A survey on LoRA for Continual Learning

A simple survey

Yang, Junran[1] Deepseek[2]

[1]School of AI
Tianjin Univeristy

[2]Deepseek Co., Ltd.

Group Meeting, March. 27th

Table of Contents

Table of Contents

Subspace

Lemma (spanning space)

n linearly independent vectors can span an n-dimensional space

The spanning space could be written as

$$\mathscr{F} = \{\boldsymbol{u}_1\beta_1 + \boldsymbol{u}_2\beta_2 + \cdots \boldsymbol{u}_n\beta_n \mid \beta_1, \beta_2, \cdots, \beta_n \in \mathbb{R}\} \tag{1}$$

The elements in the set are the vectors in the space, which is also the elements that can be linearly represented by $(\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots \boldsymbol{u}_n)$.

Projection Matrix

given a direction vector $\boldsymbol{u}$ s.t. $\|\boldsymbol{u}\| = 1$, a random vector $\boldsymbol{x}$, the formula

$$\mathbf{proj}_u(\boldsymbol{x}) = (\boldsymbol{u}^T\boldsymbol{x})\boldsymbol{u}, \tag{2}$$

$$\text{where} \quad \boldsymbol{u}^T\boldsymbol{x} \in \mathbb{R}, \boldsymbol{u} \in \mathbb{R}^n, \boldsymbol{x} \in \mathbb{R}^n \tag{3}$$

represents $\boldsymbol{x}$'s projection on $\boldsymbol{u}$. Since $(\boldsymbol{u}^T\boldsymbol{x})$ is scalar, Equation (2) is commutative and could be written as:

$$\mathbf{proj}_u(\boldsymbol{x}) = \boldsymbol{u}\boldsymbol{u}^T\boldsymbol{x} \tag{4}$$

$$= \boldsymbol{P}\boldsymbol{x} \tag{5}$$

$$\text{where} \quad \boldsymbol{P} = \boldsymbol{u}\boldsymbol{u}^T, \quad \boldsymbol{P} \in \mathbb{R}^{n\times n} \tag{6}$$

$\boldsymbol{P}$ is the **Projection Matrix**

Projection Matrix

now $\boldsymbol{M} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n]$, $\boldsymbol{x} = [x_1, x_2, \cdots x_n]$, $\boldsymbol{M} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{u}$ are all orthonormal.

$$\boldsymbol{M}^T \boldsymbol{x} = \begin{bmatrix} \boldsymbol{u}_1^T \boldsymbol{x} = \alpha_1 \\ \boldsymbol{u}_2^T \boldsymbol{x} = \alpha_2 \\ \vdots \\ \boldsymbol{u}_n^T \boldsymbol{x} = \alpha_n \end{bmatrix} \tag{7}$$

means $\boldsymbol{x}$'s projection on every vectors in $\boldsymbol{M}$, then

$$\boldsymbol{M}\boldsymbol{M}^T \boldsymbol{x} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n] * \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} \tag{8}$$

$$= \boldsymbol{u}_1 \alpha_1 + \boldsymbol{u}_2 \alpha_2 + \cdots \boldsymbol{u}_n \alpha_n \tag{9}$$

which is a vector in the subspace spanned by the vectors in $\boldsymbol{M}$.

Projection Matrix

a simple example: 3-D vector projected into a plane spanned by 2 3-D vector.

