LoRA - Limitations and Variants
A discussion of LoRA, DyLoRA, QLoRA, and LoRA+

Silvie Opolka (sopolka@uni-osnabrueck.de)
Argha Sarker (asarker@uni-osnabrueck.de)
Maria Oprea (moprea@uni-osnabrueck.de)

Supervision

– Prof. Dr. Elia Bruni (elia.bruni@uni-osnabrueck.de)

Context

Pre-trained Large Language Models (LLMs) often contain large parameter matrices, e.g. query (W_q), key (W_k), and value (W_v) matrices in a Transformer-based model. In full fine-tuning, all parameters of such a pre-trained model are considered, and this is for multiple epochs. This high training cost can only be covered by the big companies.
Parameter-efficient fine-tuning (PEFT) methods aim to increase the accessibility of LLM fine-tuning by making it more cost-efficient, while maintaining a performance comparable to full fine-tuning. **Low-Rank Adaption (LoRA)** and its variants fall into the 'Reparameterized Fine-tuning' category of PEFT, i.e. the number of trainable parameters is reduced via low-rank transformation on the large weight matrices.[1]
There are various PEFT methods developed already. This project report discusses LoRA[2], and three LoRA variants (DyLoRA[3], QLoRA[4], and LoRA+[5]) that each cover an essential limitation of vanilla LoRA. Figure 1 gives an overview of the variants discussed in this project report.

LoRA

Previous findings show that when fine-tuning a pre-trained model, it resides on a low intrinsic dimension, and learns efficiently, despite random projection into this smaller subspace. Based on those findings, LoRA assumes that the *change* in weights during fine-tuning has a low 'intrinsic rank' as well.[2]
Every matrix has a **rank** r, which is the total amount of linearly independent columns or rows. It can be proven that the column rank is equal to the row rank. A column/row is linearly independent if it can't be represented as a combination of other columns/rows in the matrix. A matrix has a so-called '**intrinsic rank**' if there are dependent columns/rows, i.e. columns/rows that can be removed

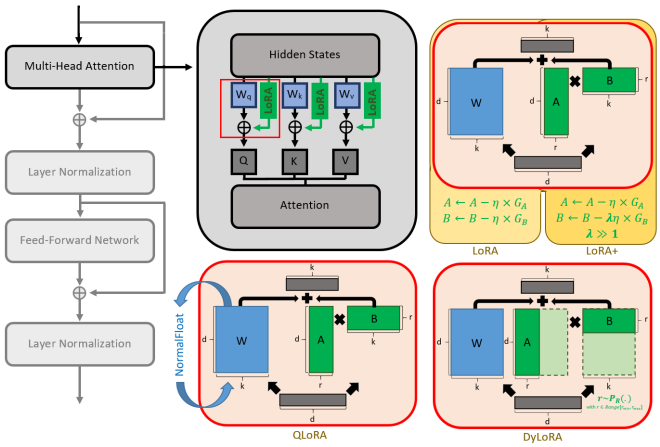

Figure 1: Exemplary application of LoRA variants in transformers.

from the matrix without losing information. In **low-rank decomposition**, a lower-rank matrix is found that captures the essential information of the original matrix. In **Reparameterization**, the original weight matrix is transformed into its low-rank representation.

For each large parameter matrix W, i.e. the original, pre-trained parameters, LoRA introduces two trainable, low-rank matrices A and B, called the **LoRA adapter matrices**, which approximate W and represent its value updates ΔW. Matrix A is initialized randomly (sampled from a Normal distribution), and matrix B is initialized with zeros. Hence, $\Delta W = 0$ at the start of fine-tuning. During fine-tuning, W is frozen. Only the adapter parameters are updated.

