Uni-LoRA: One Vector is All You Need

论文讲解

吴雨欣

中国人民大学高瓴人工智能学院

2025 年 12 月

中国人民大學 RENMIN UNIVERSITY OF CHINA

Outline

What is Parameter-Efficient Fine-Tuning (PEFT)?

The Challenge:

- Full fine-tuning of Large Language Models is extremely expensive
- Example: Fine-tuning a 7B parameter model requires massive computational resources

What is Parameter-Efficient Fine-Tuning (PEFT)?

The Challenge:

- Full fine-tuning of Large Language Models is extremely expensive
- Example: Fine-tuning a 7B parameter model requires massive computational resources

The Solution: PEFT

- Leverage strong prior knowledge in foundation models
- Adapt to downstream tasks by updating only a small amount of parameters
- Significantly reduce fine-tuning cost while maintaining performance

LoRA: Low-Rank Adaptation

Core Idea:

- Given a pre-trained weight matrix $W_0 \in \mathbb{R}^{m \times n}$
- Constrain weight increment as low-rank decomposition:

$$\Delta W = BA$$

- Where $B \in \mathbb{R}^{m \times r}$, $A \in \mathbb{R}^{r \times n}$, and $r \ll \min(m, n)$

LoRA: Low-Rank Adaptation

Core Idea:

- Given a pre-trained weight matrix $W_0 \in \mathbb{R}^{m \times n}$
- Constrain weight increment as low-rank decomposition:

$$\Delta W = BA$$

- Where $B \in \mathbb{R}^{m \times r}$, $A \in \mathbb{R}^{r \times n}$, and $r \ll \min(m, n)$

Benefits:

- Dramatically reduces trainable parameters: from $m \times n$ to $(m + n) \times r$
- Example: $m = n = 1000, r = 4$ reduces parameters by 99%!
- Achieves impressive performance while being highly efficient

Recent LoRA Variants

Goal: Further reduce trainable parameters beyond LoRA

Representative Methods:

- **Tied-LoRA:** Ties B and A matrices across layers + diagonal scaling
- **VeRA:** Freezes randomly initialized B and A, trains only diagonal vectors
- **LoRA-XS:** Introduces $r \times r$ matrices per module
- **VB-LoRA:** Learns a global vector bank + compositional coefficients
- **FourierFT:** Uses Fourier transform for sparse representation

Recent LoRA Variants

Goal: Further reduce trainable parameters beyond LoRA

Representative Methods:

- **Tied-LoRA:** Ties B and A matrices across layers + diagonal scaling
- **VeRA:** Freezes randomly initialized B and A, trains only diagonal vectors
- **LoRA-XS:** Introduces $r \times r$ matrices per module
- **VB-LoRA:** Learns a global vector bank + compositional coefficients
- **FourierFT:** Uses Fourier transform for sparse representation

Common Challenge:

- Despite similarities, **no unified framework** exists
- Difficult to systematically analyze and compare these methods

Baseline Method 1: VeRA and Tied-LoRA

Core Idea: Add diagonal scaling to shared low-rank matrices

Architecture:

$$\Delta W = \Lambda_b \cdot P_B \cdot \Lambda_d \cdot P_A$$

- $P_B \in \mathbb{R}^{m \times r}$, $P_A \in \mathbb{R}^{r \times n}$: Shared across all layers
- $\Lambda_b \in \mathbb{R}^{m \times m}$, $\Lambda_d \in \mathbb{R}^{r \times r}$: Trainable diagonal matrices per layer

Baseline Method 1: VeRA and Tied-LoRA

Core Idea: Add diagonal scaling to shared low-rank matrices

Architecture:

$$\Delta W = \Lambda_b \cdot P_B \cdot \Lambda_d \cdot P_A$$

- $P_B \in \mathbb{R}^{m \times r}$, $P_A \in \mathbb{R}^{r \times n}$: Shared across all layers
- $\Lambda_b \in \mathbb{R}^{m \times m}$, $\Lambda_d \in \mathbb{R}^{r \times r}$: Trainable diagonal matrices per layer

Difference:

- **Tied-LoRA:** P_B and P_A are trainable
- **VeRA:** P_B and P_A are randomly initialized and frozen

Baseline Method 1: VeRA and Tied-LoRA

Core Idea: Add diagonal scaling to shared low-rank matrices

Architecture:

$$\Delta W = \Lambda_b \cdot P_B \cdot \Lambda_d \cdot P_A$$

- $P_B \in \mathbb{R}^{m\times r}$, $P_A \in \mathbb{R}^{r\times n}$: Shared across all layers
- $\Lambda_b \in \mathbb{R}^{m\times m}$, $\Lambda_d \in \mathbb{R}^{r\times r}$: Trainable diagonal matrices per layer

