ICIC 2024

APTIKOM
ASOSIASI PENDIDIKAN TINGGI INFORMATIKA DAN KOMPUTER

UPH
UNIVERSITAS PELITA HARAPAN

Towards Offline GenAI Fine Tuning Model with LoRA Derivatives for IoT Edge Server

P. Yugopuspito,
I Made Murwantara,
Evan K. Alim,
Wiputra Cendana,
Aditya R. Mitra

Universitas Pelita Harapan

Agenda

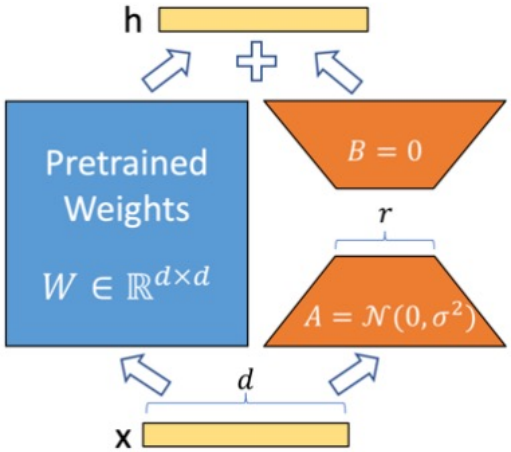
- Background
- LoRA's Core Principles
- LoRA Derivatives
  - Quantized LoRA (QLoRA)
  - Multi-task LoRA (MTLoRA)
  - Adaptive LoRA (AdaLoRA)
  - Accurate LoRA (IR-QLoRA)
- Results
- Further Research

Background

- The Internet of Things (IoT) has become ubiquitous, collecting and analyzing data from various sources.
- Dependence of constant internet connection makes its functionality unreliable, therefore research focused on offline artificial intelligence (AI) systems for IoT is highly needed.
- Generative AI (GenAI) has become a trendy topic since late 2022, and is expected to continue at least for the foreseeable future.
- Our research will focus on how to optimize GenAI so it will become usable on consumer devices.
- One of the possible optimizations is to minimize the memory usage of GenAI, especially during fine tuning.
- The most popular method is low-rank adaptation (LoRA), and it has given rise to lots of derivatives, the most popular of it being Quantized LoRA (QLoRA).

ICIC 2024

UPH
UNIVERSITAS PELITA HARAPAN



LoRA's core principles

Low-rank adaptation (LoRA) is a popular technique used during GenAI fine-tuning to improve the efficacy.

The core principles include:

- Parameter efficiency fine-tuning (PEFT), i.e. by fine-tuning only a small subset of the model's parameters, thereby greatly increasing efficacy.
- Low-rank decomposition, i.e. by decomposing the weight matrices into lower-rank matrices, which are easier to fine-tune.
- Modularity, which is useful for transfer learning, by only training specific parts of the model.
- Scalability, meaning that a linear increase on model size should result in only a sublinear increase of computational resources.
- Regularization, to prevent model overfitting.
- Generalization, for adapting to new tasks.

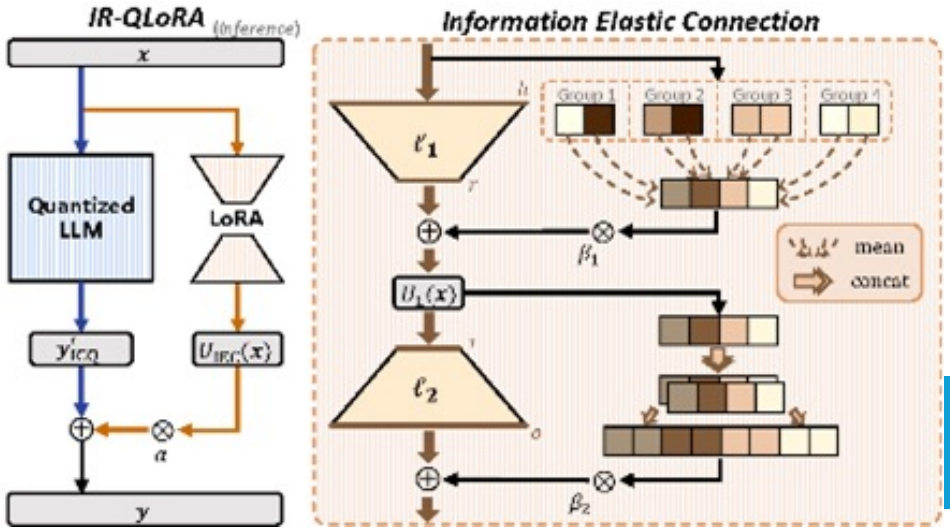
LoRA Derivatives

- Quantized LoRA (QLoRA) - most popular
- Multi-task LoRA (MTLoRA)
- Adaptive LoRA (AdaLoRA)
- Accurate LoRA (IR-QLoRA)
- Continual LoRA (C-LoRA)
- Quantized Dynamic LoRA (QDyLoRA)
- Residual LoRA (ResLoRA)
- Orthonormal LoRA (OLoRA)
- Weight-Decomposed LoRA (DoRA)
- High-rank Adaptation (HiRA) - intriguing

ICIC 2024

Quantized LoRA (QLoRA)

The most popular LoRA derivative, it includes quantizing the weights used for fine-tuning in order to minimize computational resources while maintaining LLM performance.



