LoRA

LOW-RANK ADAPTATION

FOR EFFICIENT FINE-TUNING

COMPREHENSIVE GUIDE TO
PARAMETER-EFFICIENT TRAINING

PROFESSIONAL TECHNICAL DOCUMENTATION

MADE BY: SAFIA TIFOUR

July 31, 2025

Contents

1 What is LoRA? **2**

2 Motivation: Why Use LoRA? **2**

3 Intuition Behind LoRA **2**

4 Mathematical Formulation **2**
4.1 Standard Linear Layer in Transformers 3
4.2 LoRA Reparameterization 3
4.3 Where is LoRA Applied? 3

5 Training with LoRA: Step-by-Step **4**

6 Parameter Efficiency **4**

7 Theoretical Motivation **4**
7.1 Hypothesis 4
7.2 Formal Optimization View 5

8 Empirical Justification **5**

9 Linear Algebra Justification **5**

10 Benefits of LoRA **6**

11 Limitations of LoRA **6**

1 What is LoRA?

LoRA (Low-Rank Adaptation) is a technique for fine-tuning large pre-trained models by adding small, trainable low-rank matrices to specific parts of the model, instead of updating the original model weights.

2 Motivation: Why Use LoRA?

Large models (e.g., GPT, BERT) have:

- Hundreds of millions to billions of parameters

Fine-tuning them fully:

- Requires massive compute and memory
- Is expensive and storage-heavy
- Can lead to catastrophic forgetting

LoRA was introduced to:

- **Make fine-tuning parameter-efficient**
- **Preserve the knowledge of the base model**
- **Enable task-specific tuning using minimal compute**
- **Avoid the need to store multiple full copies of the model for different tasks**

3 Intuition Behind LoRA

"Why retrain the whole model when you can steer it with a small nudge?"

The intuitive approach:

- You **freeze the big brain** (pre-trained model)
- You attach tiny trainable **"steering modules"**
- These small modules (LoRA) adapt behavior to your specific task

4 Mathematical Formulation

4.1 Standard Linear Layer in Transformers

$$y = Wx$$

Where:

- $W \in \mathbb{R}^{d \times d}$: full-rank weight matrix
- x: input vector

4.2 LoRA Reparameterization

Instead of fine-tuning W, you modify the forward pass to:

$$y = Wx + \frac{\alpha}{r} \cdot B(Ax)$$

Where:

- $A \in \mathbb{R}^{r \times d}$
- $B \in \mathbb{R}^{d \times r}$
- r is the rank (e.g., 1–64)
- α is a scaling factor (to stabilize training, typically = r)
- $\Delta W = BA$ is the low-rank update
- W is frozen
- Only A and B are trainable

So:

$$W' = W + BA$$

And during training:

- Only A and B are updated via backpropagation
- W remains unchanged

4.3 Where is LoRA Applied?

Primarily inside the Transformer attention layers, such as:

- W_q (query projection)
- W_v (value projection)
- (Sometimes also W_k, W_o, and FFN layers)

5 Training with LoRA: Step-by-Step

1. Load pre-trained model
2. Freeze all original weights
3. Inject LoRA adapters into specific weight matrices
4. Forward pass uses $W + BA$
5. Only A and B receive gradients
6. Train for the target task
7. Save only LoRA parameters (A, B)
8. Deploy by applying $W' = W + BA$ during inference

6 Parameter Efficiency

Example: BERT-base has $\sim$110M parameters.

Fine-tuning Type	Trainable Params	%
Full fine-tuning	110M	100%
LoRA (r=8)	$\sim$0.5M	0.45%

Table 1: Parameter Efficiency Comparison

LoRA can reduce trainable parameters by over **200×** while retaining performance.

7 Theoretical Motivation

7.1 Hypothesis

The weight updates needed for adaptation lie in a low-dimensional subspace.

- Empirical studies show that full fine-tuning often produces updates ΔW that are low-rank
- So instead of learning a full-rank ΔW, LoRA learns $\Delta W = BA$

This idea is supported by:

- **Linear algebra** (low-rank matrix approximation via SVD)
- **Subspace optimization**: You optimize in a constrained, low-dimensional subspace
- **Gradient projection view**: Instead of optimizing over all directions in parameter space, you restrict to a smaller, meaningful subspace

7.2 Formal Optimization View

With full fine-tuning:

$$\min_{\theta} L(f(x;\theta), y)$$

With LoRA:

$$\min_{A,B} L(f(x;\theta + BA), y)$$

Where:

- θ is the frozen original weight
- BA is the trainable low-rank update

LoRA is a reparameterization of model updates under a low-rank constraint.

8 Empirical Justification

The LoRA paper (Hu et al., 2021) shows strong performance on:

- NLP tasks (e.g., GLUE, machine translation, QA)
- Vision-language tasks (when combined with CLIP)
- Models like GPT-2, GPT-3, T5, RoBERTa, etc.

Key findings:

- LoRA matches or beats full fine-tuning in accuracy
- Training and inference are faster and cheaper
- Memory and storage usage is dramatically reduced

9 Linear Algebra Justification

Any matrix can be approximated using low-rank SVD:

$$W \approx U_r \Sigma_r V_r^T$$

LoRA mimics this using:

$$\Delta W = BA$$

With B and A randomly initialized and trained via gradient descent.

10 Benefits of LoRA

Benefit	Description
Parameter-efficient	Trains only $\sim$0.1–1% of model
Storage-efficient	Store just LoRA weights per task
No forgetting	Base model remains unchanged
Modular	Easy to switch between task adapters
Plug-and-play	Simple to integrate with existing models
Low compute cost	Much lighter training footprint

Table 2: Benefits of LoRA

11 Limitations of LoRA

- Performance depends on choice of:
  - r (rank)
  - α (scaling factor)
  - `target_modules`
- May not capture very task-specific features for low r
- Not usable for black-box models where internal weights aren't accessible