GoRA: Gradient-driven Adaptive Low Rank Adaptation

Haonan He[1,2,3] Peng Ye[3,4,5] Yuchen Ren[3,6] Yuan Yuan[2]
Luyang Zhou[7] Shucun Ju[7] Lei Chen[2]

[1]University of Science and Technology of China

[2]Institute of Intelligent Machines, HFIPS, CAS

[3]Shanghai Artificial Intelligence Laboratory

[4]Fudan University

[5]The Chinese University of Hong Kong

[6]University of Sydney

[7]Anhui Disaster Warning & Agrometeorological Information Center

NeurIPS 2025 (Submission)

Introduction & Motivation

The Problem: Full Fine-Tuning is Expensive

- Full Fine-Tuning (FFT) of Large Language Models (LLMs) is memory-intensive.
- An Adam optimizer requires $\approx 16\phi$ bytes of memory for ϕ parameters.

Existing Solution: LoRA

- Low-Rank Adaptation (LoRA) freezes pre-trained weights W_0 and trains low-rank adapters $\Delta W = sAB$.
- This significantly reduces the memory for optimizer states.
- **The Gap:** LoRA's performance on complex tasks (e.g., math, code) still lags behind FFT.

Two Key Challenges of LoRA

1. Rank Selection

- **Challenge:** Higher rank $\rightarrow$ better performance, but also $\rightarrow$ more memory and overhead.
- **Existing Fix:** Methods like AdaLoRA dynamically mask ranks.
- **Flaw:** These methods require larger matrices (e.g., 1.5x parameters) to reserve space, limiting the max rank and increasing overhead.

2. Initialization

- **Vanilla LoRA:** Initializes $B = 0$ to ensure $\Delta W = 0$ at step 0.
- **Non-Zero Init (PiSSA, LoRA-GA, etc.):** $A_0 B_0 \neq 0$.
- **CRITICAL FLAW:** All existing non-zero methods *must manipulate the pre-trained weights* ($W'_0 = W_0 - A_0 B_0$).
- This creates a **training-inference gap** and sacrifices LoRA's key benefits: minimal storage and easy multi-adapter serving.

Our Solution: GoRA

Core Idea: View LoRA as a Gradient Compressor

Inspired by LoRA-FA, we hypothesize that LoRA adapters act as compressors for the gradients $\frac{\partial \mathcal{L}}{\partial \mathrm{W}_0}$.

GoRA: Gradient-driven Adaptive Low Rank Adaptation

GoRA is a unified framework that uses gradients to **simultaneously** adapt both rank and initialization.

1. **Pre-computation:** Briefly compute N-batch accumulated gradients G before training.
2. **Adaptive Rank Allocation:** Use G and W_0 to assess layer importance and allocate a specific rank r^i to each layer.
3. **Adaptive Initialization:** Use G to initialize B_0, "priming" the model for optimization.

Our Solution: GoRA

Core Idea: View LoRA as a Gradient Compressor

Inspired by LoRA-FA, we hypothesize that LoRA adapters act as compressors for the gradients $\frac{\partial \mathcal{L}}{\partial \mathrm{W}_0}$.

GoRA: Gradient-driven Adaptive Low Rank Adaptation

GoRA is a unified framework that uses gradients to **simultaneously** adapt both rank and initialization.

1. **Pre-computation:** Briefly compute N-batch accumulated gradients G before training.
2. **Adaptive Rank Allocation:** Use G and W_0 to assess layer importance and allocate a specific rank r^i to each layer.
3. **Adaptive Initialization:** Use G to initialize B_0, "priming" the model for optimization.

Key Advantage: GoRA achieves a data-driven, non-zero initialization **without manipulating the pre-trained weights** W_0.

GoRA Method: 1. Adaptive Rank Allocation

Step 1: Compute Importance

We use the pre-computed gradient G to find importance scores for weights:

$$I(\mathrm{W}) = \mathrm{avg}\left(|\mathrm{W} \odot \mathrm{G}|\right)$$

Normalize scores to get an "advantage" a^i for i-th weight:

$$a^i = \frac{I(\mathrm{W}_0^i)}{\sum_{i=1}^{N} I(\mathrm{W}_0^i)}$$

Step 2: Allocate Rank

Distribute a total parameter budget b (based on a reference rank r^{ref}):

$$r^i = \mathrm{clip}\left(\left[\frac{b \cdot a^i}{\sqrt{m+n}}\right], r^{\min}, r^{\max}\right)$$

GoRA Method: 2. Adaptive Initialization

Goal: Approximate the Gradient

We want our initial adapter state A_0B_0 to be the best low-rank approximation of the gradient G.

Step 1: Initialize A

Initialize A_0 using a standard method (e.g., Kaiming Uniform).

Step 2: Initialize B (The Key)

We solve for B_0 using the Moore-Penrose pseudo-inverse:

$$B_0 = -(A_0^\top A_0)^{-1} A_0^\top G$$

We apply a scaling factor γ to control the magnitude:

$$W_0 + \frac{\alpha}{\sqrt{r}} A_0(\xi B_0) \approx W_0 - \gamma G$$

Experimental Setup

Tasks and Models

- **NLU:** T5-Base on 5 GLUE tasks.
- **NLG:** Llama-3.1-8B & Llama-2-7B on MTBench (Chat), GSM8k (Math), and HumanEval (Code).
- **CV:** CLIP-ViT-B/16 on 7 image classification tasks.

