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

Amit Kumar

AI/NLP Engineer at E42.ai

Table of contents

1. Introduction

2. GaLore: Gradient Low-Rank Projection

3. ADAM with GaLore

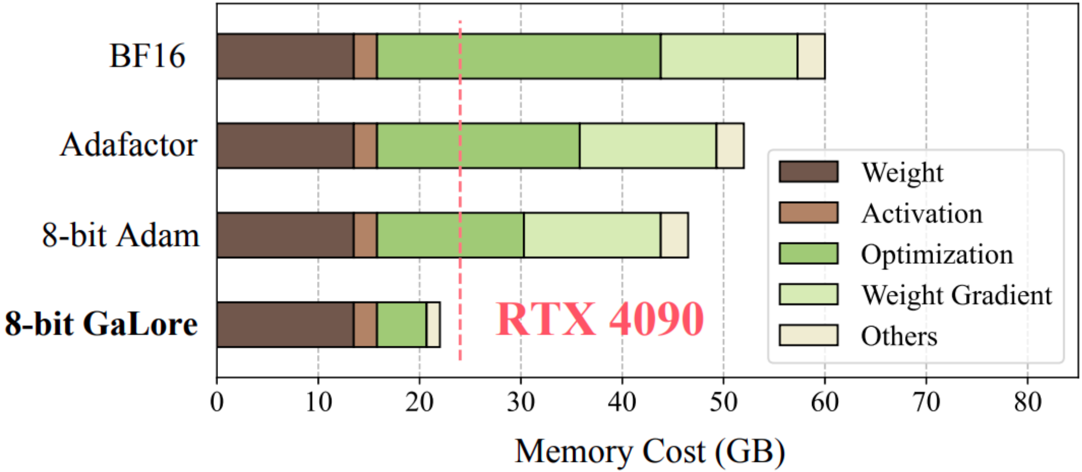
4. Experimental Results

5. Conclusion

- Pre-training a LLaMA 7B model from scratch with a single batch size necessitates a minimum of 58 GB memory allocation.
- This breakdown includes 14GB for trainable parameters, 42GB for Adam optimizer states and weight gradients, and 2GB for activations.
- Consequently, conducting such training is impractical on consumer-level GPUs like the NVIDIA RTX 4090, which offers 24GB memory capacity.

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1) G_t$$
$$V_t = \beta_2 V_{t-1} + (1 - \beta_2) G_t^2$$
$$\tilde{G}_t = M_t / \sqrt{V_t + \epsilon}$$

Memory allocation for LLaMa 7B



Exploring LORA: Constraints and Challenges

- Low-Rank Adaptation reparameterizes weight matrix $W \in R_{m\times n}$, into $W = W_0 + BA$, where W_0 is a frozen full-rank matrix and $B \in R_{m\times r}$, $A \in R_{r\times n}$ are additive low-rank adaptors to be learned.
- ReLoRA is also used in pre-training, by periodically updating W0 using previously learned low-rank adaptors.
- Drawback of LORA:
  1. the optimal weight matrices may not be low-rank.
  2. the reparameterization changes the gradient training dynamics.

GaLore: Gradient Low-Rank Projection

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t$$

Full Rank:

$$G_t = -\nabla_W \varphi_t(W_t) \in \mathbb{R}^{m \times n}$$

$$= W_0 + \eta \sum_{t=0}^{T-1} \rho_t(G_t)$$

GaLore:

$$\tilde{G}_t = P_t \rho_t(P_t^\top G_t Q_t) Q_t^\top$$

$\tilde{G}_t \in \mathbb{R}^{m \times n}$, $P_t \in \mathbb{R}^{m \times r}$, $Q_t \in \mathbb{R}^{n \times r}$

$$G_t = USV^\top \approx \sum_{i=1}^{r} s_i u_i v_i^\top$$

$$P_t = [u_1, u_2, ..., u_r], \quad Q_t = [v_1, v_2, ..., v_r]$$

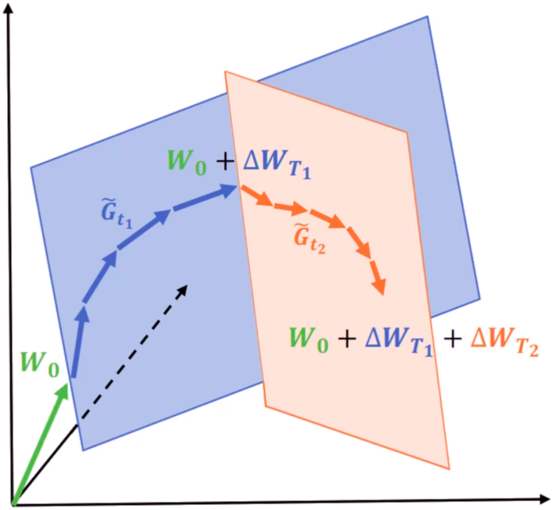
GaLore:

