Learn More, But Bother Less

Parameter-Efficient Continual Learning for LLMs

Fuli Qiao   Mehrdad Mahdavi

Pennsylvania State University

NeurIPS 2024

Introduction: The Continual Learning Challenge

- Large Language Models (LLMs) need to learn multiple tasks sequentially
- Full fine-tuning is computationally expensive for many tasks
- **Two major problems in continual learning:**
  1. Catastrophic Forgetting: Performance on old tasks drops dramatically when learning new tasks
  2. Forward Transfer: New tasks should benefit from knowledge learned in previous tasks
- Most existing methods only focus on preventing forgetting
- **Goal**: Balance both forgetting prevention AND knowledge transfer

Background: Low-Rank Adaptation

- **LoRA (Low-Rank Adaptation)**: Instead of updating all parameters, use small adapter matrices
- LoRA: $W_0 + BA$ where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times d}$, and $r \ll d$
- **SVD-based approach** (this paper): $W_0 + U\Sigma V$
  - U and V are orthogonal matrices (left/right singular vectors)
  - Σ is diagonal matrix of singular values
  - Each "triplet" (u_i, σ_i, v_i) represents an independent direction
- **Why SVD over LoRA?**
  - Orthogonal structure enables clean subspace separation
  - Independent triplets make it easier to measure importance
  - Better suited for identifying task-specific knowledge

Continual Learning Setup

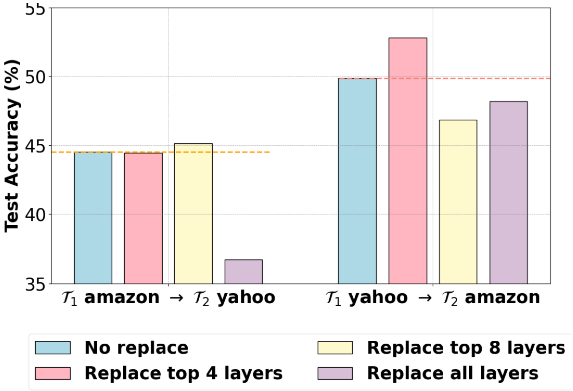
- Sequence of tasks: $T_1, T_2, \ldots, T_T$
- Pre-trained model: W_0 (frozen)
- For each task T_k, add low-rank adapter: $U^k \Sigma^k V^k$
- **Model after task** T:

$$\theta^T = W_0 + U^1 \Sigma^1 V^1 + U^2 \Sigma^2 V^2 + \cdots + U^T \Sigma^T V^T$$

- **Key idea**: Adapters accumulate and stay in the model
- Each task has its own adapter at every layer of the network
- Testing uses all adapters together (not task-specific)

Motivation: Simple Layer Replacement Helps

- **Experiment**: Train T1 (Amazon), then train T2 (Yahoo)
- At T2 initialization, try copying layers from T1
- **Strategies tested**:
  - No replacement (random init)
  - Replace top 4 layers
  - Replace top 9 layers
  - Replace all layers
- Test both tasks after T2 training



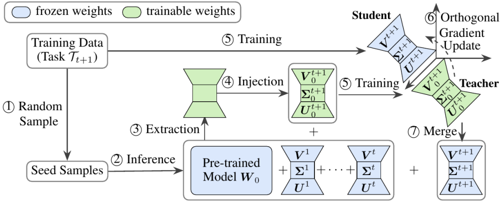
Motivation: Key Findings

- **Best strategy (top 4 or top 9)**: Improves BOTH tasks!
  - Amazon $\rightarrow$ Yahoo: T1 improves from 15.4% to 16.6%, T2 from 73.3% to 73.9%
  - Yahoo $\rightarrow$ Amazon: T1 improves from 46.8% to 50.7%, T2 from 52.9% to 54.9%
- **Copying all layers hurts**: T1 drops to 0.2% (catastrophic!)
- **Key insights**:
  1. Transferring knowledge from old tasks helps new tasks learn better
  2. Selective transfer is crucial (not all parameters are equally important)
  3. Need a smart strategy to identify which knowledge to transfer
- **This motivates LB-CL**: Use sensitivity scores to find important parameters

LB-CL Method Overview

Two-Stage Approach:

1. Knowledge Extraction & Injection
   - Calculate sensitivity scores for old task parameters
   - Use weighted combination to initialize new task
2. Orthogonal Training
   - Train new task in orthogonal subspace
   - Prevents interference with old tasks



Philosophy: Learn more (good initialization) but bother less (orthogonal training)

Stage 1: Knowledge Extraction via Sensitivity

- **Goal**: Identify which parameters from old tasks are most important
- **Sensitivity score**: Measures impact on loss when parameter is removed
- For each triplet $(U_{l,i}^k, \sigma_{l,i}^k, V_{l,i}^k)$ at layer l:

$$S_{l,i}^k = S(\sigma_{l,i}^k) + \frac{1}{d_1} \sum_j S(U_{l,ji}^k) + \frac{1}{d_2} \sum_j S(V_{l,ij}^k)$$

- Use small set of seed samples from new task (8 samples work well)
- Higher sensitivity $=$ more important for the task
- **Intuition**: If removing a parameter hurts performance, it's important!

