Better Call LoRA

Robert Trifan Stefan Popa

University of Bucharest

Abstract

Low-rank adaptation (LoRA) has become a lightweight alternative to full fine-tuning for large language models. In this work, we benchmark vanilla LoRA and four recent LoRA variants: swapped-init LoRA (A = 0, B $\sim \mathcal{U}(-0.01, 0.01)$), LoRA-XS ($A \cdot R \cdot B$ factorisation), LoRA+ ($\eta_A \neq \eta_B$ learning rate scaling) and PiSSA-initialised LoRA (SVD(W) warm-start) on the TinyLlama-1.1 B backbone and the GLUE SST-2 sentiment-classification task. All experiments were performed on a single RTX 2070 (8 GB), enforcing strict memory budgets. We report classification accuracy, macro-F1, wall-clock training time and peak GPU memory to highlight the trade-offs each variant offers under resource constrained conditions.

1 Introduction

Large language models (LLMs) have rapidly become the de facto standard for various natural language processing tasks, ranging from search and dialog to code generation and summarization. Their ubiquity is largely attributed to their capacity to learn from vast amounts of data, but this comes at a cost: the computational resources required for training and fine-tuning these models are substantial.

Training or fully fine-tuning these models typically implies billions of parameters, weeks of GPU time and significant energy consumption. Such costs are often out of reach for many practitioners, leading to a growing interest in more efficient alternatives.

Low-rank adaptation (LoRA) [4] tackles this challenge by injecting a pair of low-rank matrices into the weights of a pre-trained model and learning only those additional parameters while freezing the original model. Despite the simplicity of the approach, LoRA has matched or even surpassed the performance of full fine-tuning on many tasks, while reducing the memory footprint and training time by orders of magnitude.

Building on top of this idea, a growing literature has emerged, proposing improvements that modify the initialization, learning rates schedules or factorisation structure of the low-rank matrices hoping to further enhance the effectiveness of LoRA. While these extensions are promising, their practical impact remains unclear due to inconsistencies in experimental setups.

This survey offers a comprehensive overview of five LoRA variants, including the original LoRA, and benchmarks them on a sentiment classification task using the TinyLlama-1.1B model [7] and the GLUE SST-2 dataset [6]. By standardizing the model size, dataset and evaluation metrics, we aim to provide an apples-to-apples comparison of these methods under realistic resource constraints.

2 Setup

2.1 Model

We employ the `TinyLLaMA-1.1B-Chat-v1.0` model from HuggingFace, a 1.1B parameter decoder-only transformer trained on 3T tokens. The architecture features 22 layers, 32 attention heads and an embedding size of 2048. The dataset used during pretraining consists of 950B tokens from a mixture of Slimpajama (excluding GitHub) and Starcoderdata (code only), with a natural language to code ratio of 7:3. Total training spanned 1.43M steps (just over 3 epochs). The model's scale relative to downstream datasets like SST-2 motivates low-rank fine-tuning approaches.

2.2 Dataset

The General Language Understanding Evaluation (GLUE) benchmark is a collection of diverse natural language understanding tasks designed to evaluate and compare the performance of language models across multiple domains. For downstream evaluation and fine-tuning, we use the Stanford Sentiment Treebank v2 (SST-2) from GLUE. SST-2 is a binary sentiment classification task over natural language movie reviews, for example:

- **Sentence:** "klein, charming in comedies like american pie and dead-on in election"
- **Label:** positive

The dataset contains 70,042 samples in total: 67,349 in the train set, 872 in the validation set, and 1,821 in the test set (labels hidden). From this, we use 20,000 samples for training and the full validation set of 872 samples. The total number of tokens in the training set is 760,459, which is kept consistent across all runs.

As per the label distribution, the training set is slightly asymmetrical, as it contains 8,940 negative samples and 11,060 positive samples. In contrast, the validation label distribution is nearly balanced, with 428 negative and 444 positive sentences.

2.3 Training

During training, we cast SST-2 as a single-turn instruction-following language-model task. Formally, for every sentence *sentence* and label *sentiment*, we build the prompt `Classify the sentiment of this sentence: <sentence>\n\n.Sentiment: <sentiment>\n`. The tokenized prompt is fed to the model as both `input_ids` and `labels` to ensure the model learns to predict the sentiment label directly. All pretrained model weights stay frozen by setting `requires_grad=False`, while the LoRA parameters are trained with a batch size of 1 and gradient accumulation of 16. Given TinyLLaMA's 1.1B parameters, the dataset is orders of magnitude smaller, both in tokens and examples. As such, we fine-tuned for a single epoch, since training beyond that risks overfitting.

2.4 Evaluation

At validation time, we feed the same prompt *without* the sentiment token `Classify the sentiment of this sentence: <sentence>\n\n.Sentiment:`. A single forward pass yields the logits for the final position. We verified that each label corresponds to a single token (e.g., " positive" and " negative") to ensure a valid head-to-head comparison in the final logit vector. We manually extract the token for each sentiment and look up its index in the model's vocabulary. The prediction is the index of the sentiment token with the highest logit.