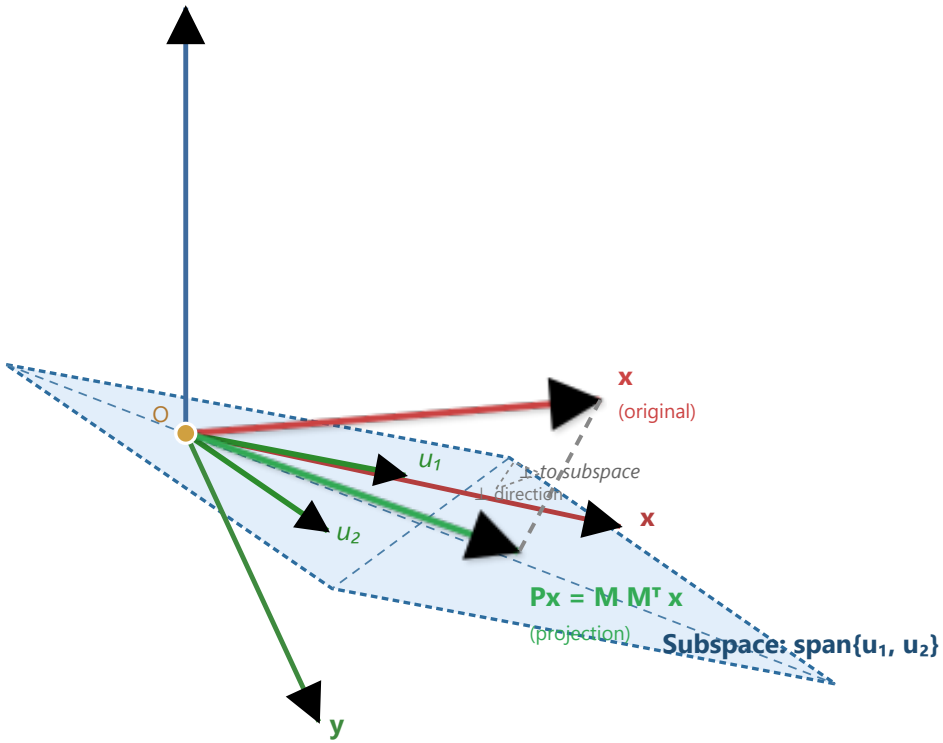

Figure: vector projection in subspace

Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a fundamental matrix factorization technique in linear algebra. For any real (or complex) matrix $\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{m} \times \boldsymbol{n}}$, SVD decomposes it as:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{10}$$

where:

- $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix whose columns are called left singular vectors,
- $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix whose columns are called right singular vectors,
- $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix containing non-negative singular values.

Singular Value Decomposition (SVD)

Rewritten in vectors

$$\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_m], \quad \boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n], \quad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{11}$$

$$\boldsymbol{A} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_m] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^T \\ \boldsymbol{v}_2^T \\ \vdots \\ \boldsymbol{v}_n^T \end{bmatrix} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \tag{12}$$

, where the rank of $\boldsymbol{u}_i \boldsymbol{v}_i^T$ is 1

Low-rank Approximation

assume that $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_r$ the best rank-k approximation A_k for $\boldsymbol{A} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ is

$$\boldsymbol{A_k} = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \tag{13}$$

Gradient of Matrix

$$L = \frac{1}{2}\|\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y}\|_2^2 = \frac{1}{2}(\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y})^T(\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y}) \tag{14}$$

$$\nabla_W L = (\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y})\boldsymbol{x}^T \tag{15}$$

Introduction

highlights

- neither store old gradient directions
- nor store old examples for generating reference directions
- partitions the entire gradient space of the weights into two orthogonal subspaces: Core Gradient Space (CGS) and Residual Gradient Space (RGS)

Methodology

Task 1

1. training one task 1 normally, obtain a set of parameters $\{\mathbb{W}_1^l\}_{l=1}^L$
2. randomly samples n_s input data for each layers $[\boldsymbol{x}_{1,1}^l, \boldsymbol{x}_{2,1}^l, \cdots \boldsymbol{x}_{n_s,1}^l]$
3. construct representation matrix $\boldsymbol{R}_1^l = [\boldsymbol{x}_{1,1}^l, \boldsymbol{x}_{2,1}^l, \cdots \boldsymbol{x}_{n_s,1}^l]$, perform SVD on $\boldsymbol{R}_1^l = \boldsymbol{U}_1^l \boldsymbol{\Sigma}_1^l (\boldsymbol{V}_1^l)^T$ followed by its k-rank approximation $(R_1^l)_k$ according to the given threshold ϵ_{th}^l

$$\|(\boldsymbol{R}_1^l)_k\|_F^2 \geq \epsilon_{th}^l \|\boldsymbol{R}_1^l\|_F^2 \tag{16}$$

$(\boldsymbol{U}_1^l)_k = [\boldsymbol{u}_{1,1}^l, \boldsymbol{u}_{2,1}^l, \cdots, \boldsymbol{u}_{k,1}^l] = \boldsymbol{M}^l$, where $\boldsymbol{M}^l$ is the projection matrix would be used later.