Baselines

We compare against a comprehensive set of baselines:

- Full Fine-Tuning (FFT)
- Vanilla LoRA
- Convergence-Optimized: rsLoRA, DoRA, LoRA+
- Init-Optimized: PiSSA, OLoRA, LoRA-GA
- Adaptive: AdaLoRA

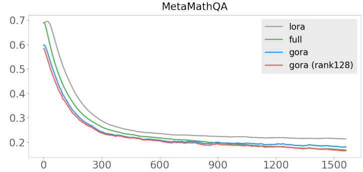
Results: NLG (Llama-3.1-8B)

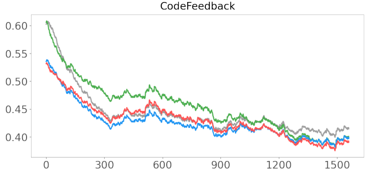
Method	**MTBench**	**GSM8k**	**HumanEval**
Full	5.88	73.69	51.63
LoRA	6.15	67.78	43.09
rsLoRA	6.18	68.36	45.78
DoRA	6.24	69.17	43.70
LoRA+	**6.35**	71.29	44.51
PiSSA	6.08	68.56	44.10
LoRA-GA	5.99	71.39	43.29
AdaLoRA	6.19	70.63	41.46
GoRA (r^{ref}=8)	6.34	**72.91**	**48.98**
GoRA (r^{ref}=32)	6.21	75.59	51.22
GoRA (r^{ref}=128)	5.82	**75.74**	**52.03**

Key Takeaways:

- At $r^{ref} = 8$, GoRA beats all baselines on GSM8k and HumanEval.

- At $r^{ref} = 128$, GoRA **surpasses Full Fine-Tuning** on complex math and code tasks.





GoRA shows lower start loss and faster convergence.

Results: NLU (T5-Base) & CV (CLIP)

T5-Base on GLUE (Average Score)

- **GoRA: 87.96**
- Full Fine-Tuning: 87.91
- LoRA-GA (2nd best): 87.77
- LoRA: 82.08

Results: NLU (T5-Base) & CV (CLIP)

T5-Base on GLUE (Average Score)

- **GoRA: 87.96**
- Full Fine-Tuning: 87.91
- LoRA-GA (2nd best): 87.77
- LoRA: 82.08

GoRA outperforms all baselines and even FFT on average.

CLIP-ViT-B/16 on 7 Tasks (Average Score)

- **GoRA: 89.47**
- LoRA-Pro (2nd best): 89.20
- LoRA-GA: 88.51
- Full Fine-Tuning: 88.06

Results: NLU (T5-Base) & CV (CLIP)

T5-Base on GLUE (Average Score)

- **GoRA: 87.96**
- Full Fine-Tuning: 87.91
- LoRA-GA (2nd best): 87.77
- LoRA: 82.08

GoRA outperforms all baselines and even FFT on average.

CLIP-ViT-B/16 on 7 Tasks (Average Score)

- **GoRA: 89.47**
- LoRA-Pro (2nd best): 89.20
- LoRA-GA: 88.51
- Full Fine-Tuning: 88.06

GoRA again outperforms FFT and all baselines, showing cross-modality effectiveness.

Ablation Studies

Rank Allocation Strategy

A wide adaptive range (e.g., $r = 4$–32) **beats** a fixed rank ($r = 8$).

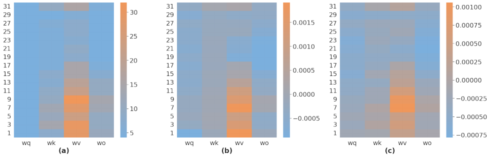
This confirms the adaptive strategy is effective.

Initialization Strategy

The scaling factor γ is crucial.

$\gamma = 0$ (no init) performs worst.

This confirms the gradient-based initialization provides a strong "kick-start".



GoRA allocates rank heterogeneously. Most ranks go to `wv` layers, fewest to `wq`.

Importance Metric

Our metric avg($|W \odot G|$) clearly outperforms others (e.g., $\|G\|_*$).

Conclusion & Key Contributions

Key Contributions

1. We identify critical limitations in existing LoRA variants:
   - Adaptive rank methods add overhead.
   - Non-zero initializations create a training-inference gap.
2. We propose **GoRA**, a unified framework that uses gradients to:
   - **Adaptively Allocate Rank** based on weight importance.
   - **Adaptively Initialize Adapters** via pseudo-inverse, *without* manipulating W_0.
3. **GoRA consistently outperforms** strong baselines and even **Full Fine-Tuning** on complex tasks, while preserving the efficiency of LoRA.

Additional Findings

- **QGoRA:** GoRA is compatible with quantization and outperforms QLoRA.
- **Usability:** GoRA's overhead is negligible, and it includes auto-tuning