2.5 Metrics

We evaluate each method using classification accuracy, macro F1-score, wall-clock training time, and peak GPU memory usage. Accuracy measures the proportion of correctly predicted sentiment labels, while macro F1-score does assumes class balance, which is compatible with the evaluation distribution presented above. We use Weights & Biases (wandb) to log all metrics in real time during training. Models are trained for 1250 steps with an evaluation round every 50 steps to monitor progression. Training time and peak GPU usage are measured from the beginning of the first step until the end of the last step, excluding model/tokenizer loading and initialization.

2.6 Hardware

All experiments were run on a single NVIDIA RTX 2070 with 8 GB of GPU memory. We enforced strict memory budgets to ensure that all methods could be compared under the same conditions. The training time was measured in wall-clock time, and the peak GPU memory usage was monitored throughout the training process.

3 Low Rank Adaptation

LoRA. Low-Rank Adaptation (LoRA) [4] freezes the pre-trained weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and learns a low-rank residual $\Delta W = AB^\top$ with $A \in \mathbb{R}^{d_{\text{out}} \times r}$ and $B \in \mathbb{R}^{d_{\text{in}} \times r}$, rank $r \ll \min(d_{\text{out}}, d_{\text{in}})$. During fine-tuning the forward pass uses $W + \Delta W$

while gradients flow only through A and B, reducing trainable parameters from $d_{\text{out}}d_{\text{in}}$ to $r(d_{\text{out}} + d_{\text{in}})$. Because the base weights never change, LoRA brings large memory savings by orders of magnitude (no optimizer states on W) and enables faster experimentation on consumer GPUs.

Swapped-init LoRA ($\mathbf{A = 0,\ B \sim \mathcal{U}}$). The original LoRA initialises A from a small uniform distribution and sets B to zero so that ΔW starts at 0. Hayou et al. [2] dive deeper into the intuition behind this choice, arguing that it's is indeed better to start from (A = 0, B $\sim \mathcal{U}$) instead of (A $\sim \mathcal{U}$, B = 0) and the reason behind this might be that the first initializatino allows the use of larger learning rates (without causing output instability) resulting in more efficient learning.

LoRA-XS. LoRA-XS [1] adds an intermediate low-rank matrix $R \in \mathbb{R}^{r\times r}$ and factorises the residual as $\Delta W = ARB^\top$. The extra r^2 parameters increase expressiveness while preserving the $\mathcal{O}(r(d_{\text{out}} + d_{\text{in}}))$ memory budget (for typical $r \ll d$ the r^2 term is negligible). Empirically, LoRA-XS allows for more expressive low-rank representations while maintaining the same memory footprint as vanilla LoRA.

LoRA+ [3] modifies the original LoRA method by applying different learning rates to the low-rank matrices A and B. Since $A \in \mathbb{R}^{d_{\text{out}}\times r}$ and $B^\top \in \mathbb{R}^{r\times d_{\text{in}}}$ have completely different shapes, the authors argue that their differing dimensionality and roles justify independent optimization dynamics. Empirically, using a higher learning rate for A (typically by a factor of 2-4 for LLaMA models) improves convergence and downstream performance. Note that in the LoRA+ paper the terminology is slightly different, with A and B swapped compared to the original LoRA paper.

PiSSA. PiSSA [5] (Principal Singular Values and Singular Vectors Adaptation of Large Language Models) introduces a singular value decomposition (SVD)-based initialization strategy for LoRA. Given a target weight matrix $W \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$, PiSSA computes its SVD: $W = U\Sigma V^\top$. The top-r singular components are then used to initialize the low-rank adaptation matrices A and B such that $\Delta W = AB^\top$, where $A = U_{[:,:r]}\Sigma^{1/2}_{[:r,:r]} \in \mathbb{R}^{d_{\text{out}}\times r}$ and $B = \Sigma^{1/2}_{[:r,:r]}V^\top_{[:,:r]}$. Furthermore, before freezing, the original weight matrix W is replaced with the rest of the singular components, i.e., $W = U_{[:,r:]}\Sigma_{[r:,r:]}V^\top_{[:,r:]} \in \mathbb{R}^{m\times n}$. By $\Sigma^{1/2}$ we mean the square root of the diagonal matrix Σ and the slicing notations are consistent with PyTorch's tensor indexing. This initialization aligns the low-rank structure with the principal subspace of the original weight matrix, improving convergence speed and performance stability in downstream tasks.

4 Experiments

For each method, describe the hyperparameters explored with a table.

Gather the best results into a final table, comparing the methods.

5 Conclusion

Explain that, because of limited compute resources, we couldn't see meaningful results.

References

[1] Klaudia Baazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters, 2024.

[2] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. The impact of initialization on lora finetuning dynamics, 2024.

[3] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models, 2024.

[4] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[5] Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models, 2025.

[6] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[7] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.