustness. Addressing these risks requires transparency in research dissemination, rigorous evaluation against adversarial use cases, and responsible AI deployment strategies.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *ACL*, pages 7319–7328.

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. *arXiv:2312.11514*, page 12.

Hicham Badri and Appu Shaji. 2023. Hqq: Half-quadratic quantization of large machine learning models.

Matan Ben Noach and Yoav Goldberg. 2020. Compressing pre-trained language models by matrix decomposition. In *IJCNLP*, pages 884–889. ACL.

Daniel Bershatsky, Daria Cherniuk, Talgat Daulbaev, Aleksandr Mikhalev, and Ivan Oseledets. 2024. LoTR: Low Tensor Rank Weight Adaptation. *arXiv:2402.01376*, page 15.

Patrick H Chen, Si Si, Yang Li, Ciprian Chelba, and Cho Jui Hsieh. 2018. GroupReduce: Block-wise low-rank approximation for neural language model shrinking. In *NIPS '18*.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. 2024. Fira: Can We Achieve Full-rank Training of LLMs Under Low-rank Constraint? *Preprint*, arXiv:2410.01623.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Vin de Silva and Lek-Heng Lim. 2008. Tensor rank and the ill-posedness of the best low-rank approximation problem. *Preprint*, arXiv:math/0607647.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *NeurIPS*, 36.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. 2016. Ultimate Tensorization: Compressing Convolutional and FC layers Alike. *arXiv:1611.03214*, page 6.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. 2018. Gradient Descent Happens in a Tiny Subspace. *arXiv:1812.04754*, page 19.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. LoRA+: Efficient Low Rank Adaptation of Large Models. *Preprint*, arXiv:2402.12354.

Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. 2023. Parameter-efficient model adaptation for vision transformers. *Preprint*, arXiv:2203.16329.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. *arXiv:1902.00751*, page 12.

Ignacio Hounie, Charilaos Kanatsoulis, Arnuv Tandon, and Alejandro Ribeiro. 2024. LoRTA: Low Rank Tensor Adaptation of Large Language Models. *arXiv:2410.04060*, pages 1–17.

Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, Elena Orlova, and Ivan Oseledets. 2019. Tensorized Embedding Layers for Efficient Model Compression. *arXiv:1901.10787*, page 13.

Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaption of Large Language Models. In *ICLR*, page 26.

Wei Huang, Xingyu Zheng, Xudong Ma, Haotong Qin, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. 2024. An empirical study of llama3 quantization: From llms to mllms. *Preprint*, arXiv:2404.14047.

Brian Hutchinson, Mari Ostendorf, and Maryam Fazel. 2011. Low rank language models for small training sets. *IEEE Signal Processing Letters*, 18(9):489–492.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *ICLR '17*.

Ajay Jaiswal, Lu Yin, Zhenyu Zhang, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. 2024. From GaLore to WeLore: How Low-Rank Weights Non-uniformly Emerge from Low-Rank Gradients. *Preprint*, arXiv:2407.11239.

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, and Fuzhen Zhuang. 2024. MoRA: High-Rank Updating for Parameter-Efficient Fine-Tuning. *Preprint*, arXiv:2405.12130.

Shibo Jie and Zhi Hong Deng. 2023. FacT: Factor-Tuning for Lightweight Adaptation on Vision Transformer. *AAAI '23*, 37:1060–1068.

Reinhard Kneser. 1995. Improved Backing-Off for M-GRAM Language Modeling. In *ICASSP '95*, pages 181–184.

Manny Ko, Ujjawal K Panchal, Héctor Andrade-Loarca, and Andres Mendez-Vazquez. 2022. CoShNet: A Hybrid Complex Valued Neural Network using Shearlets. *arXiv:2208.06882*, page 16.

Tamara G Kolda and BW Bader. 2009. Tensor decompositions and applications. *SIAM review '09*, 51(3):455–500.