Task 2 and successive tasks

On Initialization:

$$\boldsymbol{W}_2^l \leftarrow \boldsymbol{W}_1^l \tag{17}$$

Before taking gradient step, bases of the CGS are retrieved from GPM

$$\nabla_{\boldsymbol{W}_2^l} L_2 \leftarrow \nabla_{\boldsymbol{W}_2^l} L_2 - \nabla_{\boldsymbol{W}_2^l} L_2 \boldsymbol{M}^l (\boldsymbol{M}^l)^T \tag{18}$$

such that

$$\hat{\boldsymbol{W}}_2^l \boldsymbol{x}_{.,1} = (\boldsymbol{W}_2^l - \epsilon \nabla_{\boldsymbol{W}_2^l} L_2) \boldsymbol{x}_{.,1} = \boldsymbol{W}_1^l \boldsymbol{x}_{.,1} \tag{19}$$

Task 2 and successive tasks

We samples from task 2 input data $\boldsymbol{x}_{.,2}$, and Construct $\boldsymbol{R}_2^l = [\boldsymbol{x}_{1,2}^l, \boldsymbol{x}_{2,2}^l, \cdots \boldsymbol{x}_{n_s,2}^l]$. Also, we perform SVD again to expand the subspace.

$$\hat{\boldsymbol{R}}_2^l = \boldsymbol{R}_2^l - \boldsymbol{M}^l(\boldsymbol{M}^l)^T(\boldsymbol{R}_2^l) = \boldsymbol{R}_2^l - \boldsymbol{R}_{2,\text{Proj}}^l \tag{20}$$

$$\hat{\boldsymbol{R}}_2^l = (\hat{\boldsymbol{U}}_2^l \hat{\boldsymbol{\Sigma}}_2^l (\hat{\boldsymbol{V}}_2^l)^T) \tag{21}$$

and update $\boldsymbol{M}^l$

$$\boldsymbol{M}^l \leftarrow [\boldsymbol{M}^l, \boldsymbol{u}_{1,2}^l, \boldsymbol{u}_{2,2}^l, \cdots, \boldsymbol{u}_{k,2}^l] \tag{22}$$

Experimental Setup

Benchmarks. PMNIST (10 tasks), Split CIFAR-100 (10 tasks), Split miniImageNet (20 tasks), and 5-Datasets.
Backbones. 2-layer MLP for PMNIST; 5-layer AlexNet for Split CIFAR-100; reduced ResNet18 for miniImageNet and 5-Datasets.
Protocol. PMNIST uses *single-head* inference without task identity; other benchmarks use *multi-head* inference.
Baselines. OGD, OWM, GEM, A-GEM, ER-Reservoir, EWC, HAT, plus a multitask upper bound.
Training & Metric. Models are trained with SGD. Main metrics are average accuracy (ACC) and backward transfer (BWT), where more negative BWT means more forgetting.

Results and Discussion

Method	PMNIST		CIFAR-100		miniImageNet		5-Datasets	
	ACC	BWT	ACC	BWT	ACC	BWT	ACC	BWT
OWM	90.71	-0.01	50.94	-0.30	–	–	–	–
EWC	89.97	-0.04	68.80	-0.02	52.01	-0.12	88.64	-0.04
HAT	–	–	72.06	-0.00	59.78	-0.03	91.32	-0.01
A-GEM	83.56	-0.14	63.98	-0.15	57.24	-0.12	84.04	-0.12
ER Res	87.24	-0.11	71.73	-0.06	58.94	-0.07	88.31	-0.04
GPM	**93.91**	**-0.03**	**72.48**	**-0.00**	**60.41**	**-0.00**	91.22	-0.01

- GPM achieves the best ACC on PMNIST, CIFAR-100, and miniImageNet, while staying competitive on 5-Datasets.
- Forgetting is consistently very small: nearly zero BWT on CIFAR-100 and miniImageNet.
- The paper also reports lower memory use than replay-based methods and fast training; e.g., total training time is 245s on PMNIST, 770s on CIFAR-100, 3387s on miniImageNet, and 5008s on 5-Datasets.

Table of Contents

Introduction

Novelty

- expand new LoRA branch to learn new task
- introduce gating modules to integrate new and old LoRA branches
- minimize the influence from the new LoRA branch to old tasks

Methodology

Architecture of Gating Module The formula for the gating module of the i^{th} task could be written as:

$$\boldsymbol{p}_{i,0} = \boldsymbol{p}_0 = \text{Pool}(\text{Token}(\boldsymbol{x})), \tag{23}$$

$$\boldsymbol{p}_{i,l} = \sigma\left(\boldsymbol{G}_{i,l}\boldsymbol{p}_{i,l-1}\right), \quad l \in \{1, 2, \ldots, L\}, \tag{24}$$

$$g_i(\boldsymbol{x}) = f\left(\boldsymbol{G}_{i,L+1}\boldsymbol{p}_{i,L}\right). \tag{25}$$

$\boldsymbol{G_{i,l}} \in \mathbb{R}^{n\times n}, \quad \boldsymbol{G_{i,L+1}} \in \mathbb{R}^{n}, \quad \boldsymbol{p} \in \mathbb{R}^{n}, \quad f : \mathbb{R} \to [0, 1]$

We hope that for $\boldsymbol{x} \in \mathscr{T}_i$, $g_i(\cdot) = 1$, for $\boldsymbol{x} \notin \mathscr{T}_i$, $g_i(\cdot) = 0$

Methodology

Pipeline

1. training on task 1 freely.
2. initialize the gating module on task 2 with some constraints, update the gating module also with some constraints.
3. repeat the procedure in task 2 for the following tasks.