Difference:

- **Tied-LoRA:** P_B and P_A are trainable
- **VeRA:** P_B and P_A are randomly initialized and frozen

Trainable Parameters:

- Only diagonal elements: $L \times (m + r)$ parameters
- Much fewer than LoRA's $L \times (m + n) \times r$ parameters

Baseline Method 2: VB-LoRA

Core Idea: Learn a global vector bank for parameter sharing

How it works:

1. Decompose B and A matrices into fixed-length sub-vectors
2. Create a global vector bank: $\mathcal{B} = \{\alpha_1, \alpha_2, \ldots, \alpha_h\}$
3. Each sub-vector = weighted combination of top-K vectors from $\mathcal{B}$

Baseline Method 2: VB-LoRA

Core Idea: Learn a global vector bank for parameter sharing

How it works:

1. Decompose B and A matrices into fixed-length sub-vectors
2. Create a global vector bank: $\mathcal{B} = \{\alpha_1, \alpha_2, \ldots, \alpha_h\}$
3. Each sub-vector $=$ weighted combination of top-K vectors from $\mathcal{B}$

Example:

- Vector bank size: $h = 2048$, each vector length: $b = 256$
- Each sub-vector uses $K = 2$ vectors from the bank
- Store: bank vectors + top-K indices + top-K coefficients

Baseline Method 2: VB-LoRA

Core Idea: Learn a global vector bank for parameter sharing

How it works:

1. Decompose B and A matrices into fixed-length sub-vectors
2. Create a global vector bank: $\mathcal{B} = \{\alpha_1, \alpha_2, \ldots, \alpha_h\}$
3. Each sub-vector = weighted combination of top-K vectors from $\mathcal{B}$

Example:

- Vector bank size: $h = 2048$, each vector length: $b = 256$
- Each sub-vector uses $K = 2$ vectors from the bank
- Store: bank vectors + top-K indices + top-K coefficients

Advantage:

- Learns *what* to share (adaptive parameter sharing)
- Extremely low storage after training (sparse coefficients)

Baseline Method 3: LoRA-XS and FourierFT

LoRA-XS:

- Uses SVD of pretrained weights: $W_0 = P_B \Sigma P_A$
- Freezes P_B and P_A, trains only $\Lambda_R \in \mathbb{R}^{r \times r}$
- Weight update: $\Delta W = P_B \Lambda_R P_A$
- Trainable parameters: $L \times r^2$ (very small when r is small)

Baseline Method 3: LoRA-XS and FourierFT

LoRA-XS:

- Uses SVD of pretrained weights: $W_0 = P_B \Sigma P_A$
- Freezes P_B and P_A, trains only $\Lambda_R \in \mathbb{R}^{r \times r}$
- Weight update: $\Delta W = P_B \Lambda_R P_A$
- Trainable parameters: $L \times r^2$ (very small when r is small)

FourierFT:

- Uses Fast Fourier Transform (FFT) for sparse representation
- Represents ΔW with a small number of Fourier coefficients
- Only trains the selected Fourier coefficients
- Leverages frequency domain sparsity

Baseline Method 3: LoRA-XS and FourierFT

LoRA-XS:

- Uses SVD of pretrained weights: $W_0 = P_B \Sigma P_A$
- Freezes P_B and P_A, trains only $\Lambda_R \in \mathbb{R}^{r \times r}$
- Weight update: $\Delta W = P_B \Lambda_R P_A$
- Trainable parameters: $L \times r^2$ (very small when r is small)

FourierFT:

- Uses Fast Fourier Transform (FFT) for sparse representation
- Represents ΔW with a small number of Fourier coefficients
- Only trains the selected Fourier coefficients
- Leverages frequency domain sparsity

Common Theme:

- Both use *structured* low-dimensional representations
- LoRA-XS: structure from SVD; FourierFT: structure from FFT

Motivation: Key Limitations

Three Major Problems in Existing LoRA Variants:

1. **Lack of Unified View**
   - Each method has its own formulation
   - Hard to understand fundamental differences

Motivation: Key Limitations

Three Major Problems in Existing LoRA Variants:

1. **Lack of Unified View**
   - Each method has its own formulation
   - Hard to understand fundamental differences
2. **Suboptimal Projection Design**
   - *Layer-wise (Local):* Limited cross-layer parameter sharing
   - *Non-uniform:* Unbalanced dimension allocation
   - *Non-isometric:* Distorts optimization landscape geometry

Motivation: Key Limitations

Three Major Problems in Existing LoRA Variants:

1. **Lack of Unified View**
   - Each method has its own formulation
   - Hard to understand fundamental differences
2. **Suboptimal Projection Design**
   - *Layer-wise (Local):* Limited cross-layer parameter sharing
   - *Non-uniform:* Unbalanced dimension allocation
   - *Non-isometric:* Distorts optimization landscape geometry
3. **High Computational Complexity**
   - Some methods (e.g., Fastfood) have $O(D \log d)$ time complexity

The Core Insight: Unified Formulation

Key Observation:

- All LoRA variants can be viewed as parameter space reduction
- They project full LoRA parameters into a lower-dimensional subspace

The Core Insight: Unified Formulation

Key Observation:

- All LoRA variants can be viewed as parameter space reduction
- They project full LoRA parameters into a lower-dimensional subspace

Unified Framework:

$$\theta_D = P\theta_d$$

Where:

- $\theta_D \in \mathbb{R}^D$: Full LoRA parameter space ($D = L(m+n)r$)
- $\theta_d \in \mathbb{R}^d$: Low-dimensional trainable parameters ($d \ll D$)
- $P \in \mathbb{R}^{D\times d}$: Projection matrix

The Core Insight: Unified Formulation

Key Observation:

- All LoRA variants can be viewed as parameter space reduction
- They project full LoRA parameters into a lower-dimensional subspace

Unified Framework:

$$\theta_D = P\theta_d$$

Where:

- $\theta_D \in \mathbb{R}^D$: Full LoRA parameter space ($D = L(m+n)r$)
- $\theta_d \in \mathbb{R}^d$: Low-dimensional trainable parameters ($d \ll D$)
- $P \in \mathbb{R}^{D \times d}$: Projection matrix

The Key Difference: All methods differ only in the choice of P!

Constructing θ_D: Full LoRA Parameter Vector

How to construct θ_D?

For L LoRA-adapted modules, each with $B_\ell \in \mathbb{R}^{m\times r}$ and $A_\ell \in \mathbb{R}^{r\times n}$:

$$\theta_D = \text{Concat}\left[\text{vec}_{\text{row}}(B_1), \text{vec}_{\text{row}}(A_1), \ldots, \text{vec}_{\text{row}}(B_L), \text{vec}_{\text{row}}(A_L)\right]$$

Where:

- $\text{vec}_{\text{row}}(\cdot)$: Row-wise flattening of a matrix into a vector
- Total dimension: $D = L(m+n)r$

Constructing θ_D: Full LoRA Parameter Vector

How to construct θ_D?

For L LoRA-adapted modules, each with $B_\ell \in \mathbb{R}^{m\times r}$ and $A_\ell \in \mathbb{R}^{r\times n}$:

$$\theta_D = \text{Concat}\left[\text{vec}_{\text{row}}(B_1), \text{vec}_{\text{row}}(A_1), \dots, \text{vec}_{\text{row}}(B_L), \text{vec}_{\text{row}}(A_L)\right]$$

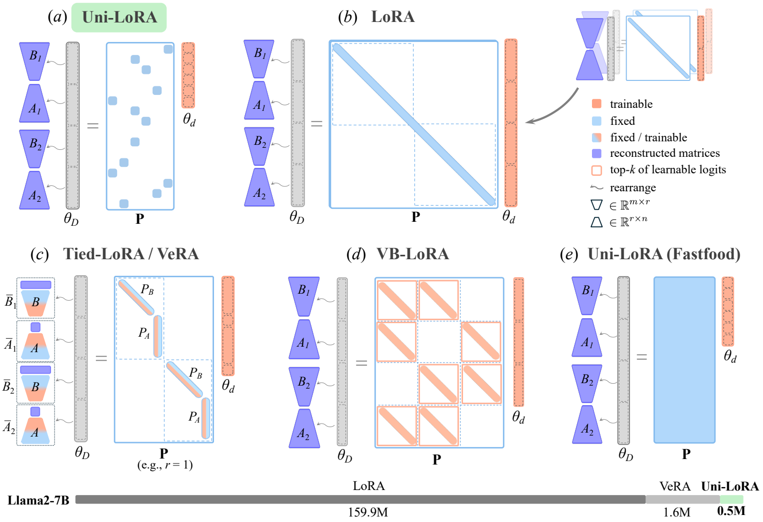
Where:

- $\text{vec}_{\text{row}}(\cdot)$: Row-wise flattening of a matrix into a vector
- Total dimension: $D = L(m+n)r$

Intuition:

- Stack all LoRA matrices into one long vector
- This is the “full parameter space” we want to compress

Unified View: Representing Different Methods



Unified View: Representing Different Methods

Key Insight: The projection matrix P determines the method!