In the forward pass, pre-trained and adapter matrices are merged to create the updated weight matrix. A **scaling factor** $\frac{\alpha}{r}$ determines how strongly the parameter changes contribute to the parameter update (when adding them to the original parameters).[1]

$$W = W + \frac{\alpha}{r} \cdot \Delta W = W + \frac{\alpha}{r} \cdot (A \times B)$$

$$\text{with } W \in \mathbb{R}^{d \times k},\ B \in \mathbb{R}^{d \times r} \text{ and } A \in \mathbb{R}^{r \times k}, \text{ where } r \ll \min(d, k)$$

Given an input x, which is similar for W and ΔW, the matrices are summed coordinate-wise to generate an output h:

$$h = x \times W + x \times \frac{\alpha}{r} \cdot \Delta W = x \times W + \frac{\alpha}{r} \cdot (x \times A \times B)$$

One of the most notable findings presented in the LoRA paper is that the number of trainable parameters can be reduced by 10,000 times and the GPU memory required can be reduced by 3 times. Furthermore, performance score analysis

[1] In the original LoRA paper, they do not treat α as a hyperparameter, but set it as a constant (the first r tried), because tuning α and the learning rate seem to be quite similar when optimizing with Adam.

shows that adapting more weight matrices with smaller ranks is preferable over adapting a single weight matrix with a larger rank. Furthermore, a small rank already performs competitively, suggesting that ΔW has a very low 'intrinsic rank', however this rank seems to differ between the weight matrices.[2] Lastly, ΔW seems to amplify sorely the features of W that are important for the specific downstream task it is fine-tuned for and that are not already emphasised in W.[2]
LoRA brings many advantages. First, it reduces the storage requirement for multi-task / task-switching models. The matrices A and B are trained and saved for each task individually, but the large pre-trained model only needs to be saved once and can be reused. Second, it reduces the hardware barrier by optimizing fewer parameters (typically 1% of the original model), eliminating the need to calculate gradients and maintain optimizer states for most parameters, thus making training more efficient. Third, due to the simple, linear design of LoRA, no additional inference latency is introduced. Fourth, LoRA is compatible with many parameter-efficient methods. Fifth, LoRA is generalised to full fine-tuning when setting r to the rank of the pre-trained weight matrices.[2]
Additionally, LoRA has several advantageous side effects. First, its structure allows for better interpretability of the correlation between pre-trained and updated weights. Second, like other PEFT methods, LoRA reduces the risk of catastrophic forgetting. It tends to preserve the knowledge captured by the pre-trained LLM because only a small set of trainable parameters gets updated during fine-tuning. Third, overfitting is circumvented by not updating the pre-trained parameters.[3][1][2]
The original LoRA paper only mentions two limitations directly. First, there are still uncertainties about where to best apply LoRA modules and how to combine them with other methods for efficiency adaptation.[2] Second, there are limitations when it comes to a multi-task learning setting. LoRA assumes that the inputs of a batch can be adapted using the same low-rank matrix. Hence, a challenge is introduced, when batching inputs of different tasks (which require different adapter matrices) in a single forward pass.[2]
However, more limitations of vanilla LoRA can be found when taking a look into the motivation of its adaptations.

DyLoRA

In vanilla LoRA, the pre-trained model is fine-tuned for a *single, fixed rank.* Finding the best rank for a specific fine-tuning task requires an *exhaustive search* and *no universal method* for determining the rank size has been found yet. Also, the fine-tuned model does not work well for inputs that require other rank values, i.e. if *multiple ranks* or a *change of rank* is required, a separate model needs to be fine-tuned.

[2] In their example, W_q seems to have a higher "intrinsic rank" ($r = 8$) than W_v ($r = 1$).

[3] Full fine-tuning might lead to overfitting if the dataset used for fine-tuning is much smaller than the one used for pre-training, which is usually the case.

Dynamic low-rank adapter (DyLoRA) addresses those limitations of LoRA.[3] Instead of learning representations of a single rank, it fine-tunes adapter matrices for a *range of ranks* $r \in [r_{min}, r_{max}]$. r_{min} and r_{max} are treated as hyper-parameters, and span the predefined *categorical distribution* P_R over all considered rank values. At each training iteration, r (with $r \in \{r_{min}, r_{min} + 1, ..., r_{max} - 1, r_{max}\}$) is *randomly selected* from P_R, and the adapter matrices A and B are *truncated* accordingly.[4] Only the truncated parameter matrices $A_r = A[1:r,:]$ and $B_r = B[:,1:r]$ are updated.