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t$$

$$\tilde{G}_t = P_t \rho_t (P_t^\top G_t Q_t) Q_t^\top$$

$$W_t = W_0 + \Delta W_{T_1} + \Delta W_{T_2} + \ldots + \Delta W_{T_n}$$

$$\Delta W_{T_i} = \eta \sum_{t=0}^{T_i - 1} \tilde{G}_t$$



Algorithm 2: Adam with GaLore

Input: A layer weight matrix $W \in \mathbb{R}^{m \times n}$ with $m \leq n$. Step size η, scale factor α, decay rates β_1, β_2, rank r, subspace change frequency T.

Initialize first-order moment $M_0 \in \mathbb{R}^{n \times r} \leftarrow 0$
Initialize second-order moment $V_0 \in \mathbb{R}^{n \times r} \leftarrow 0$

Initialize step $t \leftarrow 0$

repeat

$\quad G_t \in \mathbb{R}^{m \times n} \leftarrow -\nabla_W \varphi_t(W_t)$

$\quad$ **if** $t \bmod T = 0$ **then**

$\quad\quad U, S, V \leftarrow \text{SVD}(G_t)$

$\quad\quad P_t \leftarrow U[:, :r]$ {Initialize left projector as $m \leq n$}

$\quad$ **else**

$\quad\quad P_t \leftarrow P_{t-1}$ {Reuse the previous projector}

$\quad$ **end if**

$\quad R_t \leftarrow P_t^\top G_t$ {Project gradient into compact space}

$\quad$ UPDATE(R_t) **by Adam**

$$M_t \leftarrow \beta_1 \cdot M_{t-1} + (1 - \beta_1) \cdot R_t$$
$$V_t \leftarrow \beta_2 \cdot V_{t-1} + (1 - \beta_2) \cdot R_t^2$$
$$M_t \leftarrow M_t / (1 - \beta_1^t)$$
$$V_t \leftarrow V_t / (1 - \beta_2^t)$$
$$N_t \leftarrow M_t / (\sqrt{V_t} + \epsilon)$$

$\quad \tilde{G}_t \leftarrow \alpha \cdot P N_t$ {Project back to original space}

$\quad W_t \leftarrow W_{t-1} + \eta \cdot \tilde{G}_t$

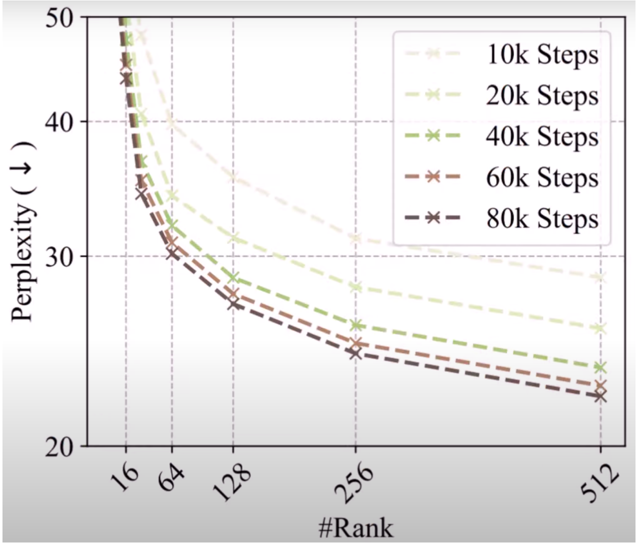
$\quad t \leftarrow t + 1$

until convergence criteria met

return W_t

Table 1: Comparison between GaLore and LoRA. Assume $W \in \mathbb{R}^{m \times n}$ $(m \leq n)$, rank r.

	GaLore	LoRA
Weights	mn	$mn + mr + nr$
Optim States	$mr + 2nr$	$2mr + 2nr$
Multi-Subspace	✓	✗
Pre-Training	✓	✗
Fine-Tuning	✓	✓



Ablation study of GaLore

Conclusion

1. GaLore significantly reduces memory usage by up to 65.5% in optimizer states while maintaining both efficiency and performance for large-scale LLM pre-training and fine-tuning.
2. Training with a rank of 128 using 80K steps achieves a lower loss than training with a rank of 512 using 20K steps. This shows that GaLore can be used to trade-off between memory and computational cost.
3. This can help us solve many current challenges we're facing in the existing architecture on the platform, such as summarization, querying a large number of knowledge bases at once with efficient memory requirements, and fewer components.

Thank You