- **LoRA:** $P = I_{D \times D}$ (identity matrix, no compression)
- **VeRA/Tied-LoRA:** Block-diagonal structure (local projection)
- **VB-LoRA:** Learned sparse projection
- **Uni-LoRA:** Random uniform global projection (our method!)

Uni-LoRA's Projection Matrix: The Innovation

Construction (Extremely Simple!):

Step 1: Random Grouping

- Each row of P is a one-hot vector
- The position of "1" is uniformly sampled from $\{1, 2, \ldots, d\}$

Uni-LoRA's Projection Matrix: The Innovation

Construction (Extremely Simple!):

Step 1: Random Grouping

- Each row of P is a one-hot vector
- The position of "1" is uniformly sampled from $\{1, 2, \ldots, d\}$

Step 2: Column Normalization

- Count n_j = number of 1's in column j
- Set all entries in column j to $1/\sqrt{n_j}$

Uni-LoRA's Projection Matrix: The Innovation

Construction (Extremely Simple!):

Step 1: Random Grouping

- Each row of P is a one-hot vector
- The position of "1" is uniformly sampled from $\{1, 2, \ldots, d\}$

Step 2: Column Normalization

- Count $n_j =$ number of 1's in column j
- Set all entries in column j to $1/\sqrt{n_j}$

Intuitive Interpretation:

- Randomly partition D parameters into d groups ("buckets")
- Parameters in the same group share the same value
- Normalization ensures balanced weighting

Uni-LoRA's Projection Matrix: The Innovation

Construction (Extremely Simple!):

Step 1: Random Grouping

- Each row of P is a one-hot vector
- The position of "1" is uniformly sampled from $\{1, 2, \ldots, d\}$

Step 2: Column Normalization

- Count n_j = number of 1's in column j
- Set all entries in column j to $1/\sqrt{n_j}$

Intuitive Interpretation:

- Randomly partition D parameters into d groups ("buckets")
- Parameters in the same group share the same value
- Normalization ensures balanced weighting

Important: P is fixed after initialization; only θ_d is trained!

Example: How the Projection Works

Toy Example: $D = 10$ parameters, $d = 3$ subspace dimensions

Before normalization:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Column counts: $n_1 = 4, n_2 = 3, n_3 = 3$

Example: How the Projection Works

Toy Example: $D = 10$ parameters, $d = 3$ subspace dimensions

Before normalization:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Column counts: $n_1 = 4, n_2 = 3, n_3 = 3$

After normalization: Replace all 1's in column j with $1/\sqrt{n_j}$

If $\theta_d = [a, b, c]^T$, then parameters $\{1, 3, 5, 8\}$ all equal $a/2$, etc.

Example: How the Projection Works

Toy Example: $D = 10$ parameters, $d = 3$ subspace dimensions

After normalization:

$$P = \begin{bmatrix} a/\sqrt{4} & 0 & 0 \\ 0 & b/\sqrt{3} & 0 \\ a/\sqrt{4} & 0 & 0 \\ 0 & 0 & c/\sqrt{3} \\ a/\sqrt{4} & 0 & 0 \\ 0 & b/\sqrt{3} & 0 \\ 0 & 0 & c/\sqrt{3} \\ a/\sqrt{4} & 0 & 0 \\ 0 & b/\sqrt{3} & 0 \\ 0 & 0 & c/\sqrt{3} \end{bmatrix}$$

Property 1: Globality

Definition:

- Global parameter sharing across **all layers** and **all matrix types**
- Breaks the physical barrier between layers

Property 1: Globality

Definition:

- Global parameter sharing across **all layers** and **all matrix types**
- Breaks the physical barrier between layers

Comparison:

- VeRA/Tied-LoRA: Layer-wise projection (local)
- LoRA-XS: Separate projections for each module
- Uni-LoRA: All parameters project to the same d-dimensional space

Property 1: Globality

Definition:

- Global parameter sharing across **all layers** and **all matrix types**
- Breaks the physical barrier between layers

Comparison:

- VeRA/Tied-LoRA: Layer-wise projection (local)
- LoRA-XS: Separate projections for each module
- Uni-LoRA: All parameters project to the same d-dimensional space

Benefit:

- Maximizes parameter redundancy reduction
- Enables cross-layer knowledge sharing
- Achieves extreme parameter efficiency

Property 2: Uniformity / Load-Balanced

Definition:

- Each dimension in θ_d corresponds to **roughly equal** number of original parameters
- Information is **evenly distributed** across all subspace dimensions

Property 2: Uniformity / Load-Balanced

Definition:

- Each dimension in θ_d corresponds to **roughly equal** number of original parameters
- Information is **evenly distributed** across all subspace dimensions

Comparison:

- VeRA/Tied-LoRA: Non-uniform
  - B matrices: mapped to m dimensions
  - A matrices: mapped to r dimensions
  - Imbalanced even though B and A have same parameter count!
- Uni-LoRA: Each dimension handles $\approx D/d$ parameters

Property 2: Uniformity / Load-Balanced

Definition:

- Each dimension in θ_d corresponds to **roughly equal** number of original parameters
- Information is **evenly distributed** across all subspace dimensions

Comparison:

- VeRA/Tied-LoRA: Non-uniform
  - B matrices: mapped to m dimensions
  - A matrices: mapped to r dimensions
  - Imbalanced even though B and A have same parameter count!
- Uni-LoRA: Each dimension handles $\approx D/d$ parameters

Benefit:

- No “wasted” dimensions
- Better utilization of limited subspace capacity

Property 3: Isometry (Distance-Preserving)

Definition:

- The projection preserves distances between parameter vectors
- Mathematically: $\|P(x-y)\| = \|x-y\|$ for all $x, y \in \mathbb{R}^d$

Property 3: Isometry (Distance-Preserving)

Definition:

- The projection preserves distances between parameter vectors
- Mathematically: $\|P(x-y)\| = \|x-y\|$ for all $x, y \in \mathbb{R}^d$

Why is this important?

- Preserves the **geometry** of the optimization landscape
- Optimization in subspace behaves like optimization in full space
- No distortion of the loss surface

Property 3: Isometry (Distance-Preserving)

Definition:

- The projection preserves distances between parameter vectors
- Mathematically: $\|P(x-y)\| = \|x-y\|$ for all $x, y \in \mathbb{R}^d$

Why is this important?

- Preserves the **geometry** of the optimization landscape
- Optimization in subspace behaves like optimization in full space
- No distortion of the loss surface

Comparison:

- VeRA/Tied-LoRA/VB-LoRA: Not isometric
- Uni-LoRA & Fastfood: Isometric (distance-preserving)

Theorem 1: Uni-LoRA's Projection is Isometric

Theorem 1: Let $P \in \mathbb{R}^{D \times d}$ be constructed as described (random one-hot + normalization). Then P is isometric:

$$\|P(x-y)\| = \|x-y\|, \quad \forall x, y \in \mathbb{R}^d$$

Theorem 1: Uni-LoRA's Projection is Isometric

Theorem 1: Let $P \in \mathbb{R}^{D \times d}$ be constructed as described (random one-hot + normalization). Then P is isometric:

$$\|P(x-y)\| = \|x-y\|, \quad \forall x, y \in \mathbb{R}^d$$

Proof Sketch:

- Show that $P^T P = I_d$ (identity matrix)
- For $j \neq k$: No row has 1's in both columns j and k $\Rightarrow [P^T P]_{j,k} = 0$
- For $j = k$: Column j has n_j entries of $1/\sqrt{n_j}$ $\Rightarrow [P^T P]_{j,j} = n_j \cdot (1/\sqrt{n_j})^2 = 1$
- Therefore: $\|P(x-y)\|^2 = (x-y)^T P^T P(x-y) = \|x-y\|^2$

Theorem 1: Uni-LoRA's Projection is Isometric

Theorem 1: Let $P \in \mathbb{R}^{D \times d}$ be constructed as described (random one-hot + normalization). Then P is isometric:

$$\|P(x-y)\| = \|x-y\|, \quad \forall x, y \in \mathbb{R}^d$$

Proof Sketch:

- Show that $P^T P = I_d$ (identity matrix)
- For $j \neq k$: No row has 1's in both columns j and k $\Rightarrow [P^T P]_{j,k} = 0$
- For $j = k$: Column j has n_j entries of $1/\sqrt{n_j}$ $\Rightarrow [P^T P]_{j,j} = n_j \cdot (1/\sqrt{n_j})^2 = 1$
- Therefore: $\|P(x-y)\|^2 = (x-y)^T P^T P(x-y) = \|x-y\|^2$

Consequence: Optimization landscape geometry is perfectly preserved!

Summary: Three Properties Comparison

Method	Globality	Uniformity	Isometry
VeRA	×	×	×
Tied-LoRA	×	×	×
VB-LoRA	✓	✓	×
LoRA-XS	×	✓	✓
Fastfood	✓	✓	✓
Uni-LoRA	✓	✓	✓

Uni-LoRA is the only method satisfying all three properties while being simpler and faster than Fastfood!

Storage Complexity: One Vector is All You Need!