Constraint on Initialization for Task 2 and the Successive

$$\text{Init}(\boldsymbol{G}_{t,l}) \leftarrow \boldsymbol{G}_{t-1,l},\ l \in [1, L] \tag{26}$$

$$\text{Init}(\boldsymbol{G}_{t,L+1}) \leftarrow \text{Init}(\boldsymbol{G}_{t,L+1}) - \boldsymbol{M}_{t,L+1}\boldsymbol{M}_{t,L+1}^{T}\,\text{Init}(\boldsymbol{G}_{t,L+1}) \tag{27}$$

such that

$$\boldsymbol{M}_{t,L+1}^{T}(\text{Init}(\boldsymbol{G}_{t,L+1}) - \boldsymbol{M}_{t,L+1}\boldsymbol{M}_{t,L+1}^{T}\,\text{Init}(\boldsymbol{G}_{t,L+1})) \tag{28}$$

$$= \boldsymbol{M}_{t,L+1}^{T}(\boldsymbol{I} - \boldsymbol{M}_{t,L+1}\boldsymbol{M}_{t,L+1}^{T})\,\text{Init}(\boldsymbol{G}_{t,L+1}) \tag{29}$$

$$= (\boldsymbol{I} - \boldsymbol{M}_{t,L+1}^{T}\boldsymbol{M}_{t,L+1})\boldsymbol{M}_{t,L+1}^{T}\,\text{Init}(\boldsymbol{G}_{t,L+1}) \tag{30}$$

$$= \boldsymbol{0} \tag{31}$$

Constraint on Updating

$$\Delta \boldsymbol{G}_{t,l} \leftarrow \Delta \boldsymbol{G}_{t,l} - \boldsymbol{M}_{t,l}\boldsymbol{M}_{t,l}^{T} \Delta \boldsymbol{G}_{t,l}, \quad l \in [1, L+1] \tag{32}$$

No Constraint on LoRA parameters, only on gating modules.

Experiment

Benchmarks and Settings

- Evaluated on two continual learning benchmarks: **SuperNI** and **Long Sequence**.
- Each benchmark contains **15 sequential tasks**, and two different task orders are considered.
- Evaluation metrics are **Average Performance (AP)** and **Forgetting (FT)**.
- Experiments are conducted on multiple backbones, including **T5-Large, T5-XL, Llama-2-7B/13B, and Llama-3-8B**.

Main Results

- GainLoRA consistently outperforms strong LoRA-based CL baselines such as **O-LoRA** and **InfLoRA**.
- On **T5-Large**, GainLoRA substantially improves AP and reduces FT across all task orders.
- The same advantage remains when scaling to larger backbones, showing good **robustness across model sizes**.

Experiment

Method	Order 1		Order 2		Order 3		Order 4	
	AP↑	FT↓	AP↑	FT↓	AP↑	FT↓	AP↑	FT↓
O-LoRA	26.37	19.15	32.83	11.99	70.98	3.69	71.21	4.03
GainLoRA (O-LoRA)	**47.84**	**2.26**	**46.84**	**2.91**	73.37	3.02	76.01	2.49
InfLoRA	39.78	7.64	39.57	8.93	75.15	4.19	75.79	3.47
GainLoRA (InfLoRA)	46.21	2.40	46.44	2.61	**78.01**	**0.77**	**77.54**	**1.25**

Observation: GainLoRA consistently improves *average performance* and reduces *forgetting* over the corresponding LoRA-based continual learning baselines.

Conclusion

Takeaway

- GainLoRA achieves **better overall continual learning performance** than existing state-of-the-art methods.
- Its advantages are consistent across **different task orders, benchmarks, and backbone scales**.
- Therefore, the paper shows that **how to integrate LoRA branches** is a key factor in continual learning of LLMs.

Limitations

- The imposed constraints may **accumulate as the number of tasks increases**, which can hinder future learning.
- The method is mainly evaluated on **non-overlapping task settings**; its behavior in more complex scenarios with **task overlap or blurry boundaries** still needs further study.