Stage 1: Knowledge Injection

- **Goal**: Initialize new task using weighted combination of old tasks
- Calculate weights based on sensitivity scores:

$$\alpha_{l,i}^k = \frac{S_{l,i}^k}{\sum_{k=1}^{t-1} S_{l,i}^k}$$

- Initialize new task triplet as weighted sum:

$$G_{l,i}^t = \left\{ \sum_{k=1}^{t-1} \alpha_{l,i}^k U_{l,*i}^k, \sum_{k=1}^{t-1} \alpha_{l,i}^k \sigma_{l,i}^k, \sum_{k=1}^{t-1} \alpha_{l,i}^k V_{l,i*}^k \right\}$$

- **Effect**: New task starts from a better position than random initialization
- More important old knowledge gets higher weight in initialization

Stage 2: Training in Orthogonal Subspaces

- **Problem**: Even with good initialization, training can interfere with old tasks
- **Solution**: Force new task to learn in orthogonal direction to old tasks
- **Gradient Projection**:
  - Compute gradient for new task: ∇G_l^t
  - Project onto old task subspaces: $\text{proj}(\nabla G_l^t, G_l^k)$ for $k < t$
  - Subtract projections to get orthogonal component
- Modified gradient update:

$$\nabla G_l^t \leftarrow \nabla G_l^t - \sum_{k=1}^{t-1} \text{proj}(\nabla G_l^t, G_l^k)$$

- **Intuition**: Move in a direction perpendicular to old tasks' directions
- Old task adapters remain frozen during new task training

Why Both Components Are Necessary

- **Decomposition of generalization error**:
  1. Forgetting error (how much we forget old tasks)
  2. Fine-tuning performance (how well we learn new task)
  3. Initial model quality (how good is our starting point)
- **Orthogonal training alone (B-CL)**:
  - Prevents forgetting effectively
  - But limited by poor random initialization
  - Low-rank updates cannot match full fine-tuning accuracy
- **Smart initialization alone (L-CL)**:
  - Provides good starting point
  - But training still causes interference
- **Both together (LB-CL)**: Best of both worlds!

Experimental Setup

- **Models**: T5-base and T5-large (encoder-decoder architecture)
- **Standard CL Benchmark** (5 tasks):
  - AG News, Amazon Reviews, Yelp Reviews, DBpedia, Yahoo Answers
  - Text classification tasks
  - 3 different task orders tested
- **Large Benchmark** (15 tasks):
  - Standard 5 + GLUE tasks + SuperGLUE tasks + IMDB
  - More challenging long-sequence scenario
- **Evaluation Metric**: Average Accuracy across all tasks after final training
- **Baselines**: O-LoRA (previous SOTA), ProgPrompt, LFPT5, and others

Main Results: Performance Comparison

Method	Standard CL (5 tasks)				Large Benchmark (15 tasks)			
	Ord-1	Ord-2	Ord-3	avg	Ord-4	Ord-5	Ord-6	avg
SeqFT	18.9	24.9	41.7	28.5	7.4	7.3	7.4	7.4
IncLoRA	63.4	62.2	65.1	63.6	63.0	57.9	60.4	60.5
LFPT5	66.6	71.2	76.2	71.3	69.8	67.2	69.2	68.7
O-LoRA	74.9	75.3	75.9	**75.4**	70.5	65.5	70.5	**68.8**
LB-CL	**76.9**	**76.5**	**76.8**	**76.7**	**68.4**	**67.3**	**71.8**	**69.2**
ProgPrompt	76.1	76.0	76.3	76.1	78.7	78.8	77.8	78.4
MTL (upper)	80.0	80.0	80.0	80.0	76.3	76.3	76.3	76.3

- LB-CL outperforms O-LoRA (previous SOTA) on both benchmarks
- Consistent improvement across all task orders
- Performance close to ProgPrompt (requires task IDs) and MTL (upper bound)

Ablation Study: What Makes It Work?

Variants tested:

- **L-CL**: Initialization only
- **B-CL**: Orthogonal training only
- **NLNB-CL**: Neither (baseline)
- **LB-CL**: Both components

Results on Standard Benchmark:

- L-CL: 73.6% avg
- B-CL: 74.3% avg
- NLNB-CL: 74.5% avg
- **LB-CL: 76.7% avg**

Key insights:

- Both components help individually
- But combination gives best results
- Neither component alone is sufficient
- Synergy between initialization and orthogonality

Conclusion: Need both good starting point AND careful training!

Analysis: Initialization Strategies

- **Question**: Should we use full triplets (U, Σ, V) or just (U, V)?
- **Two strategies compared**:
  1. **With Σ**: Use full triplets from old tasks
  2. **Without Σ**: Use only U and V vectors
- **Results**:
  - "Without Σ" shows better average and stability across orders
  - "With Σ" has peak performance in some orders
  - Both strategies outperform O-LoRA
- **Trade-off**:
  - With Σ: Better represents important subspaces (used in paper)
  - Without Σ: More robust and consistent

Analysis: How Many Seed Samples?