What needs to be stored after training?

- $\theta_d \in \mathbb{R}^d$: The learned parameter vector
- Random seed: To regenerate the projection matrix P

Storage Complexity: One Vector is All You Need!

What needs to be stored after training?

- $\theta_d \in \mathbb{R}^d$: The learned parameter vector
- Random seed: To regenerate the projection matrix P

Why don't we need to store P?

- P is constructed **deterministically** from a random seed
- Given the same seed, we can regenerate **exactly** the same P
- Process: seed $\rightarrow$ random indices $\rightarrow$ compute n_j $\rightarrow$ normalize
- Storage: seed (1 number) vs full P ($D \times d$ numbers!)

Storage Complexity: One Vector is All You Need!

What needs to be stored after training?

- $\theta_d \in \mathbb{R}^d$: The learned parameter vector
- Random seed: To regenerate the projection matrix P

Why don't we need to store P?

- P is constructed **deterministically** from a random seed
- Given the same seed, we can regenerate **exactly** the same P
- Process: seed $\to$ random indices $\to$ compute n_j $\to$ normalize
- Storage: seed (1 number) vs full P ($D \times d$ numbers!)

Total storage: $d + 1$ numbers!

Storage Complexity: One Vector is All You Need!

What needs to be stored after training?

- $\theta_d \in \mathbb{R}^d$: The learned parameter vector
- Random seed: To regenerate the projection matrix P

Why don't we need to store P?

- P is constructed **deterministically** from a random seed
- Given the same seed, we can regenerate **exactly** the same P
- Process: seed $\rightarrow$ random indices $\rightarrow$ compute n_j $\rightarrow$ normalize
- Storage: seed (1 number) vs full P ($D \times d$ numbers!)

Total storage: $d + 1$ numbers!

Example (Llama2-7B):

- LoRA (rank 64): 159.9M parameters
- Uni-LoRA: 0.52M parameters + 1 seed
- Reduction: **99.7%** fewer parameters!

Storage Complexity: One Vector is All You Need!

What needs to be stored after training?

- $\theta_d \in \mathbb{R}^d$: The learned parameter vector
- Random seed: To regenerate the projection matrix P

Why don't we need to store P?

- P is constructed **deterministically** from a random seed
- Given the same seed, we can regenerate **exactly** the same P
- Process: seed $\rightarrow$ random indices $\rightarrow$ compute n_j $\rightarrow$ normalize
- Storage: seed (1 number) vs full P ($D \times d$ numbers!)

Total storage: $d + 1$ numbers!

Example (Llama2-7B):

- LoRA (rank 64): 159.9M parameters
- Uni-LoRA: 0.52M parameters + 1 seed
- Reduction: **99.7%** fewer parameters!

Time Complexity: Faster than Fastfood

Projection Operation: $P\theta_d$

Method	Time	Space
Dense Gaussian	$O(Dd)$	$O(Dd)$
Fastfood	$O(D\log d)$	$O(D)$
Uni-LoRA	$O(D)$	$O(D)$

Time Complexity: Faster than Fastfood

Projection Operation: $P\theta_d$

Method	Time	Space
Dense Gaussian	$O(Dd)$	$O(Dd)$
Fastfood	$O(D\log d)$	$O(D)$
Uni-LoRA	$O(D)$	$O(D)$

Why is Uni-LoRA $O(D)$?

- P is sparse with exactly D non-zero entries
- We don't store P explicitly, only indices and values
- Projection is just: $\theta_D[i] = \theta_d[\text{index}[i]] \times \text{norm}[i]$

Time Complexity: Faster than Fastfood

Projection Operation: $P\theta_d$

Method	Time	Space
Dense Gaussian	$O(Dd)$	$O(Dd)$
Fastfood	$O(D\log d)$	$O(D)$
Uni-LoRA	$O(D)$	$O(D)$

Why is Uni-LoRA $O(D)$?

- P is sparse with exactly D non-zero entries
- We don't store P explicitly, only indices and values
- Projection is just: $\theta_D[i] = \theta_d[\text{index}[i]] \times \text{norm}[i]$

Advantage over Fastfood:

- Simpler implementation
- Lower time complexity: $O(D)$ vs $O(D\log d)$

Experimental Setup

Tasks:

1. Natural Language Understanding (GLUE benchmark)
2. Mathematical Reasoning (GSM8K, MATH)
3. Instruction Tuning (MT-Bench)
4. Computer Vision (8 datasets with ViT)

Experimental Setup

Tasks:

1. Natural Language Understanding (GLUE benchmark)
2. Mathematical Reasoning (GSM8K, MATH)
3. Instruction Tuning (MT-Bench)
4. Computer Vision (8 datasets with ViT)