$$W = W + \frac{\alpha}{r} \cdot \Delta W = W + \frac{\alpha}{r} \cdot (A[1:r,:] \times B[:,1:r])$$

When updating all parameters in the truncated adapter matrices, the lower rank parameters are affected. To avoid this, the paper proposes **frozen DyLoRA**, where only the r^{th} corresponding row and column is updated.

$$W = W + \frac{\alpha}{r} \cdot \Delta W = W + \frac{\alpha}{r} \cdot (A[r,:] \times B[:,r])$$

An evaluation of performance across various datasets shows that DyLoRA archives nearly similar performance as LoRA for the specific rank LoRA was trained for, and significantly outperforms LoRA across a broader spectrum of ranks. Although frozen DyLoRA shows a slightly lower accuracy than DyLoRA, it still shows strong performance across ranks.
In summary, DyLoRA introduces multiple improvements: First, it is **search-free** as it avoids a costly search process to find the optimal rank, making the overall training faster. Second, it has **higher flexibility** as it performs consistently well on a much larger range of ranks. Third, it makes **training more efficient** because only a single model needs to be trained for multi-rank problems, while making only a negligible compromise in performance. Frozen DyLoRA can improve efficiency even further.

QLoRA

Quantization and Low-Rank Adapters (QLoRA) aims to reduce the memory footprint of fine-tuning LLMs.To fine-tune a LLAMA-65B model, LoRA requires 780GB of GPU memory, which is equivalent to 16 A40 GPUs. **Quantization** reduces the precision of the parameters in the model by mapping the original, high-precision parameters to a smaller set of low-precision values. QLoRA introduces a '**4-bit NormalFloat quantization**:
For *storage*, the pre-trained parameters of the model (w_W), typically 16-bit floating-point numbers, are quantized to 4-bit - a more compact format. However, the adapter parameters $w_{A,B}$ are retained in their original format to maintain the precision crucial for adapting the pre-trained model well to the specific

[4]This truncation is inspired by **nested dropout**, which is a stochastic regularization technique used in the training of autoencoders to enforce a representation order in the learning process. Given a randomly chosen unit k, all connections from 1^{st} to k^{th} unit are kept, while all connections larger then k are dropped.

task it's fine-tuned for. For *computations*, the 4-bit quantized parameters are dequantized to their original format.
To minimize the dequantization error, which results from the loss of information during quantization, QLoRA uses a special quantization approach, which includes a new data type '**NormalFloat**' and '**Double/Blockwise Quantization**'. Both aspects attempt to avoid *unnecessary* loss of information due to values not being uniformly distributed across all bins. Firstly, parameter values in pre-trained neural networks are typically normally distributed and centered around zero. Instead of dividing the parameter values into evenly spaced blocks, '**NormalFloat** quantization' considers the parameters' normal distribution. To ensure that the parameters are distributed normally, w_W are normalized to have zero mean and unit variance before being quantized. Secondly, outliers might occur in the input tensor. Instead of mapping the parameter values to a bin directly, '**Double/Blockwise Quantization**' is applied to W. The input tensor is chunked into blocks[5], which are then quantized independently.
Generally speaking, QLoRA makes fine-tuning possible, even when only a few GPUs are available. However, if the model suddenly receives a very long input, the spike in sequence length would break training due to memory issues. QLoRA uses **page optimizers** to prevent those **memory spikes**: The state of the optimizer is moved from the GPU memory to the CPU memory until the long sequence is read. Then, when the GPU memory is free, the optimizer state is moved back to the GPU. In summary, due to the quantization of the high dimensional and partial redundant[6] parameter matrix W, less memory is required for fine-tuning tasks while keeping a similar performance level. This improvement is supported by the result that fine-tuning in less than twelve hours on a single consumer GPU led to a model reaching 97.8% of ChatGPT's performance level (Vicuna benchmark).[4]