Algorithm 1 QLoRA fine-tuning algorithm

Input: - Input embedding $\mathbf{x} \in \mathbb{R}^d$ - Quantization levels L - Low-rank factorization parameters k - Quantization codebook $\mathbf{C} \in \mathbb{R}^{L \times d}$ - Low-rank weight matrix $\mathbf{W} \in \mathbb{R}^{k \times k}$ - Low-rank activation matrix $\mathbf{H} \in \mathbb{R}^{d \times k}$

Output: - Quantized embedding $\mathbf{z} \in \mathbb{R}^d$

1: Project the input onto the codebook: $\mathbf{v} = \mathbf{C}\mathbf{x}$
2: Quantize the projected vector: $\mathbf{q} = \text{Quantize}(\mathbf{v}, L)$
3: Embed the quantized vector: $\mathbf{z} = \mathbf{H}\mathbf{W}\mathbf{q}$

Quantization function

1: Define a quantization function $\text{Quantize}(\mathbf{v}, L)$
2: Find the nearest neighbor in the codebook: $\hat{i} = \arg\min_i ||\mathbf{v} - \mathbf{c}_i||^2$
3: Return the corresponding codeword: $\mathbf{q} = \mathbf{c}_{\hat{i}}$

Multi-task LoRA (MTLoRA)

It uses both task-agnostic modules and task-specific modules, in order to adapt from just learning a certain task to learning different but related tasks.

Algorithm 2 MTLoRA fine-tuning algorithm

Input: Model $\mathcal{M}$, tasks $\mathcal{T} = \{T_1, \ldots, T_N\}$, datasets $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$, optimizer $\mathcal{O}$, learning rate η
Output: Fine-tuned model $\mathcal{M}$
for t in 1 to E **do**
  Sample mini-batches $\mathcal{B}_i \subset \mathcal{D}_i$ for all tasks $T_i \in \mathcal{T}$
  Compute task-specific losses $L_i(\mathcal{M}, \mathcal{B}_i, T_i)$ for all tasks $T_i \in \mathcal{T}$
  Compute the overall loss $L(\mathcal{M}) = \sum_{i=1}^{N} L_i(\mathcal{M}, \mathcal{B}_i, T_i)$
  Update model parameters $\theta \leftarrow \mathcal{O}(\theta, \eta, \nabla L(\mathcal{M}))$
end for

Algorithm 3 MTLoRA Task-specific Loss ($L_i(\mathcal{M}, \mathcal{B}_i, T_i)$)

1: **Input:** Model $\mathcal{M}$, mini-batch $\mathcal{B}_i$, task T_i
2: **Return:** L_i
3: Compute model predictions $\hat{y}_i = \mathcal{M}(\mathcal{B}_i)$ for task T_i
4: Compute task-specific loss L_i based on $\hat{y}_i$ and ground truth labels in $\mathcal{B}_i$ (e.g., cross-entropy for classification)

ICIC 2024

Adaptive LoRA (AdaLoRA)

- AdaLoRA dynamically adjust the parameters' weights and the matrices' rank based on the model performance to focus only on the parameters that need improvement instead of training a fixed number of parameters on every step.

Input: Model $\mathcal{M}$, task $\mathcal{T}$, dataset $\mathcal{D}$, optimizer $\mathcal{O}$, learning rate η
Output: Fine-tuned model $\mathcal{M}$
Initialize task-specific importance weights $w_i = 1$ for all layers i
for t in 1 to E **do**
  Sample mini-batch $\mathcal{B} \subset \mathcal{D}$
  Compute model predictions $\hat{y} = \mathcal{M}(\mathcal{B})$ for task $\mathcal{T}$
  Compute task loss $L(\mathcal{M}, \mathcal{B}, \mathcal{T})$
  Backpropagate to compute gradients ∇L
  for i in all layers of $\mathcal{M}$ **do**
    Compute layer-wise importance score $s_i = \text{metric}(\nabla L_i)$ (e.g., magnitude of gradients)
    Update importance weight: $w_i \leftarrow \tau w_i + (1 - \tau) s_i$ (exponential moving average)
  end for
  Apply importance weights during gradient update: $\theta \leftarrow \mathcal{O}(\theta, \eta, w \odot \nabla L)$ where $\odot$ element-wise multiplication and $w = [w_1, \ldots, w_n]$ is the weight vector
end for

Accurate LoRA (IR-QLoRA)

- IR-QLoRA uses the Information Calibration Quantization (ICQ) and Information Elastic Connection (IEC) to preserve information in the quantized parameters while only having a minimum additional training time.