Soroush Abbasi Koohpayegani, KL Navaneet, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. 2023. NOLA: Networks as Linear Combination of Low Rank Random Basis. In *ICLR '23*, page 17.

Jean Kossaifi, Aran Khanna, Zachary C. Lipton, Tommaso Furlanello, and Anima Anandkumar. 2017. Tensor Contraction Layers for Parsimonious Deep Nets. In *CVPR '17*, pages 1940–1946. IEEE Computer Society.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the Intrinsic Dimension of Objective Landscapes. In *ICLR '18*, page 23.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2024. ReLoRA: High-Rank Training Through Low-Rank Updates. In *ICLR '24*, page 16.

Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, Bonita Bhaskaran, Bryan Catanzaro, Arjun Chaudhuri, Sharon Clay, Bill Dally, Laura Dang, Parikshit Deshpande, Siddhanth Dhodhi, Sameer Halepete, Eric Hill, Jiashang Hu, Sumit Jain, Ankit Jindal, Brucek Khailany, George Kokai, Kishor Kunal, Xiaowei Li, Charley Lind, Hao Liu, Stuart Oberman, Sujeet Omar, Ghasem Pasandi, Sreedhar Pratty, Jonathan Raiman, Ambar Sarkar, Zhengjiang Shao, Hanfei Sun, Pratik P Suthar, Varun Tej, Walker Turner, Kaizhe Xu, and Haoxing Ren. 2024. ChipNeMo: Domain-Adapted LLMs for Chip Design. *arXiv:2311.00176*, page 23.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Dawei Song, and Ming Zhou. 2019. A tensorized transformer for language modeling. *Preprint*, arXiv:1906.09777.

Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, et al. 2024. Openelm: An efficient language model family with open-source training and inference framework. *arXiv e-prints*, pages arXiv–2404.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024a. Pissa: Principal singular values and singular vectors adaptation of large language models. *Preprint*, arXiv:2404.02948.

Xiangdi Meng, Damai Dai, Weiyao Luo, Zhe Yang, Shaoxiang Wu, Xiaochen Wang, Peiyi Wang, Qingxiu Dong, Liang Chen, and Zhifang Sui. 2024b. PeriodicLoRA: Breaking the Low-Rank Bottleneck in LoRA Optimization. *arXiv:2402.16141*, page 11.

Mahdi Nikdan, Soroush Tabesh, Elvir Crnčević, and Dan Alistarh. 2024. Rosa: accurate parameter-efficient fine-tuning via robust adaptation. In *ICML*, ICML'24. JMLR.org.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. 2015. Tensorizing Neural Networks. In *NIPS '15*, pages 1–9.

I. V. Oseledets. 2011. Tensor-train decomposition. In *SIAM J. on Scientific Computing*, volume 33, pages 2295–2317.

Tara N. Sainath, Brian Kingbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-Rank Matrix Factorization for Deep Neural Network Training with High-Dimensional Output Targets. In *ICASSP' 13*, pages 6655–6659.

Sheng Shen, Dong Zhen, Jiayu Ye, Linjian Ma, Zhewei Yao, Asghar Gholami, Michael Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. *AAAI*, 34:8815–8821.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2024. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. *Preprint*, arXiv:2311.03285.

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Benyou Wang, Yuxin Ren, Lifeng Shang, Xin Jiang, and Qun Liu. 2022. Exploring extreme parameter compression for pre-trained language models. In *ICLR '22*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Cola: The corpus of linguistic acceptability (with added annotations).

Yeming Wen and Swarat Chaudhuri. 2024. Batched low-rank adaptation of foundation models. In *ICLR*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Wenhan Xia, Chengwei Qin, and Elad Hazan. 2024. Chain of LoRA: Efficient Fine-tuning of Language Models via Residual Learning. *Preprint*, arXiv:2401.04151.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. 2018. Breaking the Softmax Bottleneck: A High-Rank RNN Language Model. *arXiv:1711.03953*, page 18.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. *ICML '24*, page 23.