Base Models:

- RoBERTa-base/large, Mistral, Gemma, Llama2
- ViT-Base, ViT-Large

Experimental Setup

Tasks:

1. Natural Language Understanding (GLUE benchmark)
2. Mathematical Reasoning (GSM8K, MATH)
3. Instruction Tuning (MT-Bench)
4. Computer Vision (8 datasets with ViT)

Base Models:

- RoBERTa-base/large, Mistral, Gemma, Llama2
- ViT-Base, ViT-Large

Baselines:

- LoRA, VeRA, Tied-LoRA, VB-LoRA, LoRA-XS, FourierFT

Experimental Setup

Tasks:

1. Natural Language Understanding (GLUE benchmark)
2. Mathematical Reasoning (GSM8K, MATH)
3. Instruction Tuning (MT-Bench)
4. Computer Vision (8 datasets with ViT)

Base Models:

- RoBERTa-base/large, Mistral, Gemma, Llama2
- ViT-Base, ViT-Large

Baselines:

- LoRA, VeRA, Tied-LoRA, VB-LoRA, LoRA-XS, FourierFT

Configuration:

- Rank $r = 4$ for all methods
- Subspace dimension d matched to best baseline

Results: GLUE Benchmark (RoBERTa-large)

Method	# Params	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg
LoRA	0.786M	96.2	90.2	68.2	94.8	85.2	92.3	87.8
VeRA	0.061M	96.1	90.9	68.0	94.4	85.9	91.7	87.8
Tied-LoRA	0.066M	94.8	89.7	64.7	94.1	81.2	90.8	85.9
VB-LoRA	0.162M	96.1	91.4	68.3	94.7	86.6	91.8	88.2
LoRA-XS	0.025M	95.9	90.7	67.0	93.9	88.1	92.0	87.9
FourierFT	0.048M	96.0	90.9	67.1	94.4	87.4	92.0	88.0
Uni-LoRA	**0.023M**	**96.3**	**91.3**	**68.5**	**94.6**	86.6	**92.1**	**88.3**

Key Observations:

- **Best performance:** Ranks 1st or 2nd in 11 out of 12 tasks
- **Fewest parameters:** 0.023M (60% fewer than Tied-LoRA/VeRA)
- **Matches VB-LoRA:** Similar accuracy with simpler design

Results: Mathematical Reasoning

Model	Method	# Params	GSM8K	MATH
Mistral-7B	Full-FT	7242M	67.02	18.60
	LoRA	168M	67.70	19.68
	LoRA-XS	0.92M	68.01	17.86
	VB-LoRA	93M	69.22	17.90
	VeRA	1.39M	68.69	18.81
	Uni-LoRA	**0.52M**	**68.54**	**18.18**
Gemma-7B	Full-FT	8538M	71.34	22.74
	LoRA	200M	74.90	31.28
	LoRA-XS	0.80M	74.22	27.62
	VB-LoRA	113M	74.86	28.90
	VeRA	1.90M	74.98	28.84
	Uni-LoRA	**0.52M**	**75.59**	**28.94**

Remarkable: Uni-LoRA achieves or exceeds LoRA performance with **300Œ fewer parameters**!

Results: Instruction Tuning (MT-Bench)

Model	Method	# Params	Score1	Score2
Llama2-7B	LoRA	159.9M	5.62	3.23
	VB-LoRA	83M	5.43	3.46
	Uni-LoRA	**0.52M**	**5.58**	**3.56**
Llama2-13B	LoRA	250.3M	6.20	4.13
	VB-LoRA	256M	5.96	4.33
	Uni-LoRA	**1.0M**	**6.34**	**4.43**

Score1/Score2: GPT-4 evaluation on single-turn / multi-turn dialogues

Highlights:

- Uses only 0.3% of LoRA's parameters
- **Outperforms** both LoRA and VB-LoRA
- Demonstrates strong generalization to instruction-following

Results: Computer Vision (ViT)

Model	Method	# Params	Avg Acc	vs Full-FT
ViT-Base	Full-FT	85.8M	86.49	-
	FourierFT	72K	77.75	-8.74
	Uni-LoRA	**72K**	**85.15**	**-1.34**
ViT-Large	Full-FT	303.3M	90.20	-
	FourierFT	144K	83.71	-6.49
	Uni-LoRA	**144K**	**88.00**	**-2.20**

Key Findings:

- **Significantly outperforms FourierFT** (+7.4% on ViT-Base)
- With $<$ **0.1%** of full fine-tuning parameters, achieves **97.5%** of its performance
- Shows Uni-LoRA generalizes beyond NLP to vision tasks

Ablation 1: Uni-LoRA vs Fastfood

Question: Can our simple projection replace the classical Fastfood?