Algorithm 5 IR-QLoRA fine-tuning algorithm

Input: Model $\mathcal{M}$, tasks $\mathcal{T} = \{T_1, \ldots, T_N\}$, datasets $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$, optimizer $\mathcal{O}$, learning rate η
Output: Fine-tuned model $\mathcal{M}$
Initialize task-specific scaling factors $\gamma_i = 1$ for all tasks $T_i \in \mathcal{T}$
for t in 1 to E **do**
    Sample mini-batches $\mathcal{B}_i \subset \mathcal{D}_i$ for all tasks $T_i \in \mathcal{T}$
    for i in all tasks $T_i \in \mathcal{T}$ **do**
        Compute model predictions $\hat{y}_i = \mathcal{M}(\mathcal{B}_i)$ for task T_i
        Compute task-specific loss $L_i(\mathcal{M}, \mathcal{B}_i, T_i)$
    end for
    Compute overall loss: $L(\mathcal{M}) = \sum_{i=1}^{N} \gamma_i L_i(\mathcal{M}, \mathcal{B}_i, T_i)$
    Update scaling factors: $\gamma_i \leftarrow \gamma_i \exp(\alpha(1 - \text{acc}_i(\mathcal{M}, \mathcal{D}_i, T_i)))$ - α is a hyperparameter - acc_i is task-specific accuracy metric on validation set
    Update model parameters $\theta \leftarrow \mathcal{O}(\theta, \eta, \nabla L(\mathcal{M}))$
end for

Methodology

- LLM to be fine-tuned is the 4-bit quantized version of Llama 3.1.
- We'll fine tune the 8B (smallest) model, followed by the 70B (standard) model.
- The dataset used to fine-tune is the cleaned Alpaca dataset.
- We'll use the free Google Colaboratory, which has a single 16GB Nvidia T4 GPU installed. Then, we'll try to run it on a device with a single laptop 8GB Nvidia Geforce RTX 4070 installed.

ICIC 2024

UPH
UNIVERSITAS PELITA HARAPAN

TABLE I
COMPARISON OF LORA DERIVATIVES

Aspect	QLoRA	IR-QLoRA	AdaLoRA	QDyLoRA [13]	MTLoRA	ResLoRA [14] [15]	OLoRA [4]
Quantization	Yes	No	No	Yes	No	No	No
Low-Rank Adaptation	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic Rank Selection	No	No	Yes	Yes	No	No	No
Efficiency	High due to quantization and low-rank adaptation	Moderate, focuses on accurate low-rank approximations	High, dynamic rank selection optimizes resource usage	Very high, combines quantization and dynamic low-rank adaptation	High, due to shared and task-specific adaptations	High, due to residual connections improving stability	High, orthogonal matrices reduce redundancy
Model Performance	Good, preserves essential model capabilities	Very Good, maintains model integrity	Excellent, adapts to model complexity	Excellent, balances efficiency and accuracy	Excellent, leverages shared knowledge across tasks	Very Good, preserves essential features with residuals	Very Good, maintains feature diversity and stability
Application	Efficient training and fine-tuning on less powerful hardware	Fine-tuning with a focus on high accuracy	Versatile model adaptation and fine-tuning	Efficient fine-tuning and adaptation with limited resources	Multi-task learning and adaptation across various tasks	Efficient fine-tuning with residual connections for stability	Fine-tuning with orthogonal low-rank matrices for efficiency

Comparison of the LoRA Derivatives

ICIC 2024

UPH
UNIVERSITAS PELITA HARAPAN

Current Finding

TABLE II
COMPARISON OF VARIOUS FINE-TUNING SETTINGS OF LLAMA 3.1.

Fine-tuning settings	**Default**	**500 steps**	**Full fine tuning (6,470 steps)**	**70B parameters**
Memory usage	8.7 GB	8.7 GB	8.7 GB	30 GB
Training time	7.7 minutes	63 minutes	12 hours 47 minutes	Error
Trainable parameters	41,943,040	41,943,040	41,943,040	Error
Pass default test case	Yes	Yes	Yes	Error
Pass 9.9 vs 9.11 problem	No	No	No	Error

Future Research

- Try to run the fine-tuning algorithm locally and compare the performance to the cloud-based, given the same or similar specs.
- Try to use other models as the based model and other datasets as the fine-tuning data.
- Try to further optimize memory usage to use less than 8 GB (e.g. Nvidia Geforce RTX 4070 laptop GPU) of GPU memory.
- Use more test cases for testing the LLM. Especially, try to tinker the fine-tuning parameters until it passes the 9.9 vs 9.11 problem.
- Develop a metric that also implements training time, disk usage, and LLM quality (accuracy and non-hallucination)
- Implement safety guardrails for ethical AI.
- Adapt the code to use accelerated computing alternatives, such as Polars and Mojo.
- Consider other PEFT techniques (i.e. non LoRA based), such as DoRA and HiRA.

ICIC 2024

UPH
UNIVERSITAS PELITA HARAPAN

Acknowledgement

This research is partially funded by Center of Research and Community Development (LPPM), Universitas Pelita Harapan, No. 155/LPPM-UPH/VII/2024 and Faculty of Information Technology No. P-113-FIT/VII/2024 in July 2024.

Further inquiry

yugopuspito@uph.edu