LoRA+

In LoRA, there is **no principled guidance** for setting the learning rate, besides the decision to update the adapter matrices A and B with the **same learning rate** η. The motivation for LoRA+ is to provide specific guidelines on how to set the learning rates (η_A, η_B) of the adapter matrices, especially for models with a large width n (embedding dimension) like LLMs.
In order to provide such scaling guidelines, a mathematical analysis was conducted firstly on a linear model ($f(x) = (W^* + BA^\top)x$, with pre-trained $W^* \in \mathbb{R}^{1\times n}$ and adapter weights $B \in \mathbb{R}, A \in \mathbb{R}^n$) and then on a general neural network architecture, guided by the research question on how to set the learning rates of LoRA modules A and B with an increasing model width $n \to \infty$? [5].
In this context, theorem proving based on the theory of scaling for neural net-

[5] In the paper, they use a block size of 64.[4]
[6] W is pre-trained for many tasks, but we only require it to work for a small subset.

works[7] was employed to illustrate several key findings: Firstly, it demonstrated that LoRA performs suboptimally on models with a large embedding dimension. Secondly, the analysis inferred that having separate learning rates for the two modules can improve efficiency. Lastly, it derived that the learning rate for the adapter matrix B should be much larger than that of the adapter matrix A.
Therefore, LoRA+ introduces a **new guideline for η_A and η_B** that attempts to ensure stability and efficiency of LoRA fine-tuning in the infinite-width limit. Testing different combinations of learning rate pairs across four GLUE tasks (MNLI, QQP, SST-2, QNLI) with GPT-2 and RoBERTa showed that the highest test accuracies were achieved when η_B was **significantly larger** than η_A, particularly in 'harder' tasks such as MNLI and QQP. This particularity was attributed to harder tasks requiring more efficient feature learning. The idea of having a larger update for B is intuitive: Since B is initialized with zeros, it does not have a good starting point for learning the task-specific features from the data and could benefit from larger updates. To avoid additional computational costs by tuning two different learning rates, LoRA+ introduces a **fixed ratio** $\lambda = \frac{\eta_B}{\eta_A}$ ($\gg 1$) that is multiplied with the learning rate of one of the adapter matrices. Having a λ close to 1 would mirror the behavior of standard LoRA, therefore the selection of the ratio λ holds significant importance for LoRA+. In order to find an optimal value for λ, an empirical analysis was conducted by analyzing the distribution of the ratio $\frac{n_B}{n_A}$ for the top 4 learning rate pairs in terms of test loss for GPT-2 and RoBERTa on each of the four GLUE tasks. This analysis indicated that an appropriate value is $\lambda = 2^4$.
LoRA+ shows that having separate learning rates for the adapter matrices **improves feature learning abilities** (about 1-2%) and **fine-tuning speed** (up to $2X$ faster), especially for models with large embedding dimensions. However, it is acknowledged that there is a need for a more refined estimation of the optimal ratio considering task and model dependencies.

Discussion and Outlook

PEFT methods like LoRA and its variants greatly contribute to making fine-tuning LLMs more accessible, especially for entities with limited resources. The variants presented in this paper each address specific limitations of the original LoRA model, reflecting ongoing efforts to enhance its efficiency. Additionally, numerous other variants exist, each targeting different constraints or challenges. We believe that combining the variants discussed in this report, along with other present approaches holds great promise for advancing the field of parameter-efficient fine-tuning. Approaches to unifying multiple LoRA variants are already emerging with models like SuperLoRA[6], showing that this is a very valuable area of research. Exploring the potential combinations between different variants may lead to novel insights and techniques that further enhance the efficiency and effectiveness of fine-tuning strategies for LLMs.

[7]Process of increasing the size of the model's width or length to improve the model's performance.

References

[1] L. Xu, H. Xie, S. Joe Qin, X. Tao, and F. L. Wang, "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment." 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2312.12148

[2] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," no. 2, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2106.09685

[3] M. Valipour, M. Rezagholizadeh, I. Kobyzev, and A. Ghodsi, "Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2210.07558

[4] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.14314

[5] S. Hayou, N. Ghosh, and B. Yu, "Lora+: Efficient low rank adaptation of large models," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.12354

[6] X. Chen, J. Liu, Y. Wang, P. P. Wang, M. Brand, G. Wang, and T. Koike-Akino, "Superlora: Parameter-efficient unified adaptation of multi-layer attention modules," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.11887