Task	Uni-LoRA	Fastfood	Time Speedup
MRPC	91.3	90.7	2.9Œ (9 vs 26 min)
CoLA	68.5	65.3	2.9Œ (21 vs 60 min)
SST-2	96.3	96.1	3.1Œ (80 vs 251 min)
QNLI	94.6	94.1	2.4Œ (147 vs 358 min)

Conclusion:

- **Better performance** across all tasks
- **2-3Œ faster** training time
- Validates our $O(D)$ vs $O(D \log d)$ analysis

Ablation 2: Global vs Local Projection

Question: Is global projection really better than local projection?

Setup:

- **Global:** All layers share the same d-dimensional subspace (Uni-LoRA)
- **Local:** Each layer has its own subspace, total dimension $= d$

Ablation 2: Global vs Local Projection

Question: Is global projection really better than local projection?

Setup:

- **Global:** All layers share the same d-dimensional subspace (Uni-LoRA)
- **Local:** Each layer has its own subspace, total dimension $= d$

Task	Global	Local	Improvement
MRPC	91.3	90.9	+0.4
CoLA	68.5	68.5	0.0
SST-2	96.3	96.2	+0.1
QNLI	94.6	94.5	+0.1

Conclusion: Global projection consistently matches or outperforms local projection

Ablation 3: Uniform vs Non-uniform Projection

Question: Does uniformity matter?

Setup:

- **Uniform:** All parameters treated equally (Uni-LoRA)
- **Non-uniform:** B matrices use $2/3$ of dimensions, A matrices use $1/3$

Ablation 3: Uniform vs Non-uniform Projection

Question: Does uniformity matter?

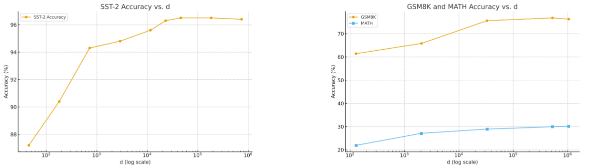
Setup:

- **Uniform:** All parameters treated equally (Uni-LoRA)
- **Non-uniform:** B matrices use $2/3$ of dimensions, A matrices use $1/3$

Task	Uniform	Non-uniform	Gap
MRPC	91.3	90.7	+0.6
CoLA	68.5	67.0	+1.5
SST-2	96.3	96.1	+0.2
QNLI	94.6	94.0	+0.6

Conclusion: Uniform projection is **consistently better**, especially on CoLA

Impact of Subspace Dimension d



(a) Accuracy on SST-2 as d increases. (b) Accuracies on GSM8K and MATH as d increases.

图 2: Performance vs subspace dimension d (log scale)

Observations:

- Performance improves rapidly when d is small
- Plateaus as d increases
- Suggests an “intrinsic dimension” for fine-tuning

Contributions Summary

Key Contributions:

1. **Unified Framework**
   - Provides a common language to analyze LoRA variants
   - Shows all methods differ only in projection matrix P
2. **Simple yet Effective Projection**
   - Random uniform partition + normalization
   - Enjoys all three key properties: global, uniform, isometric
   - $O(D)$ time complexity, simpler than Fastfood
3. **State-of-the-Art Performance**
   - Matches or outperforms existing methods
   - Achieves extreme parameter efficiency (0.5M for 7B models)
   - Generalizes across NLP and CV tasks

Limitations and Future Directions

Current Limitations:

1. **Limited Scale of Evaluation**
   - Mainly tested on small-to-medium benchmarks
   - Need validation on larger-scale tasks
2. **Fixed Subspace Dimension**
   - d is currently fixed
   - Could benefit from adaptive d based on task complexity

Limitations and Future Directions

Current Limitations:

1. **Limited Scale of Evaluation**
   - Mainly tested on small-to-medium benchmarks
   - Need validation on larger-scale tasks
2. **Fixed Subspace Dimension**
   - d is currently fixed
   - Could benefit from adaptive d based on task complexity

Future Research Directions:

1. **Adaptive Dimensionality**
   - Dynamically adjust d during training
   - Task-specific intrinsic dimension estimation
2. **Hierarchical Projection**
   - Balance between global and local projection
   - Layer-specific subspace refinement
3. **Theoretical Analysis**
   - Can we predict optimal d theoretically?

Thanks!

Questions?

Paper: *Uni-LoRA: One Vector is All You Need*
Code: `https://github.com/KaiyangLi1992/Uni-LoRA`