- **Purpose**: Seed samples from new task used to compute sensitivity of old tasks
- **Experiment**: Vary number of seed samples (1, 2, 4, 8, 16, 32, 64)
- **Results**:
  - Performance improves gradually with more samples
  - Diminishing returns after 8 samples
  - Lower variance with 4-8 samples
- **Choice**: 8 seed samples selected as optimal
  - Good balance of performance and efficiency
  - Stable and reliable sensitivity estimates
  - Minimal computational overhead
- **Practical note**: Very few samples needed for effective knowledge transfer!

Analysis: Which Layers Matter Most?

- **Question**: Where is task-specific knowledge located in the model?
- **Method**: Analyze sensitivity scores and Fisher information across layers
- **Findings**:
  - **Higher-level layers** (closer to output) show highest sensitivity
  - Especially important: **Top 3-4 decoder layers**
  - Encoder layers less sensitive than decoder layers
  - Consistent pattern across different task orders
- **Practical implication**:
  - Can focus computation on high-level layers
  - Reduces computational cost significantly
  - Still captures most important knowledge
- **Validation**: Fisher information confirms same pattern

Analysis: Impact of Rank r

- **Question**: Does adapter rank r affect performance?
- **Ranks tested**: $r = 2, 4, 8, 16$
- **Results on Standard Benchmark**:

Rank	Order 1	Order 2	Order 3	Average
$r = 2$	76.7	77.2	75.2	76.3
$r = 4$	77.0	76.8	75.9	76.6
$r = 8$	76.9	76.5	76.8	76.7
$r = 16$	77.4	76.0	75.5	76.3
Std	0.25	0.44	0.60	0.18

- **Key insight**: Performance remarkably stable across ranks!
- Small standard deviation (0.18) shows consistency
- Method works well without extensive rank tuning

Analysis: Impact of Model Scale

- **Question**: Does method scale to larger models?
- **Models compared**: T5-base vs T5-large

Model	Method	Order 1	Order 2	Order 3	Average
T5-base	O-LoRA	72.9	72.3	72.6	72.6
	LB-CL	**73.8**	**74.4**	72.4	**73.5**
T5-large	O-LoRA	74.9	75.3	75.9	75.4
	LB-CL	**76.9**	**76.5**	**76.8**	**76.7**

- LB-CL outperforms O-LoRA on both model sizes
- **Larger improvement on T5-large**: +1.3% vs +0.9% on T5-base
- Method scales effectively to larger models
- More consistent across orders in larger model

Computational Considerations

- **GPU Memory**:
  - O-LoRA: 24.82 GB
  - LB-CL: 28.28 GB
  - Modest increase (14% more)
- **Training Parameters per Task**:
  - O-LoRA: $r(m+n)$ parameters
  - LB-CL: $r(m+n)+r$ parameters
  - Nearly identical (difference is just r singular values)
- **Additional Computation**:
  - Sensitivity score calculation: One-time per task with 8 seed samples
  - Orthogonal gradient projection: Minimal overhead during training
  - Focus on high-level layers reduces cost
- **Trade-off**: Slight increase in memory for significant performance gain

Summary and Key Takeaways

- **Problem**: Continual learning for LLMs faces forgetting and poor knowledge transfer
- **Solution - LB-CL**: Two-stage approach
  1. Smart initialization via sensitivity-based knowledge extraction
  2. Orthogonal training to prevent interference
- **Key Results**:
  - Outperforms previous SOTA (O-LoRA) by 1.3% on standard benchmark
  - Consistent improvements across task orders and model scales
  - Approaches multi-task learning upper bound
- **Critical Insights**:
  - Both components necessary for best performance
  - High-level layers contain most task-specific knowledge
  - Method robust to rank choice and requires few seed samples

Contributions

1. **Novel Framework**: First to combine parametric knowledge transfer with orthogonal subspace learning for continual learning in LLMs
2. **Strong Empirical Results**: State-of-the-art performance on standard continual learning benchmarks
3. **Thorough Analysis**:
   - Decomposition of generalization error
   - Comprehensive ablation studies
   - Investigation of initialization strategies
   - Layer importance analysis
4. **Practical Insights**:
   - Importance of initialization for low-rank methods
   - Efficient sensitivity estimation with few samples
   - Computational efficiency through layer selection

Limitations and Future Work

Current Limitations:

- Tested primarily on T5 models (encoder-decoder architecture)
- Moderate performance gap remains vs ProgPrompt on long sequences
- Focuses on text classification tasks

Future Directions:

- Extend to decoder-only models (GPT-style architectures)
- Scale to much longer task sequences (50+ tasks)
- Apply to diverse task types (generation, reasoning, multimodal)
- Investigate dynamic rank allocation based on task complexity
- Explore automated layer selection strategies
- Study knowledge transfer patterns across different domains

Thank You!

Questions?